matcos-10

Proceedings of the 2010 Mini-Conference on Applied Theoretical Computer Science

University of Primorska Press

MATCOS-10

Proceedings of the 2010 Mini-Conference on Applied Theoretical Computer Science

Edited by

Andrej Brodnik and Gábor Galambos

Reviewers and Programme Committee

Andrej Brodnik, *University of Primorska and University of Ljubljana, Slovenia (co-chair)*
Gábor Galambos, *University of Szeged, Hungary (co-chair)*
Gabriel Istrate, *Universitatea Babes Bolyai, Cluj, Romania*
Miklós Krész, *University of Szeged, Hungary*
Gerhard Reinelt, *Ruprecht-Karls-Universität, Heidelberg, Germany*
Borut Robič, *University of Ljubljana, Slovenia*
Magnus Steinby, *University of Turku, Finland*
Borut Žalik, *University of Maribor, Slovenia*

Organizing Committee

David Paš, chair
Iztok Cukjati
Milan Djordjević
Andrej Kramar
Tine Šukljan

Published by
University of Primorska Press
Titov trg 4, 6000 Koper

Koper 2011

Editor in Chief
Jonatan Vinkler

Managing Editor
Alen Ježovnik

CIP – Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

004(082)(0.034.2)

MINI-Conference on Applied Theoretical Computer Science (2010 ; Koper)
MATCOS-10 [Elektronski vir] : proceedings of the 2010 Mini-Conference on Applied Theoretical Computer Science / edited by Andrej Brodnik and Gábor Galambos. – El. knjiga. – Koper : University of Primorska Press, 2011

Način dostopa (URL):
http://www.hippocampus.si/ISBN/978-961-6832-10-6.pdf

ISBN 978-961-6832-10-6 (pdf)
1. Gl. stv. nasl. 2. Brodnik, Andrej
258820352

Preface

The collaboration between the University of Primorska and the University of Szeged retrospect only a short period. Even so, there is a couple of research interest where the two groups can collaborate. The MATCOS-10 (Mini-Conference on Applied Theoretical Computer Science) conference started from the idea that we collect those results, which we have in our joint projects. We wanted to organize a mini-conference where we expected papers from several streams of operation research a broader. We invited papers, which are dealing with those methods from the theoretical computer science, which have been integrated into real world applications. Practical solutions for NP-hard problems, algorithmic oriented AI and data mining solutions, new models and methods in system biology and bioinformatics, automata theory solutions in software and hardware verification, prospective new models of computation and future computer architecture are those topics which were preferred in the picking.

We had also a second aim with this conference: we wanted to give a chance for the young researchers to lecture on their results. Therefore we organized a special Student Session. We encouraged regular and PhD students to take part on this conference in order to get the first impression about the flavour of a conference.

The MATCOS-10 conference was held on October 13–14, 2010 at the University of Primorska in Koper, in conjunction with the 13th Multi-Conference on Information Society, October 11–15, 2010, Ljubljana, Slovenia.

In response of call of papers we received 38 submissions. Each submission was reviewed by at least two referees. Based on the reviews, the Program Committee selected 23 papers for presentation in these proceedings. In addition to the selected papers, András Recski from the TU Budapest gave an invited talk on 'Applications of Combinatorics in Statics.'

We used the EasyChair system to manage the submissions. Thanks to David Paš, chair of the Organizing Committee for maintaining the on-line background of the conference. We are also grateful to Miklós Krész, who – besides giving a great activity in the Organizing Committee – managed the homepage of the conference.

Having elated from the success of the MATCOS-10 we decided to organize the conference regularly. See you on the next MATCOS conference in Szeged!

Andrej Brodnik and Gábor Galambos

Invited Paper

Student Papers

Regular Papers

Rigidity of Frameworks: Applications of Combinatorial Optimization to Statics

András Recski
Budapest University of Technology and Economics
H-1521 Budapest, Hungary
+36-1-4632585
recski@cs.bme.hu

ABSTRACT
The survey presents various applications of graphs and matroids to the rigidity of classical and tensegrity frameworks.

Categories and Subject Descriptors
F.2.2 Computations on Discrete Structures

General Terms
Algorithms

Keywords
Graphs, matroids, frameworks, rigidity, structural topology

1. INTRODUCTION
Apart from the last section, we shall consider only frameworks, composed of rigid rods and rotatable joints. Such a framework may be rigid or nonrigid, see Figure 1. Rigidity depends on the dimension of the space as well: the framework of Figure 2 is rigid in the plane but not in the space. A nonrigid framework need not necessarily be a "mechanism", rigidity also excludes "infinitesimal" motions like those at Figure 3. Hence our "rigidity" is called "infinitesimal rigidity" by most authors.

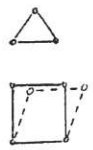


Figure 1

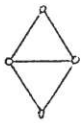


Figure 2

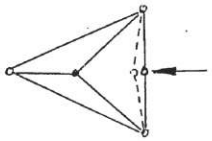


Figure 3

2. RIGIDITY OF A GIVEN FRAMEWORK
Deciding rigidity of a framework F can be formulated as a problem of linear algebra. Rigidity of a rod e_{ij}, connecting joints v_i and v_j with respective coordinates $(x_i, y_i,\ldots)$ and $(x_j, y_j,\ldots)$ means that

$$(x_i - x_j)^2 + (y_i - y_j)^2 + \ldots = \text{constant}.$$

Its derivative is

$$(x_i - x_j)dx_i/dt + (x_j - x_i)dx_j/dx_t + (y_i - y_j)dy_i/dt + (y_j - y_i)dy_j/dt + \ldots = 0.$$

All these equations together give $\mathbf{A}_F\,\mathbf{u}=\mathbf{0}$, where the number of rows of the matrix $\mathbf{A}_F$ is the number e of the rods, the number of columns of $\mathbf{A}_F$ is the number v of the joints multiplied by the dimension d of the space and $\mathbf{u}=(dx_1/dt,\ dx_2/dt,\ \ldots..,\ dy_1/dt,\ dy_2/dt,\ \ldots..)^T$. The congruent motions of the d-dimensional space (which form a subspace of dimension $d(d+1)/2$) are all the solutions of this system. A framework F will be called *rigid* in the d-dimensional space if there are no other solutions, that is, if the rank of $\mathbf{A}_F$ is $vd - d(d+1)/2$.

3. DIFFERENT FRAMEWORKS WITH ISOMORPHIC GRAPHS
A framework F can be described by a graph $G(F)$ with v vertices and e edges. However, one of the planar frameworks of Figure 4 and one of the 3-dimensional frameworks of Figure 5 are rigid, the other two are not, showing that $G(F)$ alone determines the zero-nonzero pattern of the matrix $\mathbf{A}_F$ only, but not its rank, hence not the rigidity of F either. Clearly, if two determinants have the same zero-nonzero pattern, one of them may vanish if certain entries cancel out each other.

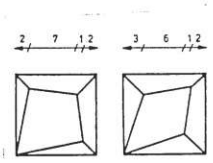


Figure 4

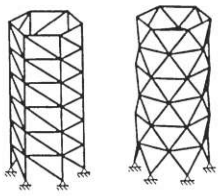


Figure 5

4. WHAT CAN COMBINATORIALISTS DO HERE?
While the examples of Section 3 show that one cannot expect to decide rigidity from the graph alone, combinatorial methods can still help in two cases: either if the framework is "very

symmetric", see Sections 7-8 below, or if it has "no regularity at all". This latter means that no cancellation may arise among the entries of $\mathbf{A}_F$ unless it is reflected by the structure of $G(F)$. Formally we call a graph G (not the framework!) *generic rigid* in dimension d if there exists a rigid d-dimensional framework F with $G(F)=G$. Alternatively, G is generic rigid in dimension d if and only if realizing it in the d-dimensional space so that the coordinates of the joints are algebraically independent over the field of the rational numbers, the resulting framework is rigid. For example, the graphs of the frameworks of Figures 4 and 5 are generic rigid (in 2- and 3-dimension, respectively) while the graph of the second framework of Figure 1 is not.

5. GENERIC RIGID GRAPHS IN THE PLANE

Deciding generic rigidity of any graph in any dimension is in **NP** (if there are rigid realizations, there are some with "small" integer coordinates as well). The problem is in **P** for dimension 2 and its status is unknown for dimension ≥ 3. For simplicity we restrict ourselves to minimum generic rigid graphs in dimension 2. Then, with the notation of Section 2, $e = 2v - 3$ and clearly $e' \leq 2v' - 3$ for any "subsystem" with e' rods among v' joints to avoid situations like that of Figure 6 where a part of the system is "overbraced". This stronger necessary condition, known to Maxwell [14] already in 1864, was proved to be sufficient (Laman [12]). This leads to a polynomial algorithm by observing (Lovász and Yemini, [13]) that the necessary and sufficient condition was equivalent to the property that doubling any edge of the graph the resulting graph with $2v-2$ edges is the union of two trees. This condition can be checked in polynomial time, using the matroid partition algorithm of Edmonds [9]. The analogue of Maxwell's condition in dimension 3 can also be reformulated with tree decompositions (Recski [15]) but this is only necessary. Figure 7 shows the smallest counterexample.

Figure 6

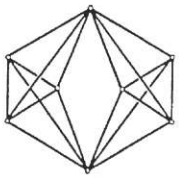


Figure 7

6. RECONSTRUCTION OF POLYHEDRA

Apart from obvious applicability in structural engineering, rigidity has an interesting relation to pattern reconstruction.

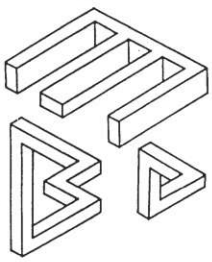


Figure 8

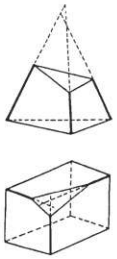


Figure 9

Suppose we wish to decide from a 2-dimensional drawing whether it is the projection of a polyhedron. The reason of a negative answer may follow from the graph of the drawing already (drawings like those in Figure 8 are well known) but sometimes the answer is positive only if some additional conditions are met (like the three dotted lines meet at the same point in Figure 9). Under some conditions a drawing arises as the projection of a 3-dimensional polyhedron if and only if it is nonrigid as a 2-dimensional framework. For example, Figure 10 shows that the second framework of Figure 4 is rigid while the first one is not

Figure 10

7. RIGIDITY OF SQUARE GRIDS

Another area of combinatorial applications arises if we restrict our attention to "very regular" frameworks. For example, the $k \times l$ square grid as a planar framework requires at least $k + l - 1$ diagonal rods to become rigid and the minimum systems of such diagonals correspond to the spanning trees of the complete bipartite graph $K_{k,l}$ with $k + l$ vertices (Bolker and Crapo, [5]). For example, Figure 11 shows a 3×3 grid with a possible deformation – the corresponding graph is disconnected.

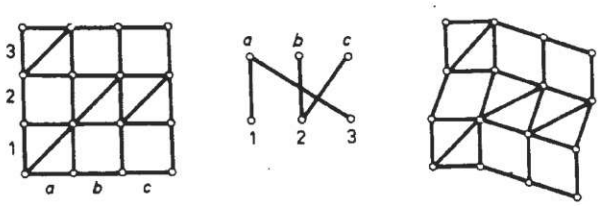


Figure 11

8. RIGIDITY OF A 1-STORY BUILDING

Consider a 1-story building, with the vertical bars fixed to the earth via joints. If each of the four external vertical walls consists of a diagonal then the four corners of the roof become fixed. Hence questions related to the rigidity of a one-story building are reduced (Bolker and Crapo, [5]) to those related to the rigidity of a 2-dimensional square grid of size $k \times l$ where the corners are pinned down. Then the minimum number of necessary diagonal rods for infinitesimal rigidity was proved to be $k + l - 2$ (Bolker and Crapo, [5]) and the minimum systems correspond to asymmetric 2-component forests (Crapo, [7]). A 2-component forest is *asymmetric* if the ratio of the cardinalities of the bipartition sets in the two components is different from k:l. For example, parts (a) and (b) of Figure 12 show two systems with the corresponding 2-component forests. The second system is nonrigid, an infinitesimal deformation is shown in part (c) of the figure. Observe that the points u, v, w are collinear in Figure 12(b), this can always be achieved after appropriate permutations of the row set and of the column set, if the 2-component forest is

symmetric.

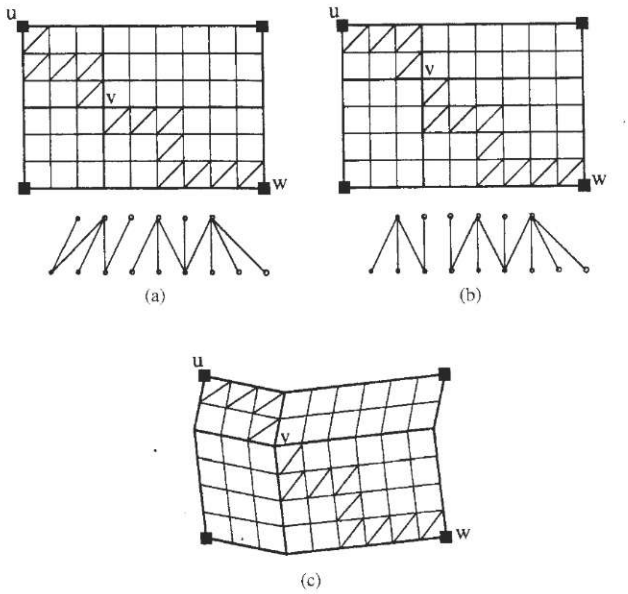


Figure 12

9. TENSEGRITY FRAMEWORKS

Since real life rods are less reliable against compression than against tension, we might prefer using more diagonal rods but under tension only. In order to study such questions, the concept of *tensegrity frameworks* has been introduced: the elements connecting the joints may be rods (which are rigid both under tension and compression), cables (used under tension only) and possibly struts as well (used under compression only). Then the diagonal tensegrity elements will correspond to directed edges and a collection of such elements will rigidify the square grid if and only if the corresponding subgraph is strongly connected (Baglivo and Graver, [2]), The corresponding questions for 1-story buildings are more complicated and need various tools of matching theory and network flow techniques (Chakravarty et al, [6], Recski and Schwärzler, [20]). More recently, some of the results described in Section 5 have also been generalized for tensegrity frameworks (Jordán et al,.[11], Recski, [21], Recski and Shai, [22]).

10. ACKNOWLEDGEMENTS

The support of the Hungarian National Science Foundation and the National Office for Research and Technology (Grant number OTKA 67651) is gratefully acknowledged.

11. REFERENCES

[1] L. Asimow and B. Roth, 1978-79. The rigidity of graphs, *Trans. Amer. Math. Soc.* 245, 279-289 (Part I) and *J. Math. Anal. Appl.* 68, 171-190 (Part II).

[2] J. A. Baglivo and J. E. Graver, 1983. *Incidence and symmetry in design and architecture,* Cambridge University Press, Cambridge .

[3] E. Bolker, 1977. Bracing grids of cubes, *Env. Plan. B*, 4, 157-172.

[4] E. Bolker, 1979. Bracing rectangular frameworks II. *SIAM J. Appl. Math.* 36, 491-508.

[5] E. Bolker and H. Crapo, 1979. Bracing rectangular frameworks, *SIAM J. Appl. Math.* 36, 473-490.

[6] N. Chakravarty, G. Holman, S. McGuinness and A. Recski, 1986. One-story buildings as tensegrity frameworks, *Struct. Topology*, 12, 11-18.

[7] H. Crapo, 1977. More on the bracing of one-story buildings, *Env. Plan. B,* 4, 153-156.

[8] H. Crapo, 1979. Structural rigidity, *Struct. Topology* 1, 26-45.

[9] J. Edmonds, 1968. Matroid partition, *Math. of the Decision Sciences, Part I., Lectures in Appl. Math.*, 11, 335-345.

[10] G. Hetyei, 1964. On covering by 2X1 rectangles, *Pécsi Tanárképző Főisk.Közl.* 151-168.

[11] T. Jordán, A. Recski and Z. Szabadka, 2009. Rigid tensegrity labelling of graphs, *European J. of Combinatorics* 30, 1887-1895.

[12] G. Laman, 1970. On graphs and rigidity of plane skeletal structures, *J. of Eng. Math.* 4, 331-340.

[13] L. Lovász and Y. Yemini, 1982. On generic rigidity in the plane, *SIAM J. Alg. Discrete Methods*, 3, 91-98.

[14] J. C. Maxwell, 1864. On the calculation of the equilibrum and stiffness of frames, *Philos. Mag.* (4), 27, 294-299.

[15] A. Recski, 1984. A network theory approach to the rigidity of skeletal structures II. Laman's theorem and topological formulae, *Discrete Applied Math.* 8, 63-68.

[16] A. Recski, 1987. Elementary strong maps of graphic matroids, *Graphs and Combinatorics*, 3, 379-382.

[17] A. Recski, 1988/89. Bracing cubic grids – a necessary condition, *Discrete Math.*, 73, 199-206.

[18] A. Recski, 1989. *Matroid theory and its applications in electric network theory and in statics*, Springer, Berlin-Heidelberg-New York; Akadémiai Kiadó, Budapest.

[19] A. Recski, 1991. One-story buildings as tensegrity frameworks II, *Struct. Topology* 17, 43-52.

[20] A. Recski and W. Schwärzler, 1992. One-story buildings as tensegrity frameworks III, *Discrete Applied Math.*, 39, 137-146.

[21] A. Recski, 2008. Combinatorial conditions for the rigidity of tensegrity frameworks, *Bolyai Society Mathematical Studies* 17, 163-177.

[22] A. Recski and O. Shai, 2010.Tensegrity frameworks in the one-dimensional space, *European Journal of Combinatorics* 31, 1072-1079.

[23] B. Roth and W- Whiteley, 1981. Tensegrity frameworks, *Trans. Amer. Math. Soc.* 265, 419-446.

[24] T. Tarnai, 1980. Simultaneous static and kinematic indeterminacy of space trusses with cyclic symmetry, *Int. J. Solids and Structures,* 16, 347-359.

[25] W. Whiteley, 1988. The union of matroids and the rigidity of frameworks, *SIAM J. Discrete Math.* 9, 237-255.

Succinct data structures for dynamic strings

Tine Sukljan
FAMNIT
Univerza na Primorskem
Koper, Slovenia
tine.sukljan@upr.si

ABSTRACT

The biggest problem of full-text indexes is their space consumption. In the last few years many efforts has been put in the research of these indexes. At first the idea was to exploit the compressibility of the text, so that the size of the index would be more efficient. Recently this idea even more evolved in the so called *self-indexes*, which stores enough information to replace the original text. This idea of an index which takes space close to that of the compressed text, replaces it and provides fast search over it was the source for many new research done in the last few years. In this article we present the most recent result done in this area of research.

1. INTRODUCTION

Succinct data structures provide solutions to reduce the storage cost of modern applications that process large data sets, such as web search engines, geographic information systems, and bioinformatics applications. First proposed by Jacobson [8], the aim is to encode a data structure using space close to the information-theoretic lower bound, while supporting efficient navigation operations in them.

We are given a sequence S of length n over an alphabet Σ of size σ. We can use operation $\texttt{access}(S, i)$ to read a symbol at position i of the sequence S $(0 \leq i < n)$. Given the specified parameters we want to support two more query operation for a symbol $c \in \Sigma$ [3]:

- $\texttt{select}_c(S, j)$: return the position of the jth occurrence of symbol c, or -1 if that occurence does not exists;
- $\texttt{rank}_c(S, p)$: count how many occurrences are found in $S[0, p-1]$.

In many situations only retrieving data (even though efficiently) is not enough, as data in these situations are updated frequently. In this case two additional operations are desired:

- $\texttt{insert}_c(S, i)$: insert character c between $S[i-1]$ and $S[i]$;
- $\texttt{delete}(S, i)$: delete $S[i]$ from S.

2. RANK AND SELECT OPERATIONS ON DYNAMIC COMPRESSED SEQUENCES

Gonzalez et al. in [3] presented a data structure with $nH_0 + o(n \log \sigma)$ bits of space and $O(\log n(1 + \frac{\log \sigma}{\log \log n}))$ worst-case time.

We first present a data structure for the *Collection of Searchable Partial Sums with Indels (CSPSI)* problem, which will be used later in the article. The problem consists of σ sequences $C = S^1, \ldots, S^\sigma$ of nonnegative integers $s^j_{i\, q \leq j \leq \sigma, 1 \leq i \leq n}$, each of $O(\log n)$ bits with the following operations required:

- $\texttt{sum}(C, j, i)$ is $\sum_{l=1}^{i} s^j_l$;
- $\texttt{search}(C, j, y)$ is the smallest i' such that $\texttt{sum}(C, j, i) \geq y$;
- $\texttt{update}(C, j, i, x)$ updates s^j_i to $s^j_i + x$;
- $\texttt{insert}(C, i)$ inserts 0 between s^j_{i-1} and s^j_i for all $1 \leq j \leq \sigma$;
- $\texttt{delete}(C, i)$ deletes s^j_i from the sequence S^j for all $1 \leq j \leq \sigma$. To perform $\texttt{delete}(C, i)$ it must hold $s^j_i = 0$ for all $1 \leq j \leq \sigma$.

We now present the data structure that Gonzalez et al. introduced in [3]. They construct a red-black tree over C. Each leaf contains a non-empty superblock of size from $\frac{1}{2}\log^2 n$ to $2\log^2 n$ bits. The leftmost leaf contains $s^1_1 \ldots s^1_{b_1} s^2_1 \ldots s^2_{b_1} \ldots s^\sigma_1 \ldots s^\sigma_{b_1}$, the second $s^1_{b_1+1} \ldots s^1_{b_2} s^2_{b_1+1} \ldots s^2_{b_2} \ldots s^\sigma_{b_1+1} \ldots s^\sigma_{b_2}$ and so on. The size of the leaves are variable but bounded. $b_1, b_2, \ldots$ are such that $\frac{1}{2}\log^2 n \leq \sigma k b_1, \sigma k(b_2 - b_1), \ldots \leq 2\log^2 n$. Each internal node v stores two additional type of counters, $p(v)$ and $r^j(v)_{1 \leq j \leq \sigma}$ such that:

- $p(v)$ is the number of positions stored in the left subtree (for any sequence);

- $r^j(v)$ is the sum of the integers in the left subtree for sequence S^j.

They further divided the superblock in the leaves into blocks of $\sqrt{\log n}$ bits. They stored these blocks in a linked list so that it's possible to can scan a leaf block by block. In the paper ([3], Section 2) they describe how to compute the operations required by CSPSI using the data structure we just described. Their result is the following theorem:

THEOREM 1. *The Collection of Searchable Partial Sums with Indels problem with σ sequences of n numbers of k bits can bi solved, in a RAM machine of $w = O(\log n)$ bits, using $\sigma kn(1 + O(\frac{1}{\sqrt{\log n}} + \frac{\sigma}{\log n}))$ bits of space, supporting all the operations in $O(\sigma + \log n)$ worst-case time. Note that, if $\sigma = O(\log n)$ the space is $O(\sigma kn)$ and the time is $O(\log n)$.*

We will omit the details because of space reasons.

We will now look at how the can to solve the rank/select problem with CSPSI. We construct the red-black tree over the original text S. Each leaf contains a non-empty superblock. Each internal node has counters $p(v)$ and $r(v)$. We call a superblock ob size less than $\log^2 n$ sparse. Operations insert and delete will ensure that no two consecutive sparse superblock exists. Because of this rule there are at most $1 + \frac{2n \log \sigma}{\log^2 n}$ superblocks. For each superblock we mantain s_i^j, the number of occurrences of symbol j in superblock i, for $1 \leq j \leq \sigma$. We store all these sequences of numbers using CSPSI as we described earlier. The partial sums operate in $O(\log n)$ worst-case time.

We show now how to compute $access(S, i)$ and $insert_c(S, i)$. The other operations are similar and explained in more details in [3].

`access`(S, i): We traverse the tree to find the leaf containing the i-th position. We start with $sb \leftarrow 1$ and $pos \leftarrow i$. if $p(v) \geq pos$ we enter the left subtree, otherwise we enter the left subtree, otherwise we enter the right subtree with $sb \leftarrow sb + r(v)$ and $pos \leftarrow pos - p(v)$. We reach the leaf that contains the i-th position in $O(\log n)$ time. Then we directly access the pos-th symbol of superblock sb.

`insert`$_c(S, i)$: We obtain sb and pos just like in the access query, except that we start with $pos \leftarrow i - 1$, so as to insert right after position $i - 1$. Then, if superblock sb contains room for one more symbol, we insert c right after the pos-th position of sb, by shifting the symbols through the blocks as explained. If the insertion causes an overflow in the last block of sb, we simply add a new block at the end of the linked list to hold the trailing bits. We also carry out $update(C, c, sb, 1)$ and retraverse the path from the root to sb adding 1 to $p(v)$ each time we go left from v. In this case we finish in $O(\log n)$ time.

If, instead, the superblock is full, we cannot carry out the insertion yet. We first move one symbol to the previous superblock (creating a new one if this is not possible): We first $delete(T, d)$ the first symbol c' from block sb (the global position of c is $d = i - pos$), and this cannot cause an underflow of sb. Now, we check how many symbols does superblock $sb - 1$ have (this is easy by subtracting the pos numbers corresponding to accessing blocks $sb - 1$ and sb). If superblock $sb - 1$ can hold one more symbol, we insert the removed symbol c' at the end of superblock $sb - 1$. This is done by calling $insert_{c'}(T, d)$, a recursive invocation that now will arrive at block $sb - 1$ and will not overflow it (thus no further recursion will occur).

If superblock $sb - 1$ is also full or does not exist, then we are entitled to create a sparse superblock between $sb - 1$ and sb, without breaking the invariant on sparse superblocks. We create such an empty superblock and insert symbol c' into it, using the following procedure: We retraverse the path from the root to sb, updating $r(v)$ to $r(v) + 1$ each time we go left from v. When we arrive again at leaf sb we create a new node μ with $r(\mu) = 1$ and $p(\mu) = 1$. Its left child is the new empty superblock, where the single symbol c' is inserted, and its right child is sb. We also execute $insert(C, sb)$ and $update(C, sb, c', 1)$.

After creating μ, we must check if we need to rebalance the tree. If it is needed, it can be done with $O(.)$ rotations and $O(\log n)$ redâĂŞblack tag updates. After a rotation we need to update $r(\dot{)}$ and $p(.)$ only for one tree node. These updates can be done in constant time. Now that we have finally made room to carry out the original insertion, we rerun $insert_c(T, i)$ and it will not overflow again. The whole insert operation takes $O(\log n)$ time.

To compress we divide every single superblock into segments representing $\lfloor \frac{1}{2} \log_\sigma n \rfloor$ original simbols from S. We then represent each segment using the (c, o)-pair encoding of Ferragina et al. [1]. For alphabet larger than $\sigma = \Omega(\frac{\sqrt{\log n}}{\log \log n})$ we use a ρ-ary wavelet tree [1] over T, where $\rho = \Theta(\frac{\sqrt{\log n}}{\log \log n})$. On each level we store a sequence over alphabet of size ρ, which are handled as described above. This way we get:

THEOREM 2. *Given a text S of length n over an alphabet of size σ and zero-order entropy $H_0(S)$, the Dynamic Sequence with Indels problem under the RAM model with word size $w = \Omega(\log n)$ can be solved using $nH_0(S) + O(\frac{n \log \sigma}{\sqrt{\log n}})$ bits of space, supporting queries access, rank, select, insert and delete in $O(\log n(1 + \frac{\log \sigma}{\log \log n}))$ worst-case time.*

3. IMPROVED RANK AND SELECT ON DYNAMIC STRINGS

Recently He et al. in [6] managed to improve the result of Gonzalez and Navarro. They followed the same idea, but instead of using red-black trees, they used B-tree. Firstly we need to introduce their modification to CSPSI solution. Then we will look at their result.

We construct a B-tree over the collection C. Let $L = \lceil \frac{\lceil \log n \rceil^2}{\log \lceil \log n \rceil} \rceil$. Each leaf of the tree stores a superblock of size between $L/2$ and $2L$ bits. Each superblock stores the same number of integers from each sequence in C. More precisely, the content of the leftmost leaf is $s_1^1 \ldots s_{b_1}^1 s_1^2 \ldots s_{b_1}^2 \ldots s_1^\sigma \ldots s_{b_1}^\sigma$, the second $s_{b_1+1}^1 \ldots s_{b_2}^1 s_{b_1+1}^2 \ldots s_{b_2}^2 \ldots s_{b_1+1}^\sigma \ldots s_{b_2}^\sigma$ and so on.

$b_1, b_2, \ldots$ satisfy the following conditions because of requirement on the sizes of superblocks: $L/2 \leq b_1 k\sigma \leq 2L, L/2 \leq (b_2 b_1)k\sigma \leq 2L, \ldots$ Every internal node v stores some additional counters (let h be the number of children of v):

- A sequence $P(v)[1..h]$, in which $P(v)[i]$ is the number of positions stored in the leaves of the subtree rooted at the i-th child of v for any sequence in C (the naumber is the same for every sequence in C);
- A sequence $R_j(v)[1..h]$ for each $j = 1, 2, \ldots d$, in which $R_j(v)[i]$ is the sum of the integers from sequence S^j that are stored in the leaves of the subtree rooted at the i-th child of v.

We additionally divide each superblock into blocks of $\lceil \lceil \log n \rceil^{\frac{2}{3}} \rceil$ bits each.

The described data structure supports sum, search an update operations in $O(\frac{\log n}{\log \log n})$ time and operations insert and delete in $O(\frac{\log n}{\log \log n})$ amortized time. Proofs are omitted but can be found in [6], Section 3.

We can now turn our interest in the rank/select problem. We construct a B-tree over S. If a superblock has fewer than L bits we called it skinny. The string S is initially partitioned into substrings that are stored in superblocks. We number the superblocks from left to right, so that i-th superblock stores i-th substring. There must not exist two consecutive skinny superblocks (this way we bound the numbers of leaves), which gives us the upper bound of $O(\frac{n \log \sigma}{L})$ leaves.

He et al. chose $b = \sqrt{\log n}$. They additionally required that the degree of each internal node is between b and $2b$. For each internal node v we need to store the following data structures (let h be the number of children of v):

- A sequence $U(v)[1..h]$, in which $U(v)[i]$ is the number of superblocks contained in the leaves of the subtree rooted at the i-th child of v;
- A sequence $I(v)[1..h]$, in which $I(v)[i]$ stores the number of characters stored in the leaves of the subtree rooted at the i-th child of v.

Finally for each character c, we construct an integer sequence $E_c[1..t]$ in which $E_c[i]$ stores the number of occurrences of character c in superblock i. We create σ sequences this way, and we construct a CSPSI structure, E, for them.

Following the approach of Ferragina et al. in [1], He et. al managed to get the following result:

THEOREM 3. *Under the word RAM model with word size $w = \Omega(\log n)$, a string S of length n over an alphabet of size $\sigma = O(\sqrt{\log n})$ can be representd using $nH_0 + O(\frac{n \log \sigma \log \log n}{\sqrt{\log n}}) + O(w)$ bits to support access, rank, select, insert and delete in $O(\frac{\log n}{\log \log n})$ time.*

Note that a bit vector of length n can be represented using $nH_0 + o(n) + O(w)$ bits and still supporting all the operations in $O(\frac{\log n}{\log \log n})$ time.

4. DYNAMIZING SUCCINCT DATA STRUCTURES

Gupta et al. in [5] presented a framework to dynamize succinct data structures. With their framework it is possible to dynamize most succinct data structures for dictionaries, trees, and text collections. We will now present the idea behind this framework and how it compares with other data structures.

The construction of their data structure is based on some static data structures which we will only list:

- For a bitvector (i.e. $|\Sigma| = 2$) of length n, there exists a static structure D called RRR solving the bit dictionary problem supporting rank, select and access queries in $O(1)$ time using $nH_0 + O(n \log \log n / \log n)$ bits of space, while taking $O(n)$ time to construct [10].
- For a text S of length n drawn from alphabet Σ, there exists a astatic data structure D called *wavelet tree* solving the text dictionary problem supporting rank, select and access queries in $O(\log |\Sigma|)$ time, using $nH_0 + o(n \log |\Sigma|)$ bits of space, while taking $O(nH_0)$ time to construct ([4]).
- For a text S of length n drawn from alphabet Σ, there exists a astatic data structure D called GMR that solves the text dictionary problem supporting select queries in $O(1)$ time and rank and access queries in $O(\log \log |\Sigma|)$ time using $n \log |\Sigma| + o(n \log |\Sigma|)$ bits of space, while taking $O(n \log n)$ time to construct ([2]).
- Let $A[1..t]$ be a nonnegative integer array such that $\sum_i A[i] \leq n$. There exists a data structure PS (prefix-sum) on A that supports sum and findsum in $O(\log \log n)$ time using $O(t \log n)$ bits of space and can be constructed in $O(t)$ time.
- A Weight Balanced B-tree (WBB) is a B-tree defined with a weight-balance condition. This means that for any node v at level i, the number of leaves in v's subtree is between $1/2b^i + 1$ and $2b^i - 1$, where b is the fanout factor. Insertion and deleteion can be performed in amortized $O(\log_b n)$ time while maintaining the condition ([10, 7, 5]).

Gupta's solution is based on three main data structures:

- *BitIndel* bitvector supporting insertion and deletion;
- *StaticRankSelect* static text dictionary structure suporting rank, select and access on a text T;
- *onlyX* non-succinct dynamic text dictionary.

StaticRankSelect is used to maintain the original text T (we can use wavelet tree or GMR data structure). We need to upgrade it, so it can support $\texttt{insert}_x(i)$ (insert a symbol $x \notin$

	space(bits)	access, rank and select	insert and delete
Gonzalez et al. [3]	$nH_0 + o(n)\log\sigma$	$O(\log n(\frac{\log\sigma}{\log\log n}+1))$	$O(\log n(\frac{\log\sigma}{\log\log n}+1))$
He et al. [6]	$nH_0 + o(n)\log\sigma$	$O(\frac{\log n}{\log\log n}(\frac{\log\sigma}{\log\log n}+1))$	$O(\frac{\log n}{\log\log n}(\frac{\log\sigma}{\log\log n}+1))$
Gupta et al. [5]	$n\log\sigma + \log\sigma(o(n)+O(1))$	$\log\log n$	$O\log^{\epsilon} n$ amortized

Table 1: A comparison of the results [6]

Σ at position $S[i]$) and `delete`(i) (deletes symbol at position $S[i]$) and we will call this new structure *inX*. We use onlyX to keep track of newly inserted symbols (N). We merge the updates in StaticRankSelect every $O(n^{1-\epsilon}\log n)$ update operations. OnlyX cannot contain more than $O(n^{1-\epsilon}\log n)$ elements, so we can maintain it using $O(n^{1-\epsilon}\log^2 n) = o(n)$ bits. Merging N with T takes $O(n\log n)$ time, so we get an amortized $O(n^{\epsilon})$ time for updating these data structures.

The final data structure is comprised of two main parts: inX and onlyX. After every $O(n^{1-\epsilon}\log n)$ updates onlyX is merged into the original text S. The cost can be amortized to $O(n^{\epsilon})$ per update. The StaticRankSelect structure on S takes $n\log|\Sigma| + o(n\log|\Sigma|)$ bits of space. The other structures takes o(n) bits of space ([5]).

We need to augment the above two structures with a few additional BitIndel structures. For each symbol c, we maintain a bitvector I_c such that $I_c[i] = 1$ if and only if the i-th occurrence of c is stored in the onlyX structure. We now describe how to support rank and select operations with the above structures.

`rank`$_c(i)$: We first find $j = inX.rank_c(i)$ and $k = inX.rank_x(i)$, and return $j + only.rank_c(k)$.

`select`$_c(i)$: We first find whether the i-th occurrence of c belongs to the inX structure or the onlyX structure. If $I_c[i] = 0$, this means that the i-th item is one of the original symbols from S; we query $inX.select_c(j)$ in this case, where $j = I_c.rank_0(i)$. Otherwise we compute $j = I_c.rank_1(i)$ to translate i to it's corresponding position among new symbols. Then we compute $j' = onlyX.select_c(j)$, and return $inX.select_x(j')$.

Gupta et al. in [5] showed how to manatina I_s during updates, and the space complexity of BitIndel data structures and proved the following theorem:

THEOREM 4. *Given a text S of length n drawn from an alphabet Σ, we create a data structure using GMR that takes $n\log|\Sigma| + o(n\log|\Sigma|) + o(n)$ bits of space and supports rank, select and access in $O(\log\log n + \log\log|\Sigma|)$ time and insert and delete updates in $O(n^{\epsilon})$ time.*

5. CONCLUSION

We presented some recent research done in the field of succinct data structures for text with the support for update operations. Gonzalez et al. ware the first to manage to get the operations in $O(\log n(1+\frac{\log\sigma}{\log\log n}))$ time with only $nH_0 + o(n)\log\sigma$ bits. Quickly after He improved their result with a slight modification of this algorithm, by replacing the red-black tree used by Gonzalez with a B-tree. They cut the wort-case time complexity by a $\log\log n$ factor. Gupta et al. took a totally different direction and managed to get a better query time complexity but sacrificed update times. Their solution is designed under the assumption that the string is queried frequently but updated infrequently.

The research of the succinct data structures has got a lot of momentum in the last few years [9] and it produced surprising results. The most successful indexes are able to obtain almost optimal space and search time.

6. REFERENCES

[1] P. Ferragina, G. Manzini, and V. Mäkinen. Compressed representations of sequences and full-text indexes. *ACM Transactions on ...*, Jan 2007.

[2] A. Golynski, J. Munro, and S. Rao. Rank/select operations on large alphabets: a tool for text indexing. *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 368–373, 2006.

[3] R. González and G. Navarro. Rank/select on dynamic compressed sequences and applications. *Theoretical Computer Science*, Jan 2009.

[4] R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. ... *of the fourteenth annual ACM-SIAM ...*, Jan 2003.

[5] A. Gupta, W. Hon, R. Shah, and J. Vitter. A framework for dynamizing succinct data structuresâŃŹ(2008). *works.bepress.com*.

[6] M. He and J. I. Munro. Succinct representations of dynamic strings. *arXiv*, cs.DS, May 2010.

[7] W. Hon, K. Sadakane, and W. Sung. Succinct data structures for searchable partial sums. *Algorithms and Computation*, pages 505–516, 2003.

[8] G. Jacobson. Space-efficient static trees and graphs. *Foundations of Computer Science, 1989., 30th Annual Symposium on*, pages 549–554, 2002.

[9] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys (CSUR)*, Jan 2007.

[10] R. Raman, V. Raman, and S. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)*, 3(4):43, 2007.

Propagation Algorithms for Protein Classification

András Bóta
Institute of Informatics
University of Szeged, Hungary
bandras@inf.u-szeged.hu

ABSTRACT

In this paper we propose a simple propagation algorithm for protein classification. It is similar to existing label propagation algorithms, with some important differences. The method is general so it can be used for other purposes as well, although in this paper it is used solely for solving a binary classification problem, and its results are evaluated using ROC analysis. It performs in reasonable time, and produces fairly good results. We have used various databases for testing purposes including SCOP40 and 3PGK.

General Terms

Classification

Keywords

label propagation, protein classification, binary classification, ROC analysis

Supervisor

Miklós Krész[1]

1. INTRODUCTION

The categorization of genes and proteins is an important field of research, and a variety of methods and algorithms has been tried to solve this problem, starting from simple pairwise comparison to the more advanced methods of artificial intelligence, like support vector machines or hidden Markov models. A database of protein sequences can be effectively represented as a network, in which edges represent some sort of similarity. This similarity network is known a priori and forms the basis of the classification process. On the other hand, similarity networks are usually large, so only fast algorithms are able to tackle with the problem. Label propagation algorithms originate from graph theory, and have the advantage of using the entire network, represented as a similarity matrix, for the process of classification. Propagation algorithms are in average faster then their more advanced opponents, and can produce reliable results. There are various propagation methods currently in the field of bioinformatics [1, 2, 5, 10]. Our method is in many ways similar to these methods, but far more simpler than them, still producing good results.

[1]Hungary, University of Szeged, Department of Applied Informatics, contact: kresz@jgypk.u-szeged.hu

2. PROTEIN CLASSIFICATION METHODS AND EVALUATION

In this chapter we will give a short introduction to protein classification, as well as ROC analysis, which is a widely accepted measurement method in bioinformatics. We will also discuss a few propagation methods.

2.1 Protein classification

There are various similarity measures that can be applied to proteins. Databases like COG are based on functional similarities, while other databases (like SCOP40) are based on structural similarity. More precisely the similarity is based on the three dimensional structure and amino acid sequences of the proteins [10]. Two protein sequences are similar if they contain subsequences that share more similar amino acids than would be expected to occur by chance. Sequence comparison algorithms can be based on exhaustive search, like the Smith-Waterman algorithm, but heuristic algorithms, like BLAST, are also used frequently. Other forms of similarity measurements include structural comparison methods (like PRIDE or DALI). For details about the above methods see [3]. The measurement counted by these methods is a true similarity measurement, meaning that for very similar proteins, they produce a high score, and for very different proteins they produce a value close to zero. It might be worth mentioning, that some algorithms produce a distance measurement, which means, that similar elements have a score close to zero, and different elements have a greater score.

Protein classification is traditionally considered to be a binary classification problem [2], although the one-class classification approach is also very popular [1]. In this paper, we stick to the traditional approach, meaning we consider the training dataset as a set of both positive and negative examples, and we expect the algorithm to produce the same classification.

The SCOP40mini database consist of 1357 protein domain sequences, which belong to 55 superfamilies [3]. The sequences were divided into training and test datasets. Members of the test datasets were chosen so that they were not represented in the training set, meaning there was a low degree of sequence similarity and no guaranteed evolutionary relationship between the two sets. The SCOP40 mini database contains nine different similarity measurements, based on sequence and structural comparison methods [3].

2.2 ROC Analysis

Measuring the goodness of the results of an algorithm is also an important issue. One of the most widely accepted measurement methods in bioinformatics is receiver operating characteristics (ROC) analysis. ROC analysis provides both visual and numerical summary of the classifiers results [9]. Originally developed for radar applications in World War II, it became widely used first in medical applications, later in machine learning and data mining.

Receiver operating analysis is generally used for evaluating the results of a binary classifier. A binary classifier assigns an element to one of two classes, usually noted positive or negative (+ or -). The classification process uses two distinct datasets, the train and the test dataset. In machine learning, the training dataset is used to train the algorithm, in other methods it is used as the basis of calculating the results. Classification methods can be divided further into two distinct categories. Discrete classifiers only predict the classes where the test elements belong. This means, that the classifier can produce 4 different results: true positive, true negative, false positive and false negative. From these, a contingency table can be constructed, and various measurements can be counted. Of these, two important ones should be mentioned:

- TPR (true positive rate) $= \dfrac{TP}{TP + FN}$
- FPR (false positive rate) $= \dfrac{FP}{FP + TN}$

Where TP, TN, FP, FN denote the four cases mentioned above. The other type of classifiers is the probabilistic classifier. This assigns a value between 0 and 1 to each member of the test set, which can be viewed as a degree of similarity to the distinct classes. This score is then used to create a ranking of elements, and the classifier is good, if the positive elements are at the top of the list. More precisely if we apply a threshold value to this ranking we can create a discrete classifier. By applying multiple threshold values, we can create an infinite number of classifiers.

Using these threshold values we can create a ROC curve by counting the true and false positive rates for each threshold value. An example can be seen on Figure 1. Each point of the curve corresponds to a threshold value, which in turn corresponds to a discrete classifier. A ROC curve can be evaluated both numerically and graphically. The ideal classifier creates a ROC curve that is identical to the unit step function. A classifier which assigns test elements to the classes at random generates a ROC curve that is identical

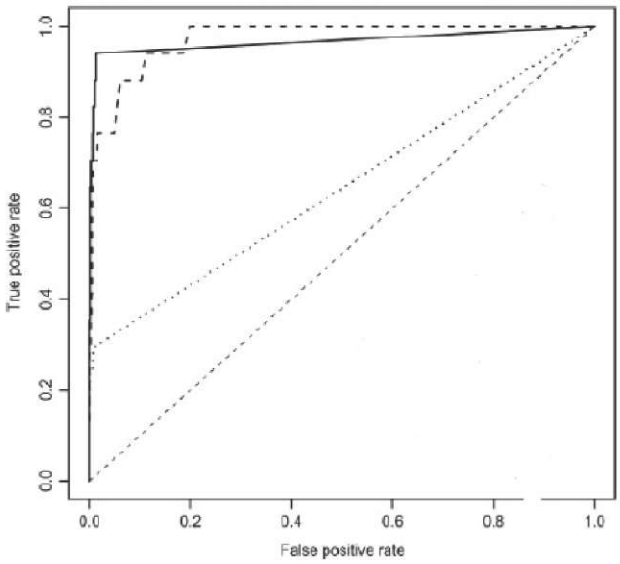


Figure 1: ROC curve

to the unit ramp function. In general, the higher the curve, the better the classification. It must be mentioned, that the curve is only defined between 0 and 1 on each axis. This is of course derived from the fact that both TPR and FPR are values between 0 and 1.

By counting the integral value of the curve we get the "area under curve value", or AUC. The AUC value of an ideal classifier is 1, the random classifier has a value of 0.5. The AUC value can be interpreted as the probability that a randomly chosen positive element is ranked higher than a randomly chosen negative element. In this paper the AUC value in itself is used to measure the goodness of the results, although it is worth mentioning that a high AUC value does not guarantee that the positive elements are ranked at the top of the list. This topic is discussed further in [8, 9].

2.3 Propagation algorithms

Originating from graph theory, propagation algorithms require a network containing vertices and edges as the basis of the method. Perhaps the most known propagation method is the PageRank algorithm [6], first used by Google for information retrieval tasks. Note, that Kleinberg had proposed a very similar algorithm for finding hubs and authorities earlier [4].

There are various ranking algorithms specifically for protein classification, namely [1, 2, 5, 10] . In general, these methods use a query mechanism to generate results. With the given similarity network, we add a new element as query, and the classification algorithm calculates the class (or the probability) this element belongs to.

In comparison, most algorithms from graph theory [7] use the entire network for propagation. In label propagation, some nodes of the network have initial labels, some not, and the purpose of the algorithms is to assign labels to nodes,

that did not have labels previously. The method described in this paper uses the latter approach, and volunteers to solve the classification task without the need to cut the edges of the similarity network.

3. A SIMPLE PROPAGATION ALGORITHM

The method we propose in this paper is a simple general propagation algorithm for function approximation closely based on the works of [7]. The method is general in a sense, that it was not created solely for the purpose of protein classification, although in this paper we will stick to this example of its application.

3.1 The label propagation algorithm by Raghavan et al.[7]

Originally a community detection algorithm, the label propagation method of Raghavan et al. is based on a simple, but effective idea. In a graph (directed or not) we assign labels to each node. In one iteration step, each node x adopts the label that a maximum number of its neighbors $x_1, x_2, ..., x_k$ share. When a tie happens, a random selection is used. At the beginning, each node is initialized with a different label, and as labels propagate through the network, sooner or later a consensus emerges among the densely connected subgraph of the original graph. Communities are formed this way. It is also easy to generalize this method to weighted graphs. An example of this process can be seen on Zachary's karate club network [11] on Figure 2. The three different shades of grey represent the different labels.

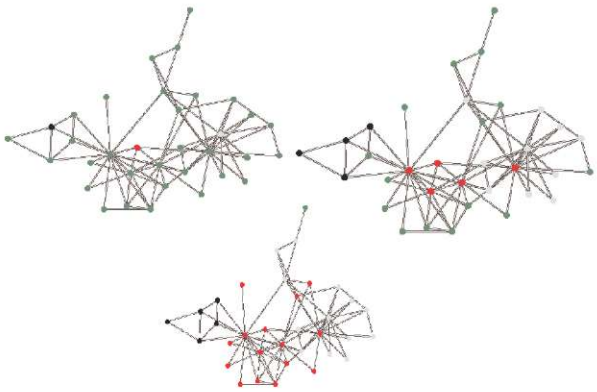


Figure 2: Propagation example

The labels can be updated synchronously or asynchronously. In the first case the labels assigned at iteration number t depend only on the labels of iteration $t-1$: $L_x(t) = f(L_{x_1}(t-1), L_{x_2}(t-1), ..., L_{x_k}(t-1))$, where $L(x)$ denotes the label of node x at iteration t, and f is the update function.

If there are subgraphs in the network, that are bipartite or nearly bipartite in structure this method might oscillate. The asynchronous update method solves this problem: we consider a node in iteration t. There are some neighbors that have already updated their labels in iteration t : $x_1, x_2, ..., x_m$, and some have not, these have the labels from iteration $t-1$: $x_{m+1}, x_{m+2}, ..., x_k$. We update the label of the node based on the labels of the current iteration in the case of neighbors, that have already updated their labels. For neighbors not yet updated, we use the labels of iteration $t-1$. $L_x(t) = f(L_{x_1}(t), L_{x_2}(t), ..., L_{x_m}(t), L_{x_{m+1}}(t-1), L_{x_{m+2}}(t-1), L_{x_k}(t-1))$. In the synchronous case the order in which we choose the nodes for updating is irrelevant. In the asynchronous case the order can be important. In the original method a random approach is used.

This algorithm can also be called the coloring algorithm, because the labels can be viewed as colors, and in the end, each community has a different color. The colors or labels can also be viewed as categories, thus if we have only two labels, we have two categories, and this correspons to a binary classification problem.

3.2 Application for binary classification

As we have mentioned before, in binary classification the two classes are usually called positive and negative, and there are two distinct datasets used by the algorithm, the train dataset, and the test dataset.

A similarity matrix is constructed (according to some comparison method) between the elements of both datasets. The similarity values are available for the whole dataset, even between the test and train datasets $s(x, y) : x, y \in (Train \cup Test)$. If we consider the dataset elements as nodes of a graph, this similarity value defines an edge weight, creating a weighted graph containing both the elements of the train and test datasets.

With a small modification, the algorithm described above can be applied to the protein classification problem. In the original algorithm each node receives an initial label different from all other labels. This is of course not correct for the purpose of binary classification. The labels are only known a priori for the train dataset, so only these nodes get an initial label, and as the algorithm terminates, the unlabeled nodes will get labels as well.

If we apply this algorithm directly to the problem described above, we get very poor results. This is not surprising, since this method (and most community detection methods) are not designed for complete graphs like the similarity networks of this problem.

One obvious way to solve this problem is to convert the graph into an unweighted form by cutting the edges according to some weight limit, but this results to the also obvious problem of choosing the appropriate value. With luck or after a significant number of trial and error steps, it is possible to get good results. Still, this is not a feasible solution to the problem.

3.3 The AProp method

The method proposed in this paper uses a different approach. Rather than using discrete labels for the algorithm, we work with continuous ones. Theoretically there are no restrictions on the labels we can use, although in our approach, we have used real numbers to represent the labels. If we choose two distinct labels for the original classes (like

0 and 1), the algorithm will assign values to the uninitialized nodes between these. It is important to point out, that these values should not be thought of as probabilities. They represent some "closeness" to one class or the other. These values can then be evaluated with ROC Analysis to produce a goodness measurement.

The update rules are only shown for the synchronous method. There are two reasons for this. In the original algorithm, the asynchronous update rule was introduced to solve convergence issues. These issues occurred because nearly bipartite subgraphs caused fluctuations of the discrete labels. Using continuous labels solves this problem, thus the asynchronous rules are not needed. In contrast to this, just for the sake of curiosity we have implemented the asynchronous version as well, but testing showed that they do not perform as good as the synchronous method.

The labels are updated according to the formula described in the previous section, the only difference being the update function f changing to a simple summation. The label of node x changes based on the labels of its neighbors $N(x)$:

$$L_x(t) = \frac{1}{d} \sum_{y \in N(x)} L_y(t-1),$$

where d denotes the degree of x. It is easy to extend this rule for weighted graphs. Let d_w denote the weighted degree of x.

$$L_x(t) = \frac{1}{d_w} \sum_{y \in N(x)} s(x,y) L_y(t-1),$$

where $s(x,y)$ is the similarity value between nodes x and y mentioned above. It is worth noting, that the similarity value is symmetric $s(x,y) = s(y,x)$, so we have designed the algorithm to work with undirected graphs. The initialized labels never change, so they should be left out of the update process. This also reduces time complexity significantly.

The algorithm consists of iteration steps, the number of iterations being a possible parameter. Another way to stop the algorithm would be to count the changes that happen in an iteration, and when this change is below a given minimum, the algorithm halts. Our experience shows us, that no significant change occurs after 6 to 8 iterations.

The label propagation algorithm of Raghavan et al. begins with initializing each node with a different label. We have dropped this approach because we wanted to use the algorithm for binary classification, but since we do not know the labels of the members of the test database a priori, some nodes will be left uninitialized. This leads us to the problem of representing the uninitialized status of nodes.

There are two ways of solving the problem. The first way is the "blind" approach. In this case, we do not represent the uninitialized status at all. These nodes can be selected for updating labels, but otherwise they are simply left out of the summation. To do this, first we have to sort out the unlabeled nodes from the neighborhood sets. Also, when we count the weighted degree, we should only consider edges that are incident to labeled nodes. This somewhat changes the update rule.

$$L_x(t) = \frac{1}{d_s} \sum_{y \in N_s(x)} s(x,y) L_y(t-1),$$

where $N_s(x)$ denotes the sorted neighborhood, and d_s denotes the sorted degree.

The second method is to assign a value to the uninitialized status. This way, we do not have to change the algorithm at all. But this also leaves us with the question: What value should be used for the uninitialized nodes? This solution is called the "biased" approach. The value of the uninitialized nodes should be chosen accordingly to the two class labels. Depending on these considerations, the algorithm can be further divided into three categories:

- Neutral biased. The value is neutral towards the class labels. For example, the class labels are 0 and 1, the value is 0.5.
- Directly biased. The value is one of the class labels. For example, the class labels are 0 and 1, the value is 0 or 1.
- Indirectly biased. Any value, that is not one of the class labels or the neutral value. In the case of 0 and 1, any value other than $0, 1$ and 0.5.

In the last two cases, further distinction can be made depending on what class label do we choose, or in the third case what class label is closer to the uninitialized value.

In the end, there are four (or six) variations of the algorithm. Perhaps it is not a big surprise that these variations perform slightly differently when applied to the protein classification problem.

4. RESULTS

Our method was tested on the SCOP40mini and 3PGK databases [3]. The SCOP40mini database consists of 55 classification tasks, and there are seven similarity (or distance) networks available for it. The 3PGK database consists of ten experiments and four similarity (or distance) networks. Further details of the networks can be seen on [3]. In Tables 2 to 5,

- BLAST stands for the basic local alignment search tool.
- SW stands for the Smith-Waterman algorithm.
- NW stands for the Needleman-Wunsch method.
- LA stands for the local alignment kernel method.

Table 1: The convergence of the algorithm

Iterations	1	2	3	4	5	6	7	8	10	16
Ex1	1	1	1	1	1	1	1	1	1	1
Ex2	1	1	1	1	1	1	1	1	1	1
Ex3	0.9548	1	1	1	1	1	1	1	1	1
Ex4	0.7391	0.8009	0.8571	0.893	0.9207	0.9253	0.9317	0.9336	0.9345	0.9354
Ex5	0.8827	0.9215	0.9279	0.9236	0.9204	0.9118	0.9107	0.9107	0.9096	0.9096
Ex6	1	1	1	1	1	1	1	1	1	1
Ex7	0.709	0.7909	0.8227	0.8363	0.85	0.85	0.85	0.85	0.85	0.85
Ex8	0.9727	0.9886	0.9931	0.9931	0.9931	0.9931	0.9931	0.9931	0.9931	0.9931
Ex9	1	1	1	1	1	1	1	1	1	1
Ex10	0.7613	0.7982	0.8153	0.8295	0.8352	0.838	0.838	0.838	0.838	0.838
Avg	0.9019	0.93	0.9416	0.9475	0.9519	0.9518	0.9523	0.9525	0.9525	0.9526

- PRIDE stands for Pride structural similarity.
- LZW stands for the Lempel-Ziv-Welch method.
- DALI stands for DaliLite structural alignment.
- PsiBLAST stands for Position-Specific Iterated BLAST.

We have also compared our results with other algorithms provided as a benchmark. Further details of these can be seen on [3]. In Tables 2 to 5,

- 1nn stands for the one nearest neighborhood method.
- RF stands for random forest classification.
- SVM stands for a support vector machine.
- ANN stands for an adaptive neural network.
- LogReg stands for logistic regression.
- AProp stands for approximate propagation.

4.1 Iteration number

For determining the optimal iteration number, we have tested the algorithm on the Smith-Waterman similarity matrix of the 3PGK database. In Table 1, the AUC values are displayed for the individual experiments and the average value as well.

It can be seen that the values do not change much after iteration 7, so this was the value we have used for the rest of the evaluation. It is also worth noting that the performance does not increase monotonously before iteration 7, but fluctuate somewhat.

4.2 3PGK

The results for all variations of the algorithm can be seen in Tables 2 and 3, as well as a comparison against other algorithms. In the first column, the algorithm variations can be seen.

- nb denotes neutral biased.
- dbn denotes directly biased towards negative.

Table 2: Results for 3PGK

Method	BLAST	SW	LA
blind	0.9241	0.9470	0.9484
nb	0.9234	0.9412	0.9481
dbn	**0.9336**	**0.9518**	0.9473
dbp	0.8973	0.9029	0.9483
idbn	0.9267	0.9317	0.9444
idbp	0.8898	0.9048	**0.9485**

Table 3: Results for 3PGK

Method	BLAST	SW	LA
1nn	0.8633	0.8605	0.8596
RF	0.8517	0.8659	0.8755
SVM	0.9533	0.9527	0.9549
ANN	**0.9584**	**0.9548**	0.9564
LogReg	0.9537	0.9476	**0.9593**
AProp	0.9336	0.9518	0.9485

- dbp stands for directly biased towards positive.
- idbn stands for indirectly biased towards negative.
- idbp denotes indirectly biased towards positive.

If we consider only the variations of the approximation algorithm, we can see that none of the approaches appear to be dominant. It can also be seen, that our method performs almost as good as the best approaches.

4.3 SCOP40mini

The results for all variations of the algorithm can be seen in the Tables 4 and 5, as well as a comparison against other algorithms. Unlike in the previous case, the idbp approach outperforms all other variations. We can only guess the reason for this behaviour. It is possible, that the experiments in this dataset are structured in a way, that initializing the test dataset towards the positive class label greatly enhances performance. In the implementation, the positve class was labeled 1 and the test dataset was intialized with the value of 2.

Table 4: Results for SCOP40mini

Method	BLAST	SW	NW	LA	PRIDE	LZW	PsiBLAST	DALI
blind	0.899	0.9476	0.9337	0.9442	0.8891	0.8261	0.9083	0.9974
nb	0.9245	0.9492	0.9353	0.9455	0.8897	0.8267	0.9239	0.995
dbn	0.8909	0.9475	0.9335	0.944	0.889	0.826	0.9065	0.9974
dbp	0.932	0.9509	0.9372	0.9466	0.8903	0.8273	0.9304	9977
idbn	0.66	0.9434	0.929	0.9412	0.8876	0.8247	0.7792	0.994
idbp	**0.9382**	**0.9536**	**0.9405**	**0.9491**	**0.8914**	**0.8283**	**0.9363**	**0.998**

Table 5: Results for SCOP40mini

Method	BLAST	SW	NW	LA	PRIDE	LZW	DALI
1nn	0.7577	0.8154	0.8252	0.7343	0.8644	0.7174	0.9892
RF	0.6965	0.8230	0.8030	0.8344	0.9105	0.7396	0.9941
SVM	0.904	0.9419	0.9376	0.9396	**0.9361**	0.8288	0.9946
ANN	0.7988	0.8875	0.8834	0.9022	-	**0.8346**	-
LogReg	0.8715	0.9063	0.9175	0.8766	0.9029	0.7487	0.9636
AProp	**0.9382**	**0.9536**	**0.9405**	**0.9491**	0.8914	0.8283	**0.998**

It can be seen that our method performs better than the other algorithms in most cases. The running time of the algorithm is very low. The database consists of 55 classification tasks. Solving all of these tasks took 15 minutes on an average notebook computer. This means that for the evaluation of one task, around 16 seconds is needed.

5. CONCLUSION AND FUTURE WORKS

It can be seen from the results, that our algorithm generally performs better on larger databases, with the running time being sufficiently low. The method is relatively general in a way, that it can be used as an approximation algorithm, but it can be extended in several ways described below.

This method is very sensitive to the starting labels of the nodes. This can be a weakness, but a proper learning algorithm can exploit this property and improve the overall performance of the algorithm.

Making the method more general can be difficult. The labeling approach is designed to handle binary classification tasks. One way to extend this would be to use multi-dimensional labels. Using one number to describe a label corresponds to a one dimensional coordinate. Extending the number of dimensions for the labels would result in multiple classes, while one coordinates would indicate some sort of a membership function towards one class or the other.

Acknowledgment

The author was supported by the Project named "TÁMOP-4.2.1/B-09/1/KONV-2010-0005 - Creating the Center of Excellence at the University of Szeged" also supported by the European Union and co-financed by the European Regional Fund.

6. REFERENCES

[1] A. Bánhalmi, R. Busa-Fekete, and B. Kégl. A one-class classification approach for protein sequences and structures. In *Proceedings of the 5th International Symposium on Bioinformatics Research and Applications*, ISBRA '09, pages 310–322, Berlin, Heidelberg, 2009. Springer-Verlag.

[2] R. Busa-Fekete, A. Kocsor, and S. Pongor. Tree-based algorithms for protein classification. In *Computational Intelligence in Bioinformatics*, pages 165–182. 2008.

[3] http://net.icgeb.org/benchmark/. Protein benchmark database.

[4] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46:604–632, September 1999.

[5] A. Kocsor, R. Busa-Fekete, and S. Pongor. Protein classification based on propagation of unrooted binary trees. *Protein and Peptide Letters*, 15(5):428–437, June 2008.

[6] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[7] U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E*, 76(3):036106, Sep 2007.

[8] A. K. Róbert Busa-Feketea, Attila Kertész-Farkasa and S. Pongor. Balanced roc analysis (baroc) protocol for the evaluation of protein similarities. *Journal of Biochemical and Biophysical Methods*, 70(6):1210 – 1214, 2008.

[9] P. Sonego, A. Kocsor, and S. Pongor. Roc analysis: applications to the classification of biological sequences and 3d structures. *Briefings in Bioinformatics*, 9(3):198–209, January 2008.

[10] J. Weston, A. Elisseeff, D. Zhou, C. Leslie, and W. Noble. Protein ranking: from local to global structure in the protein similarity network. *PNAS USA*, 101:6559–6563, 2004.

[11] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

Optimal and Reliable Covering of Planar Objects with Circles

Endre Palatinus
Institute of Informatics
University of Szeged
2 Árpád tér
Szeged, Hungary H-6720
palatinuse@gmail.com

ABSTRACT

We developed an algorithm for finding the sparsest covering of planar objects with a given number of circles with fixed centers. The covering is tested reliably using interval arithmetic. We applied our solution to a telecommunication related problem.

Supervisor: Dr. Balázs Bánhelyi assistant professor, Institute of Informatics, University of Szeged

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Global optimization*

1. INTRODUCTION

The circle packing problem has attracted much attention in the last century, and a variant called packings of equal circles in a square receives attention even nowadays [1]. The objective of it is to give the densest packing of a given number of congruent circles with disjoint interiors in a unit square.

However, its dual problem, the circle covering has not been exhaustively studied so far. Optimal coverings of the unit square with congruent circles of minimal radii have been found[2] only for small numbers and their optimality have been proved mathematically. Computational methods have been developed[3] to find solutions for higher circle counts, but the produced results were neither computationally reliable, nor provable approximations of the global optimum.

Our aim is to develop a global optimization method for finding the sparsest reliable covering of a convex or concave planar object with a given number of circles with fixed centers where the sizes of the circles' radii may be different. By sparsest we mean minimizing the sum of the circles' area. We have chosen this fitness function, because it can prove useful in many applications, such as covering a region with terrestrial radio signals and determining the optimal energy consumption of wireless sensor networks.

2. RELIABLE COMPUTING

The CPUs of todays personal computers have an arithmetic optimized for computer games. This means that the speed of the computations is more important than the accuracy of them. The floating point computations performed by these CPUs yield results in the domain of the numbers which can be represented in their arithmetic. Since every result is only an approximation of the true results, the error of a complex computation can be arbitrarily large. Consider for example the following expression:

$$2^{100} + 2010 - 2^{100} = ?$$

When calculating the result of this expression using double-precision floating point arithmetic the result will be 0 instead of 2010. This is due to the limited precision of the arithmetic.

Mathematicians don't accept computer assisted proofs if they contain computations performed with unreliable arithmetic. Even the solution of global optimizers can be doubted to be globally optimal ones if we consider that their computations contain accumulated rounding errors. Interval arithmetic is one solution for handling the uncertainty of computations. It represents the results of computations as intervals, which are guaranteed to contain the precise result. This way it incorporates the rounding errors, too, but does not hide it as the floating point arithmetic does.

3. THE CIRCLE COVERING PROBLEM

Let $(r_1, r_2, \ldots, r_n)$ denote the radii of the circles in our constrained optimization problem. Therefore the fitness function is:

$$f\left((r_1, r_2, \ldots, r_n)\right) = \sum_{i=1}^{n} r_i^2 \rightarrow min$$

To solve the previous problem we need:

- a reliable method for testing if a given arrangement is covering or not
- and an optimizer for finding the optimal one.

Checking if a given arrangement is covering or not can be done by checking whether every point of the planar object

is covered by any of the circles. This is where interval arithmetic proves to be quite useful, because applying it we can check whether a rectangle is covered by any of the circles simply by replacing points with intervals. An additional benefit of using it is that our algorithms become reliable by avoiding uncertain computations. In our implementation we used C-XSC[4], a widely recognized C++ class library for extended scientific computing, which provided the interval arithmetic extension of all the mathematical operations.

We can define the covering property as follows: Every circle is an open set on a plane. The desired property for every p point of the planar object and some $(center, r)$ circle:

$$sup((p - center)^2) < inf(r^2)$$

It is straightforward to use a Branch-and-bound–based method for checking the covering. We have chosen a parallelized B&B method introduced in [5], since there is a great potential in traversing the different branches of the B&B search tree concurrently on different CPU cores.

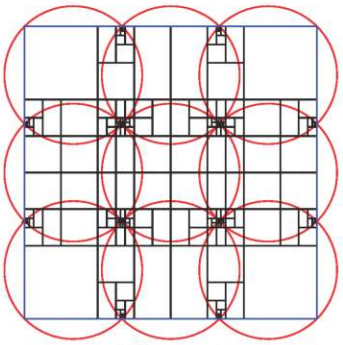
We can even check general properties of multidimensional intervals, as the following statements say: If we have a method capable of proving a P property of a point and its neighborhood, then the B&B–based testing method is able to prove this P property for a multidimensional interval. In case the interval has the given property we have to traverse the whole B&B search tree, and the subtrees can be traversed concurrently by different CPU cores. If there is no lower bound on the width of the subintervals created, then in the positive case the algorithm terminates in finite steps with positive answer[6]. In practice there's always a lower bound (ϵ), and if the width of a subinterval reaches this bound and it does not have the property, then we cannot decide the problem and return a negative answer.

To demonstrate the running of the testing method, we will use some simple testcases: n^2 circles on a regular $n * n$ grid. The results can be seen on figure 1. As our experiments have shown, we have gained an increase in performance comparable to the number of CPU-cores[7] on these simple test cases, as shown on figure 2. We can also state according to figure 3 - which is the plot of the runtime divided by n^4 versus n - that the runtime of the algorithm is $O(n^4)$, that is $O(m^2)$ if m denotes the total number of circles, which is equal to n^2.

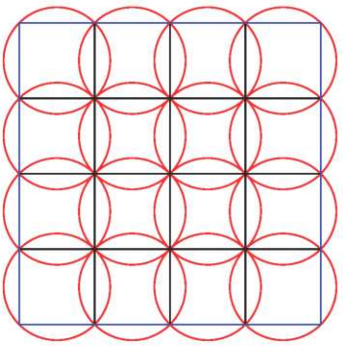
4. THE MATHEMATICAL MODEL OF THE PROBLEM

Now that we have introduced a reliable method for testing if a given arrangement is covering or not, we can concentrate on our second problem: finding the optimal one. First we have to examine the mathematical model of the problem.

We start with some simple definitions: We call $r = (r_1, r_2, \ldots, r_n)$ a *configuration*, which is a vector holding the radii of the circles. A configuration is bad if the corresponding circles do not cover the planar object, and good, otherwise. Let $C = ([\underline{c_1}, \overline{c_1}], \ldots, [\underline{c_n}, \overline{c_n}])$ denote a *set of configurations*, where the i-th component of every configuration in the set falls into the $[\underline{c_i}, \overline{c_i}]$ interval. The two *distinguished configu-*



(a) $n = 3$



(b) $n = 4$

Figure 1: The running of the testing method.

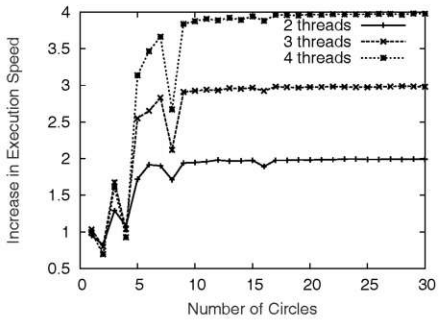


Figure 2: The efficiency of the testing method.

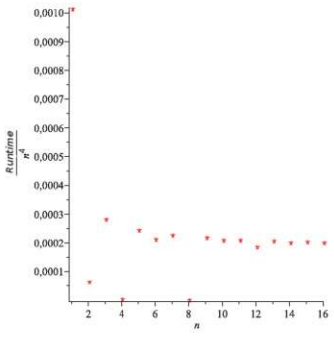
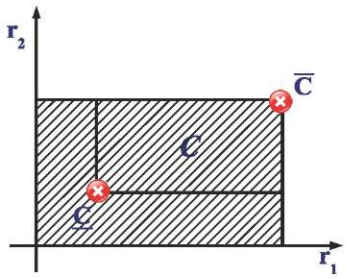
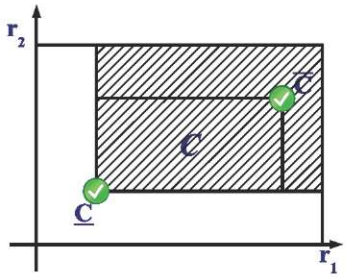


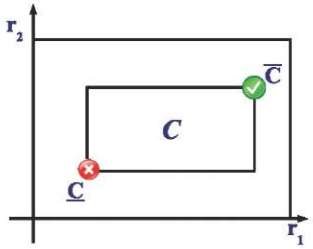
Figure 3: The runtimes of the testing method.



(a) Both bad



(b) Both good



(c) $\underline{C}$ bad and $\overline{C}$ good

Figure 4: Properties of the configuration sets.

rations of C are: $\underline{C} = (\underline{c_1}, \ldots, \underline{c_n})$ and $\overline{C} = (\overline{c_1}, \ldots, \overline{c_n})$.

Now we will show some simple properties of these configuration sets, which will prove useful in constructing an optimizer for the radii sizes. If the distinguished configurations, $\underline{C}$ and $\overline{C}$ of C are *bad*, then the $C' = ([0, \overline{c_1}], \ldots, [0, \overline{c_n}])$ set cannot contain the optimal solution. If the distinguished configurations, $\underline{C}$ and $\overline{C}$ of C are *good*, then the $C \setminus \underline{C}$ set cannot contain the optimal solution. It follows that only such a C set *may contain* the optimal solution, whose $\underline{C}$ configuration is *bad*, and $\overline{C}$ configuration is *good*. On figure 4 we have depicted these statements in two dimensions.

5. DETERMINING THE OPTIMAL RADII SIZES

Finally, we can introduce our B&B–based global optimizer for the radii sizes.

Optimizer algorithm:

- Let Q be an empty minimum priority queue, whose C_i elements' priority is determined by the value of the fitness function for the $\underline{C_i}$ configuration.
- Push the $C_0 = ([0, c_1], \ldots, [0, c_n])$ initial set of configurations into Q, where $\overline{C_0}$ is a configuration with a suitably high fitness value.
- **Loop:**
  - Pop a C set of configurations from Q.
  - Subdivide C along its widest side into two equal parts, C_1 and C_2, in such a way that $\underline{C} = \underline{C_1}$ and $\overline{C} = \overline{C_2}$.
  - If $\overline{C_1}$ is a good configuration, than push C_1 into Q.
  - If $\underline{C_2}$ is a bad configuration, than push C_2 into Q.
- **Until** the width of C is less than ϵ.
- Return $\overline{C}$ as the solution.

Using the properties of the configuration sets it can be proved that the above algorithm finds an approximation of the global optimum with arbitrary precision.

6. APPLICATIONS

We can apply our results for the problems mentioned in the introduction. Covering Hungary with terrestrial radio signals is a problem where we have to cover a concave planar object with circles, since the signals cover a circular area around the broadcasting stations. The area covered by a station is linearly dependent on the power fed into it, therefore minimizing the power consumption of the hole network is equivalent to minimizing the sum of the area of each circular region covered by the stations, which happens to be the value of our fitness function. Some optimal solutions can be seen on figure 5. Similar problems have been solved in [8] but with unreliable computations.

Some optimal coverings of the unit square are also shown on figure 6.

7. REFERENCES

[1] Szabó, P.G., M.Cs. Markót, T. Csendes, E. Specht, L.G. Casado, and I. García: New Approaches to Circle Packing in a Square – With Program Codes. *Springer, Berlin,* 2007.

[2] Erich Friedman: Circles Covering Squares. *http://www2.stetson.edu/ efriedma/circovsqu/,* (2005)

[3] Nurmela, K.J. and P.R.J. Östergard: Covering a square with up to 30 equal circles, *Helsinki University of Technology, Laboratory for Theoretical Computer Science Research Reports,* **62**(2000)

[4] Hofschuster, Krämer, Wedner, Wiethoff, *C-XSC 2.0 - A C++ Class Library for Extended Scientific Computing*, Preprint BUGHW-WRSWT 2001/1, Universität Wuppertal, 2001.

[5] Casado, L.G., J. A. Martinez, I. Garcia, and E.M.T. Hendrix :, Branch-and-Bound interval global optimization on shared memory multiprocessors, *Optimization Methods Software,* **23**(2008), 689–701.

[6] Bánhelyi, B., Csendes, T. and Garay B. M.: A Verified Optimization Technique to Locate Chaotic Regions of Hénon Systems. *Journal of Global Optimization,* **35**(2006), 145–160.

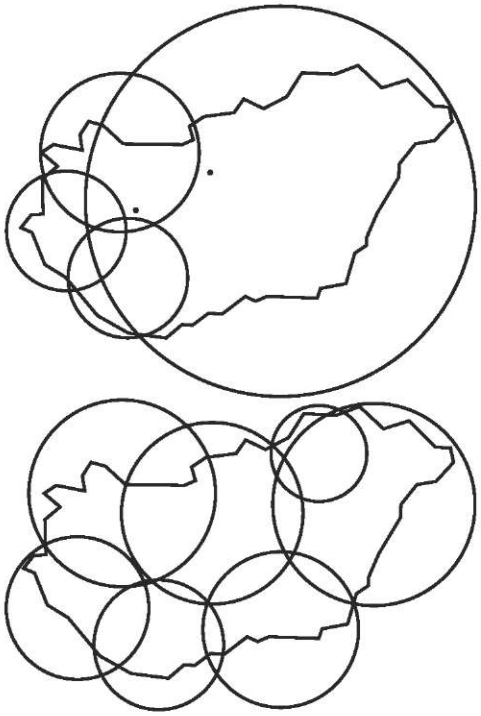


Figure 5: Optimal coverings of Hungary

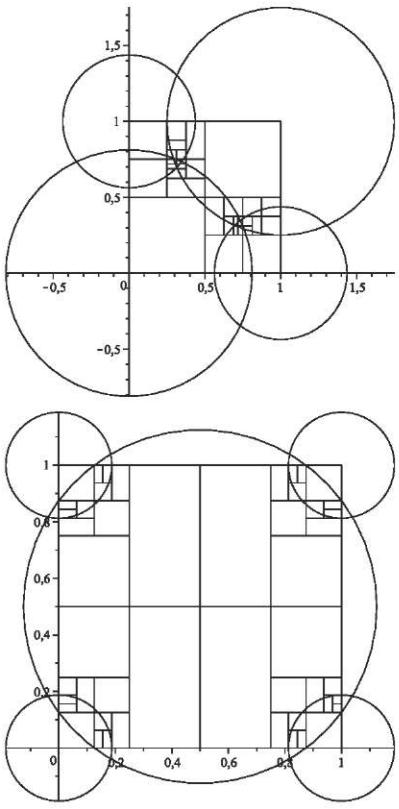


Figure 6: Optimal coverings of the unit square

[7] Palatinus, E., Bánhelyi, B.: Circle Covering and its Applications for Telecommunication Networks *Proceedings of the 8th International Conference on Applied Informatics*, Eger, Hungary, (2010), *Submitted*

[8] Das, G.K., S. Das, S.C. Nandy, and B.S. Shina: Efficient algorithm for placing a given number of base station to cover a convex region, *J. Parallel Distrib. Comput.* **66**(2006), 1353–1358.

TRUTH-TELLER—LIAR PUZZLES – A GENETIC ALGORITHM APPROACH (WITH TUNING AND STATISTICS)

Attila Tanyi
Debreceni Egyetem
Egyetem tér 1.
Debrecen, Hungary
+36 52 512900/22792
tatil@bom.hu

ABSTRACT

In this paper, satisfiability-like problems, namely a family of truth-teller—liar puzzles are considered. In our puzzles, every participant may say only one complex statement (a conjunction of atomic statements) about the type of the other participants. Truth-tellers can say only true atomic statements, while a liar must say at least one false atomic statement if he/she says anything. The problem is to find the solution, i.e., which participants are liars and which of them are truth-tellers. We present a solution algorithm based on a genetic algorithm approach, where several types of tuning can be implemented.

Categories and Subject Descriptors

I.2.m [**Artificial Intelligence**]: Miscellaneous – *genetic algorithms.*

General Terms

Algorithms, Measurement, Performance, Experimentation.

Keywords

Genetic Algorithm, Truth-Teller—Liar Puzzle, Tuning.

Supervisor

dr. Benedek Nagy.

1. INTRODUCTION

The truth-teller—liar puzzles are well-known. Scientists also like puzzles, since they can effectively be used in teaching, in argumentation etc. Puzzles are the topics of several books, they were analysed by, e.g., R. Smullyan [11,12,13]. Several variations of truth-teller-liar puzzles are characterized in [9]. There are Strong truth-tellers, whose all (atomic) statements are true, while a person is a Weak truth-teller if there is a true atomic statement among his statements. Similarly, a Strong liar can only say false (atomic) statements, while a Weak liar has at least one false atomic statement among the statements he is uttered. A puzzle has a category by the types of its truth-tellers and liars. In detail, SS-type puzzles are shown in [10]. We deal with special types of puzzles, called SW-type puzzles having Strong truth-tellers and Weak liars. They are solved by a graph modification method in [4,5]. They are also related to Boolean programming [5,7]. The duality of SW and WS puzzles is presented in [6]. Non-monotonic reasoning related to WW-puzzles is presented in [8]. In this paper, we consider only SW puzzles. The hardness of satisfiability-like problems suggest new approaches to try. Here, a genetic algorithm is used to solve these problems. Fine-tuning of its various parameters and functions are also shown to increase performance. The implementation is written in Java.

2. THE TRUTH-TELLER—LIAR PUZZLES

In this paper we use only special truth-teller puzzles, where the participants have full knowledge on their type and they can say only sentences about these facts.

There are N participants in the riddle, each of who is either a liar or a truth-teller. Statements are given in the form of

'X says: A_1 and A_2 and ... and A_j',

where A_1, A_2, ... A_j are sub-statements (or atomic statements) either in the form of

'Y is a liar' or 'Y is a truth-teller'.

Where X and Y are names of arbitrary participants. Each of the participants can only tell at most one statement about the other ones. The total number of atomic statements is M.

Example: The participants are Alice, Bob and Charlie. (N=3)
There are 2 statements given, which consist of 3 atomic statements. (M=3)
Alice says: *Bob is a liar and Charlie is a truth-teller.*
Charlie says: *Alice is a liar.*

The task is to find out which participants are liars, and which of them are truth-tellers by the given statements.

3. THE GENETIC APPROACH

In this section we give some details of our genetic algorithm. We do not recall all the details how a genetic algorithm works, they can be found in several text books in the topic, like [1,2,3]. For various terms used here we also refer to [1,2,3].
The chromosomes of the individuals are arrays of bits representing the statuses of the participants.

For example ($N = 7$):

	L	L	T	L	T	T	T
$c =$	(0	, 0	, 1	, 0	, 1	, 1	, 1)

The **fitness function** comes from biological motivation. It rates the solution candidates by assigning fitness values to them. The better a solution candidate is, the greater value it should have. Here, the fitness function shows how many statements are fulfilled by a permutation of statuses, $f: C \to \{0,1,\dots,M\}$.
A solution is optimal, if the fitness function is equal to M, $f(c) = M$. (i.e. all the statements are fulfilled)
The genetic algorithm works with PS individuals at the same time, basically performing a parallel search. In each step, it creates a new generation of individuals, **keeping the best individual** for the next generation, and creating the rest by **roulette wheel selection,** which is selecting parents randomly (the greater fitness value the individuals have, the bigger probability they become selected), and creating offsprings by using **uniform crossover** and **mutation**.
Mutation operator is $m: C \to C$.
The mutation operator negates the bits of chromosomes with a small probability p .

$$m\big((s_1, s_2, \dots, s_n)\big) = (s_1', s_2', \dots, s_n')$$

$$s_i' = \begin{cases} s_i, & probability\ 1-p \\ |1 - s_i|, & probability\ p \end{cases}$$

Uniform crossover operator is $u: C \times C \to C$.
The uniform crossover operator assigns a new chromosome to two chromosomes where the bits at each position gets either of the values of the bits at the corresponding positions of the two chromosomes with a probability of 50%.

$$u\big((s_1, s_2, \dots, s_n), (t_1, t_2, \dots, t_n)\big) = (s_1', s_2', \dots, s_n')$$

$$s_i' = \begin{cases} s_i, & probability\ 50\% \\ t_i, & probability\ 50\% \end{cases}$$

4. FINE-TUNING

The data were gathered by running the algorithm 10, 50, or 100 times, and taking average. A run is measured by the **number of individuals** it creates, to which we will refer to as **running time**.

The following parameters are tuned:

N – number of participants

PM – probability of mutation

PS – population size

First we generate the puzzle by the only parameter N, the number of participants. The program generates 5% of all possible atomic statements (for example, for 20 participants - about 19 statements) and make the complex statement from them according to the definition.

4.1 Test 1: How does running time change when decreasing PM?

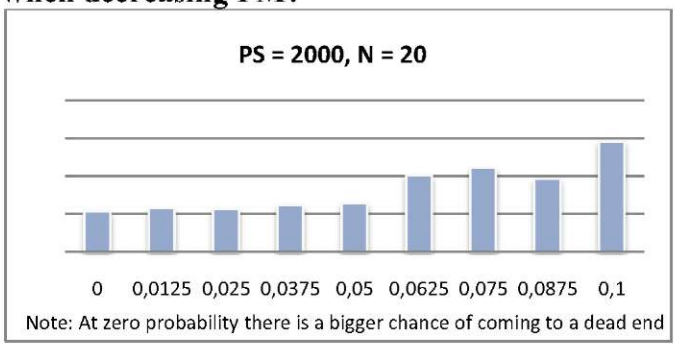


Figure 1. Running time as a function of probability of mutation.

To set the value of PM to be 0,05 might seem reasonable, as 0,05=1/N, therefore the algorithm would make one mutation in each individual. However, the graph shows the tendency: **the smaller the probability** of the mutation, the sooner we are getting the optimal result. This is because even just one mutation (changing the status of a participant) has dramatic effects on the fitness of the individuals. We should not go all the way to zero though: at zero, the probability of coming to a dead end increases, as usual in genetic algorithms.

4.2 Test 2: How does running time change when decreasing PS?

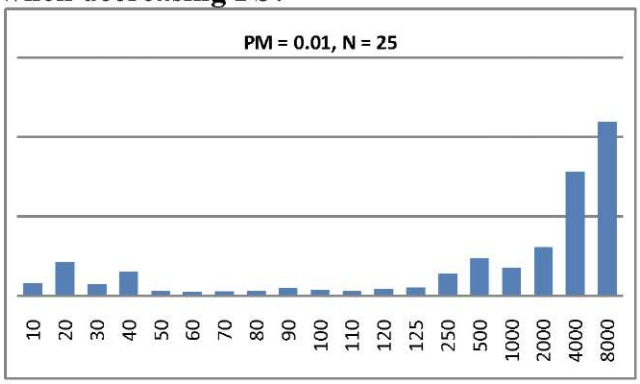


Figure 2. Running time as a function of population size.

Test 2 has another surprise, the population size of the fastest algorithms fall between 50 and 80. The **minimal value is at 60** for N=25 (while it was at 50 for N=20), which is quite far away from the initially used 2000. Even if the number of possible solution-vectors are very large, i.e., 2^N, it seems that relatively small population size suffices.

4.3 Test 3: How does running time change when increasing N?

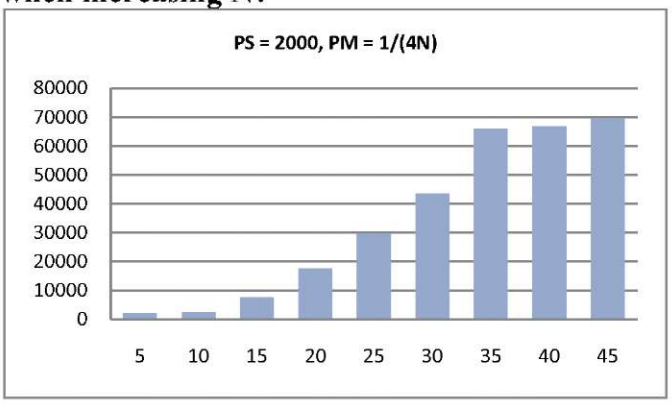
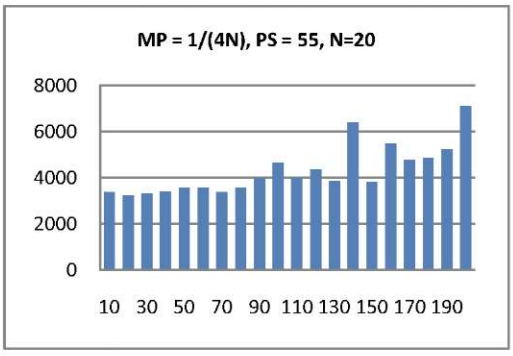


Figure 3. Running time as a function of N.

The number of individuals needed to find an optimal solution is growing fast before N=35. Above 35 it does not increase so fast by the graph, it is probably because our measure. Not to wait too much time a limit was given, i.e., if 150 generations does not provide the result, then this (dead-end) run is excluded from the experiment (and so, from the measure). At N=40 and above the running time increases to a value, that several runs were above the limit. At N=50, not even 300000 individuals (150 generations times the population size) are enough to find the optimal solution. For this reason the algorithm is developed to use a convergence checker.

4.4 Test 4: Convergence checker – setting the convergence limit

A **convergence checker** is added to the algorithm, which restarts the search, if the fitness value of the best individual of the given population does not converge. In a given number of populations it is checked whether the best individual's fitness value is changing. If it does not change in CL (convergence limit) populations, the search restarts, hoping for a better initial population (which is likely, because of the high number of possible optimal solutions). Now we show results about tuning the value of the convergence checker.



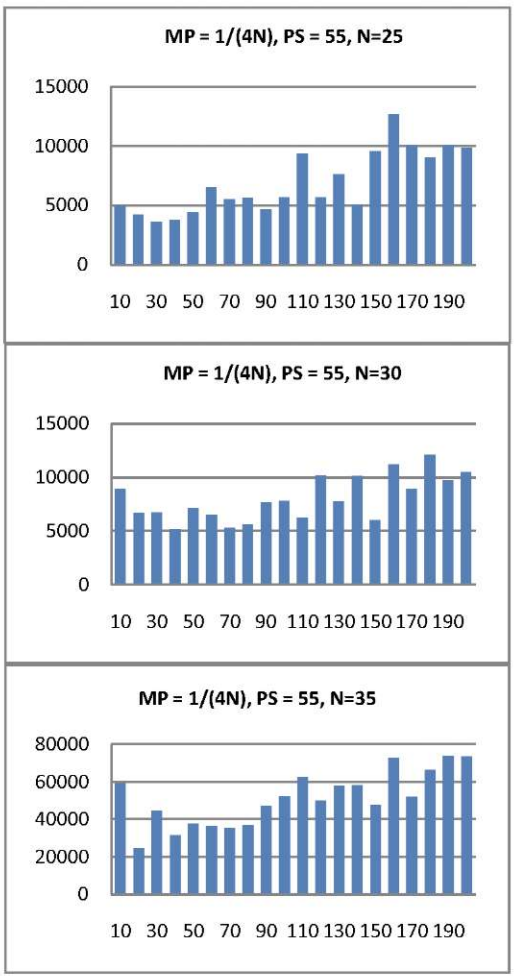


Figure 4. Running time as a function of convergence limit.

From this point, the search goes on until an optimal solution is found. In order to reach this, the program may restart the search as many times as needed. The value of CL = 40 is chosen, where all the four tests gave promising results.

5. THE RESULTS OF FINE-TUNING

Based on the previous experiences and test results, considering space complexity, the parameters of the **speed-optimized** search:

Probability of mutation:	$PM = \frac{1}{4N}$
Population size:	PS = 55
Convergence limit:	CL = 40

A run of the Java program on an average notebook computer takes

for N=30	0,747 sec,
for N=35	2,446 sec,
for N=40	3,7 sec,
for N=45	6,541 sec,
for N=50	22,587 sec

to find an optimal solution of a generated puzzle.

5.1 Test 3 revisited: How does running time change when increasing N?

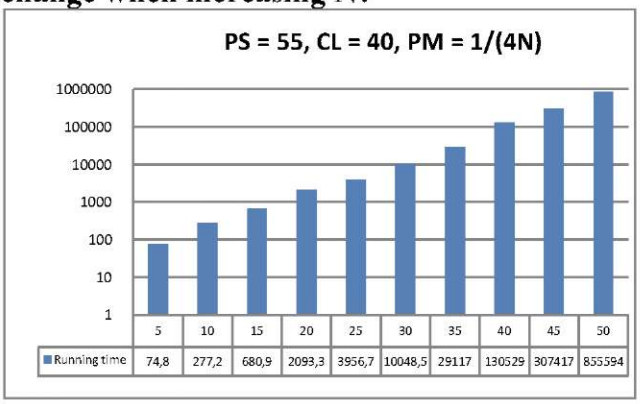


Figure 5. Running time as a function of N.

Revisiting test 3, with convergence checking and the best found parameters, the exponential time complexity is clearly shown. (The scale is logarithmic.) In this way each time the optimal solution is found.

6. FURTHER PLANS

To solve the zero mutation probability vs. dead-end problem, an extra function of the convergence checker should be added, which increases the probability of mutation, if the best fitness values do not converge. This way, a zero probability might be set as the initial value.

One might also think about changing the crossover method. However, this does not make that much difference in efficiency, since the bits of a chromosome are completely unrelated.

7. CONCLUSION

Logical puzzles, even seems-to-be-simple problems are closely related to satisfiability problems. In this paper a genetic algorithm was used to solve a specific type of such puzzles. Genetic algorithms usually have several parameters. In our case the tuning of these parameters has a dramatic effect on the performance. Initial parameters might even be quite far from the optimum. After probability of mutation and population size have been set, in some cases, searches may still come to a dead-end, failing to find an optimal solution at all. Convergence-checking enables the program to detect these dead-ends and restart the search. Examining the statistics of runs, the final parameter of dead-end detection can be set, which enables the application to always effectively find an optimal solution. We note here that the parameter CL may depend on the value of N, but analysing our statistics with N between 20 and 60 the dependence is not clear.

8. ACKNOWLEDGMENTS

The work is supported by the TÁMOP 4.2.1./B-09/1/KONV-2010-0007 project. The project is implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

9. REFERENCES

[1] Álmos, A., Győri, S., Horváth, G., Várkonyiné Kóczy, A. Genetikus Algoritmusok. Typotex, 2002.

[2] Coley, D.A. An Introduction to Genetic Algorithms for Scientists and Engineeres, World Scientific Publishing, 1999.

[3] Mitchell, M. An Introduction to Genetic Algoritms, MIT press, 1996.

[4] Nagy, B. SW-type puzzles and their graphs, *Acta Cybernetica* 16 (2003), 67-82.

[5] Nagy, B. Boolean programming, truth-teller-liar puzzles and related graphs, In *Proc. of ITI 2003, 25th International Conference on Information Technology Interfaces*, pp. 663-668. Cavtat, Croatia, 2003.

[6] Nagy, B. Duality of logical puzzles of type SW and WS - their solution using graphs, Pure Mathematics and Applications - PU.M.A. 15 (2005), 235-252.

[7] Nagy, B. Boole programozás gráfok segítségével (Boole programming and related graphs), Szigma 23 (2002), 115-130.

[8] Nagy, B., Allwein, G. Diagrams and Non-monotonicity in Puzzles. Diagrams'2004: Third International Conference on the Theory and Application of Diagrams, Cambridge, England, Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence, LNCS, LNAI 2980: Diagrammatic Representation and Inference, 82-96.

[9] Nagy, B. Truth-teller-liar puzzles and their graphs, Central European Journal of Operations Research - CEJOR 11 (2003), 57-72.

[10] Nagy, B. SS-típusú igazmondó-hazug fejtörők gráfelméleti megközelítésben (SS-type truthteller-liar puzzles and their graphs), Alkalmazott Matematikai Lapok 23 (2006), 59-72.

[11] Smullyan, R. What Is the Name of This Book? Prentice-Hall, 1978.

[12] Smullyan, R. The Lady or the Tiger? Alfred A. Knopf, Inc., 1982.

[13] Smullyan, R. The Riddle of Scheherazade: And Other Amazing Puzzles, Ancient and Modern. Knopf, 1997

Heuristics for the multiple depot vehicle scheduling problem

Balázs Dávid
Institute of Informatics
Faculty of Science and Informatics
University of Szeged
Árpád tér 2., 6720 Szeged, Hungary
+36 62 546396
davidb@inf.u-szeged.hu

ABSTRACT

In this paper, we introduce a greedy heuristic for the multiple-depot vehicle scheduling problem, which gives acceptable results with good running time for real-life instances. Our method is also compared to vehicle scheduling models and heuristics found in literature. All examined heuristics are tested on real-life data instances.

General Terms

Algorithms

Keywords

Scheduling problem, Vehicle scheduling, IP-based heuristics

Supervisor

Dr. Miklós Krész

1. INTRODUCTION

The vehicle scheduling problem is one of the most important problems arising in public transportation. The goal is to satisfy the trips of a given timetable under certain conditions. Literature has dealt extensively with this problem in the past decades, and real world applications of these scheduling methods are getting more and more attention.

The complexity of the problem arises from its size: solving a problem based on even a small or middle-sized city may not give a feasible solution in reasonable time (when aiming for an exact solution). A method needs an acceptable running time to be efficiently applied in the planning process of a bus company, as companies usually either plan for a longer period (running many instances one after the other), or have to make decisions quickly. An effective decision support system has to provide the results of several different algorithms for the same problem. The company can compare and analyze these results, and chose method that fits their problem the best.

However, such a system can only be used for planning purposes only if all its algorithms have a fast running time. There are several factors that influence running time, especially the model used to describe the problem, and its solution methods. Running time can also be improved further with the help of proper heuristics, which also have to be considered.

The outline of this paper is the following: first, the vehicle scheduling problem is presented along with two of its widely used IP representations. After this, efficient heuristics from the literature are introduced to the problem, each using a different approach to decrease the running time of the solution method. We also propose an own greedy heuristic approach for the problem, which is based on the idea found in [6]. These methods are tested on real-life instances (provided by the Szeged City Bus Company, Tisza Volán Zrt.), and the results are compared and analyzed. Finally, the conclusions are drawn and further research possibilities are addressed.

2. THE VEHICLE SCHEDULING PROBLEM

The vehicle scheduling problem satisfies a set of timetabled *trips* using a number of given *vehicles*. Every trip has a *departure* and *arrival time*, a *starting* and *ending location*, and a set of vehicles which are able to serve the trip.

The vehicles can be classified into different groups (called depots), each depot having a constraint on the number of its vehicles. Depot classification can depend on certain features of the vehicles (eg. solo, elongated, fuel type), or its geographical location.

Other types of trips also have to be considered for the problem: vehicles can use *deadhead trips* to get from a geographical location to another, and every vehicle schedule starts with a *pull-out trip* (from the depot), and ends with a *pull-in trip* (to the depot). These trips are also referred to as *unloaded trips*, because no passengers are transported on the vehicles during them. Timetabled trips will be referred to simply as trips the rest of the paper, if the context allows it (the names of other trips remain unchanged).

Our aim is to give a schedule, where each trip is executed exactly once, and the costs arising from the one-time daily cost, and the distance-based cost of the vehicles is minimal. In a real-life situation, other constraints may arise for the problem (eg. the refuelling of vehicles [2]). These are not considered in this paper, as it is enough to deal with basic vehicle schedule when planning for a longer time period.

According to the number of used depots, literature discusses two main types of the problem: single depot vehicle scheduling problem (SDVSP), and multiple depot vehicle scheduling problem (MDVSP).

2.1 Single and multiple depot

Complexity of the vehicle scheduling problem depends on its number of depots. SDVSP problems can be modelled as a minimal-cost flow, and even problems with several thousand trips can be solved efficiently with special algorithms.

As SDVSP problems can be solved in polynomial time [1], they are also useful for providing initial solutions or used as sub-problems for certain MDVSP heuristic approaches. While SDVSP problems can be solved easily, two or more depots results in an NP-hard problem.

MDVSP problems have been first studied by Bodin et al. [5], and their NP-hardness in case of two or more depots was shown by Bertossi et al. [4]. The MDVSP results in a number of vehicle schedules, where each given trip is satisfied exactly once, and the trips scheduled to a vehicle can be carried out from the same depot (the depot of the vehicle).

The vehicle scheduling problem can minimize the number of vehicles used, the amount of distance covered, or a cost function can be given which takes both aspects into consideration. In this case, each depot has to be given a *general vehicle cost* (the maintenance cost of a vehicle belonging to the depot) and an *operational vehicle cost* (which is in proportion to the distance covered by the vehicle). Let i be a vehicle schedule of the solution, and $d(i)$ denote the depot used to execute i. Considering a vehicle from depot $d(i)$ let the general vehicle cost be denoted by $gc(d(i))$, and the operational vehicle cost denoted by $oc(d(i))$. Furthermore, let $dist(i)$ be the distance covered by a schedule i. Assuming that we have n schedules, the cost of the problem is

$$\sum_{i=1..n} \left(gc\left(d(i)\right) + dist(i) \cdot oc\left(d(i)\right) \right),$$

which we want to minimize.

In addition to using this cost function, all models and methods introduced below are able to handle the capacity constraints of the depot. An overview of different models for both the single- and multiple depot vehicle scheduling problem can be found in [6]. We will introduce two of these models for the MDVSP.

2.2 Connection based IP model

IP models for the vehicle scheduling problem are widely used, and dealt with by the literature. One of the most well-know models is the connection based model.

Trips i and j are *compatible*, if there is a vehicle which satisfies i and is able to satisfy j afterwards. If the ending location of i is the same as the starting location of j, then the only condition is that the departure time of j has to be greater or equal to the arrival time of i. In other cases, we also have to take into account the time needed to carry out the deadhead trip connecting the two locations. Pull-in and pull-out trips of a depot are always compatible.

Using the above definitions, a vehicle schedule can be defined as a series of trips, where each timetabled trip is compatible with the subsequent trips. In practice, a vehicle schedule stands for the daily work of a vehicle. A feasible schedule always starts with a pull-out trip from a depot, and ends with a pull-in trip to the same depot. Other trips of the schedule can either be timetabled or deadhead trips.

The main idea behind the connection based model is to connect all compatible trips with edges.

The following notations have to be introduced: let D be the set of depots and T be the set of trips. For every $t \in T$ let $d(t)$ and $a(t)$ denote the departure and arrival time of trip t. The set of depots, which can execute a trip t is denoted by $g(t) \in D$. Let $T_d \subseteq T$ be the set of trips which can be executed from depot d. Similarly, every $d \in D$ has a $s(d)$ starting and $e(d)$ ending location. The set of nodes of our network will be the following:

$$N := \left\{ d(t) \cup a(t) \cup s(d) \cup e(d) \mid t \in T, d \in D \right\}$$

To define the edges of the network, let

$$A^d := \left\{ \left(d(t), a(t) \right) \mid t \in T_d \right\}$$

be the set of trips that can be served by depot d.

Let

$$B^d := \left\{ \left(a(t), d(t') \right) \mid t, t' \in T_d \text{ are compatible trips} \right\}$$

be the set of deadhead trips of depot d.

Let

$$P^d := \left\{ \left(s(d), d(t) \right), \left(a(t), e(d) \right) \mid t \in T_d \right\}$$

be the set of pull-in and pull-out trips of depot d.

Using the sets presented above, the set of edges of the network is:.

$$E := A^d \cup B^d \cup P^d \cup \left\{ \left(e(d), s(d) \right) \right\} \text{ for every } d \in D$$

where $(e(d), s(d))$ are the circulation edges of the network.

Based on the sets and notations introduced above, a feasible solution of the MDVSP can be determined for network (N, E). For every edge of the network, an integer vector x has to be defined. A vector component belonging to an edge e of depot d is denoted with x_e^d. Formalizing the problem:

$$\sum_{d \in g(t)} x^d_{(d(t), a(t))} = 1, \text{ for every } t \in T, \tag{1}$$

$$\sum_{e \in n^+} x_e^d - \sum_{e \in n^-} x_e^d = 0, \text{ for every } d \in D \text{ and } n \in N, \tag{2}$$

$$x_e^d \in \{0,1\}, \text{ except for circulation edges,} \tag{3}$$

$$x_e^d \text{ integer,} \tag{4}$$

where n^+ is the set of incoming edges to node n, and n^- is the set of outgoing edges from node n.

According to (1) each trip has to be executed exactly once, while (2) shows that every vehicle arriving to a geographical location has to leave that location. Constraints for the number of vehicles in each depot can be set as an upper bound for the circulation edges of the corresponding depot.

Any flow satisfying the conditions above can be a feasible solution of the problem. For an optimal solution, we have to minimize

$$\sum_e c_e x_e,$$

where c_e is the cost of edge e.

The main drawback of the connection based model comes from the fact that it represents connections between all compatible trips. The number of compatible trips is high even with a small number of timetabled trips, due to the possible deadhead trips and the number of depots. This makes the size of the problem so large that it can not be used effectively on real-life data, which consists of several thousand trips.

2.3 Time-space network

The time-space network has been introduced to vehicle scheduling by Kliewer et al. [7], and is able to efficiently solve even large-sized real-world MDVSP instances. As it was pointed out at the connection based model, the number of edges connecting compatible trips is high, while only a few of these are used in a feasible vehicle schedule. Dropping any of these connections would lose the optimality of the solution. However, the number of edges has to be reduced for solving the problem efficiently.

The time-space network solves the problem mentioned above. The model uses two dimensions, time and space. Space stands for the set of geographical locations, while time means timelines assigned to every location. The arrival and departure times are denoted on these timelines, with each point in time being a node of the model. As can be seen, the time-space network assigns the nodes to geographical locations as opposed to the connection based network. Apart from this difference, the set N of nodes of the network can be defined similarly to the connection based approach. A^d can also be given in a similar way for each depot $d \in D$, and P^d can be defined with the help of the time-lines associated with the depots.

The definition of deadhead trips is the main difference between the two models. The timelines used by the time-space network can be used to aggregate deadhead trips, with the help of newly introduced waiting edges connecting adjacent nodes on the timeline. This method significantly reduces the size of the problem. Waiting edges always connect two adjacent nodes on the appropriate timeline. Denoting the set of waiting edges with W^d for a depot $d \in D$, the set of edges of the network is

$$E := A^d \cup B^d \cup P^d \cup W^d \cup \{(e(d), s(d))\} \text{ for every } d \in D$$

The IP model of the network is similar to the one presented at the connection based model:

$$\sum_{d \in g(t)} x^d_{(d(t),a(t))} = 1, \text{ for every } t \in T, \quad (5)$$

$$\sum_{e \in n^+} x^d_e - \sum_{e \in n^-} x^d_e = 0, \text{ for every } d \in D \text{ and } n \in N, \quad (6)$$

$$x^d_e \geq 0, \quad (7)$$

$$x^d_e \text{ integer}, \quad (8)$$

where n^+ is the set of incoming edges to node n, and n^- is the set of outgoing edges from node n.

3. SOLUTION METHODS

Using the time-space network introduced in the previous section, we present some solution methods for the MDVSP. An important aspect studied in all of these methods is the applicability on real-life problems. For effective use in public transportation, solution should be obtained fast, and the cost should also be close to the optimal cost of the problem. First, the traditional approach of solving our problem with an MILP solver will be presented, then heuristics from the literature are introduced.

3.1 MILP solver

The first method we applied for solving the IP problem of the time-space network was using an MILP solver (in our case: SYMPHONY[1]). Due to the large size of the problem, only the first feasible solution was found. The cost of the resulting solution was nearly optimal (gap: 0,1-0,2% from the optimum), but running time was slow in many cases (even 1-2 hours for some instances).

Companies execute the method for a whole planning period, which usually is a time span measured in months. This planning period has several different day-types (11 in the case of Szeged), and the vehicle scheduling problem has to be solved for all of these. Solving the IP for all these problems would take a large amount of time.

As it can be seen above, a fast algorithm has to be applied for the problem, if we want to use it as the part of a decision support system. For this, we examined several different heuristics from literature for the problem.

3.2 Rounding heuristic

As the running time of the process mainly consists of finding the feasible integer solution of the relaxed LP-problem, accelerating the IP-solution process can result in a decrease in the running time of the algorithm.

The difference between the values of the LP-relaxation of the MDVSP, and the optimal integer solution is usually small, as large percent of the variables has an integer value in the solution of the LP. The heuristic approach proposed by Suhl et al. [9] aims to utilize this feature. The IP solver is stopped, when certain criteria are met (which can be e.g. the number of visited nodes, or the difference between the cost of the actual problem and the LP relaxation), and a rounding algorithm is executed for this partial solution. The algorithm uses two rounding intervals $[0, rl]$ and $[ru, 1]$, where $0 \leq rl \leq ru \leq 1$. Consider a variable x_j, and let $x_j - [x_j] = f_j$, where $0 \leq f_j \leq 1$. The value of x_j is rounded according to the following rules:

[1] http://www.coin-or.org/SYMPHONY/

- if $f_j \in [0, rl]$, then $\lfloor x_j \rfloor$
- if $f_j \in [ru, 1]$, then $\lceil x_j \rceil$

Several rounding iterations can be carried out sequentially, providing that the resulting LP still gives a feasible solution. If no variables were rounded, then the rounding intervals can be extended. After the rounding steps, the branch-and-bound IP solver is executed again for the LP. The process is repeated until all variables are fixed, or the problem has no feasible solution.

The gap from the optimum still remains around 0.2%, and there is a decrease in running time. However, the running time of some instances can still take around 1 hour to solve, which is inadequate time in certain cases.

3.3 Variable fixing heuristic

This heuristic is used to further decrease the size of the MDVSP. The basic idea of variable fixing is to solve simplified problems based on the original model, and look for series of trips that are common in all solutions. If such trips exist, they are considered as *stable chains*, and supposed that they also appear the same way in the global optimum. Stable chains are denoted in the model as one huge trip, thus significantly decreasing the size of the problem. After all stable chains have been found, the resulting smaller MDVSP can be solved with a MILP solver.

Kliewer et al. [8] proposed to use SDVSP sub-problems for all depots of our MDVSP as simplified problems. For a depot d, we solve the following sub-problem:

- The capacity of the depot is equal to the sum of all depot-capacities of the MDVSP.
- Only those trips are considered, which can be executed from depot d.

After each SDVSP sub-problems are solved, the solutions are used to create stable chains. Using these stable chains, a new MDVSP problem is created, which has the following properties:

- The number and capacity of the depots are the same as in the original problem.
- The set of trips of the new problem consists of the trips not included in any of the stable chains and a newly created trip for each stable chain. The cost of these new trips is equal to the sum of the cost of the trips that are in the stable chain they represent, and has the departure time and starting location of the first trip of the chain, and the arrival time and ending location of the last trip of the chain. These trips can be executed from any depot

Solving this MDVSP, the final solution can be acquired by substituting back the original trips instead of the stable chains.

4. A GREEDY VARIABLE FIXING HEURISTIC

The heuristic we propose is also based on the method of creating stable chains. Initially, an SDVSP problem is solved, which is created by transforming the MDVSP in the following way:

- The capacity of the depot is equal to the sum of the capacity of all the depots in the MDVSP.
- The SDVSP contains all the trips from the MDVSP, and all trips can be executed from the single depot.

This new SDVSP problem is solved, and stable chains are formed from this solution using a greedy method. The method assigns a value to the depots of the MDVSP using the following function:

$$\frac{1}{\varepsilon} \cdot d\_c(d) + d\_b(d),$$

where d_c(d) is the daily cost, and d_b(d) is the distance-based cost of depot d and parameter $\varepsilon > 0$ is an arbitrary real number. Our test cases use $\varepsilon = 210$. The average distance covered by a vehicle in a day around 210 km, thus the function above gives an estimate for the total cost of the vehicle to cover 1 km. Depots are ordered into a list L according to this cost.

For every depot d of the list L (starting with the one with the lowest cost), the greedy algorithm examines all the vehicle schedules in the result of the SDVSP. If subsequent trips are found which can be executed from this depot, they are considered as stable chains. These trips are flagged, and cannot be the part of other stable chains.

After all depots are examined, a new MDVSP problem is created. This procedure is similar to the one described in 3.3, the only difference arises when introducing the new trips. These trips can only be executed from depots that satisfy every trip of the stable chain that the new trip represents. Let D be the set of depots of the problem, and J be the set of all trips. Let V be a set to store all flagged trips, and S be a set containing chains of trips, which are the stable chains. Let L be a list of trips that is used to build up the chains. Using the notations above, the algorithmic code of the problem can be seen on Figure 1.

Greedy_variable_fixing

$D \leftarrow$ depots with assigned values

$S \leftarrow \varnothing, V \leftarrow \varnothing, L \leftarrow \varnothing$

while D is not empty, choose $d \in D$ with lowest value

  solve an SDVSP with all $j \in J$ trips

  $j =$ first trip of the solution, where $j \notin V$

  while j can be executed from depot $d \in D$, and $j \notin V$

    $L \leftarrow j$

    $V \leftarrow j$

    $j = next\_trip\_of(j)$ in the solution

  if $|L| > 1$

    $S \leftarrow L$

  else delete j from V

  $L \leftarrow \varnothing$

  Delete d from D

Return S

Figure 1: *The greedy variable fixing heuristic.*

As the running time of the heuristic was fast, further changes have been experimented with to improve the cost of the solution, with a minimal increase in the running time.

The more trips there are in a stable chain, the higher the chance is that the greedy method chooses an inadequate trip regarding optimality, making a cost of that stable chain higher. To avoid this, a constraint can be introduced to limit the length of the chains.

The SDVSP problem is created the same way as before, the only difference is in making the stable chains: if the number of trips in a chain reaches the constraint, the chain is not extended further.

Experience shows that limiting the length of the stable chains results in a solution with better cost, but has an increase in running time. Though there have not been enough tests concerning this observation yet, it looks a promising direction for further research.

5. TEST RESULTS

The above methods were tested on real-life data instances in the city of Szeged, Hungary. The company uses 14 different day-types (called combinations). For a normal planning period (2 months in the case of the company), all 14 combinations have to be calculated, and the schedule for a given day depends on the combination type of the day.

The results of the algorithms are presented on 4 day-types, and the properties of each can be seen in Table 1 below. The combinations with higher number of trips (szeged1 and szeged4) are working days of the week, while szeged2 and szeged3 are instances taken from a Sunday and Saturday, respectively.

The running time in the table is the time in seconds needed to find the first feasible solution using the SYMPHONY solver on the time-space network model. As mentioned before, the maximum gap of all first feasible solutions from the optimum could be calculated, and gave a value of 0,2%.

Table 1. Properites of data instances

	Number of trips	Running time (s)	"Bad" schedules
szeged1	2765	9503	1
szeged2	1826	1500	5
szeged3	2033	2837	6
szeged4	2758	7119	2

As can be seen, the running times of the instances are all above 1,5-2 hours. This is not acceptable in most cases, as the vehicle schedules are usually given for a longer planning period (several weeks or months), and calculating the schedules for all day-types would take more than a day this way.

The results of the heuristics will be examined using three aspects: the decrease in running time compared to the MILP solution, the maximum gap to the optimal solution and the structure of the schedules, which is explained below.

Decision support systems usually do not consider vehicle scheduling as a stand-alone problem, but use it as an initial input for driver scheduling and rostering. These problems have different working constrains of the drivers that have to be fulfilled, the most important is the maximum consecutive driving time without any rest, and the total length of the schedule given to the driver. Our vehicle schedules will also be analysed using this aspect, and if a schedule violates any of the two rules mentioned above (either by being "too long" for one driver, or "too dense" to give proper resting time), it is considered as a "bad" schedule regarding the driver modules. As can be seen from Table 1, the first feasible solutions give schedules with good structure in the aspect of drivers.

The results given by the rounding heuristic (see: Table 2) stay close to the original gap from the optimum, and also improve the running time significantly. There are even cases (see szeged3), where we get the exact same solution as the first feasible, but with an improved running time. The structure of the solution stays very similar to the ones in Table 1, so the number of bad schedules is also low (in the case of these 4 instances, it is exactly the same). However, running time is still around 1 hour in some cases, so this heuristic is still not suitable to make quick decisions. This heuristic is recommended to be applied when running time is not an important aspect, but it is guaranteed to give well-structured schedules with low cost.

Table 2. Results of the rounding heuristic

	Time ratio (%)	Max. gap to opt. (%)	"Bad" schedules
szeged1	38,0301	0,2584	1
szeged2	57,8000	0,2000	5
szeged3	66,0909	0,2000	6
szeged4	40,2304	0,2061	2

Results of the variable fixing heuristic show an even better decrease in running time: each instance can be solved around 5-15 minutes, with the cost of increasing the gap from the optimum. However, the structure of the resulting schedules is "bad" considering driver rules. This comes from the property that the algorithm produces really dense and really long stable chains (as there are many trips in our test cases that can be executed from all depots), which result in the schedules themselves being long and dense. Results can be seen in Table 3.

Table 3. Results of the variable fixing heuristic

	Time ratio (%)	Max. gap to opt. (%)	"Bad" schedules
szeged1	11,6069	0,3604	12
szeged2	16,4000	0,3110	11
szeged3	10,4688	0,6474	20
szeged4	9,5098	0,3557	12

The results of the greedy heuristic can be seen in Table 4. The heuristic needs only a couple of minutes (under 4-5 in all test cases) to finish. However, the gap from the optimal solution has risen to around 1-1,5%, which value is still acceptable considering a real life application. As opposed to the variable fixing heuristic, the greedy method fixes significantly more trips (~66% in comparison with ~33%) into stable chains, which significantly reduce the size of the problem. However, fixing trips to a stable chain means that their relation to other trips is already determined. Some trips will be placed in different schedules than it would be in the optimal scheduling, and

increasing the cost this way. Limiting the greedy choice with alternative constraints (e.g. limit the size of the chains, or the types of chosen trips) can lead to a solution with a better cost, but this would mean a larger problem size, which results in an increase in running time.
The structure of the schedules is also acceptable, the greedy heuristic stays close to the low number of bad schedules provided by the first two solution methods.

Table 4. Results of the greedy heuristic

	Time ratio (%)	Max. gap to opt. (%)	"Bad" schedules
szeged1	4,4828	1,3168	3
szeged2	9,4000	1,0379	7
szeged3	10,0458	0,3459	7
szeged4	4,8883	1,3176	2

6. CONCLUSIONS AND FUTURE WORK

We presented the vehicle scheduling problem, and examined models and heuristics from literature, which may prove to be useful on real-life instances, and especially in the decision planning process of a transportation company. A main criterion of these heuristics was a good running time.

With the help of the examined methods, a new greedy heuristic was introduced for the problem, which gives a significant decrease in running time, while its gap from the optimum stays at an acceptable level. It is also important to note, that the structures of these schedules are also good, considering the further application of them as an input to driver scheduling and rostering.

Test results show that our heuristic can be improved further. The present greedy choice has numerous properties that can be limited, and this way cost can be improved with the decrease of the running time. This method is still being tested and analysed.

Alternative options can also be introduced for the greedy choice instead of the current cost-based approach. Such methods (eg. based on the number of depots that are able to serve the trips of a chain) are currently being developed..

7. ACKNOWLEDGEMENT

This paper was partially supported by Szeged City Bus Company (Tisza Volán, Urban Transport Division).
I would also like to thank Zsuzsanna Papp and József Békési for their help with the implementation of the time-space network and the rounding heuristic.

8. REFERENCES

[1] Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. Nemhauser, Rinnooy Kan, and Todd, IV, 211-369, 1989.
[2] Balogh, J., Békési, J., Galambos, G., Krész, M.: An Assignment Model for Real-World Vehicle Scheduling Problem with Refueling. Proceedings of the 9th Workshop on Models and Algorithms for Planning and Scheduling Problems, Abbey Rolduc, The Netherlands, June 29 - July 03, 2009, pp. 229
[3] Békési, J., Brodnik, A., Krész, M., Pas, D.: An Integrated Framework for Bus Logostics Management: Case Studies. *Logistik Management* 5., pp. 389-411, 2009.
[4] Bertossi, A.A., Carraresi, P., Gallo, G.: On Some Matching Problems Arising in Vehicle Scheduling Models. *Networks* 17, pp. 271-281, 1987.
[5] Bodin, L., Golden, B., Assad, A., Ball, M.: Routing and Scheduling of Vehicles and Crews: The State of the Art. *Computers and Operations Research* 10., pp. 63-212., 1983.
[6] Bunte, S., Kliewer, N.: An Overview on Vehicle Scheduling Models.
[7] Kliewer, N., Mellouli, T., Suhl, L.: A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research* 175., pp. 1616-1627, 2006.
[8] Kliewer, N., Mellouli, T., Suhl, L.: Solving large multiple-depot multiple-vehicle-type bus scheduling problems in practice. *OR Spectrum* 27. pp. 507-523, 2005.
[9] Suhl, U.H., Friedrich, S., Waue, W.: Progress in solving large scale multi-depot multi-vehicle-type bus scheduling problems with integer programming. *Wirtschaftinformatik Proceedings*, Paper 81, 2007.

BAYESIAN GAMES ON A MAXMIN NETWORK ROUTER

Grigorios G. Anagnostopoulos
Department of Electrical and Computer Engineering, Democritus University of Thrace
67100 Xanthi, Greece
Telephone: 2541020486 (0030)
griganag@ee.duth.gr

ABSTRACT
In this paper, we consider network games with incomplete information. In particular, we apply a game-theoretic network model to analyze Bayesian games on a MaxMin router. The results show that the MaxMin mechanism induces desirable Nash equilibria.

Categories and Subject Descriptors
F.2.2 **[Theory of Computation]**: Analysis of Algorithms and Problem Complexity—Nonnumerical Algorithms and Problems; G.2.2 **[Mathematics of Computing]**: Discrete Mathematics—graph theory, network problems

General Terms
Algorithms, Theory

Keywords
Bayesian games, MaxMin fairness, Network games, Nash equilibrium, Algorithmic Game Theory

Sypervisor: Pavlos S. Efraimidis

1. INTRODUCTION
The interaction of independent Internet flows that compete for common network resources like the bandwidth at routers has recently been addressed with tools from algorithmic game theory. In this context, the flows are considered independent players who seek to optimize personal utility functions, such as the bandwidth. The mechanism of the game is determined by the network infrastructure and the policies used. The solution concept commonly used is the Nash equilibrium (NE), i.e., a state of the game in which no player has anything to gain by unilaterally changing her strategy.

An assumption underlying Nash equilibria is that each player holds the correct belief about the other players' actions.

To do so, a player must know the game she is playing. However, in many situations the participants are not perfectly informed about their opponents' characteristics: firms may not know each others' cost functions, players-flows competing for bandwidth of a network may not know the exact number of active flows etc. The model of a Bayesian game [4], generalizes the notion of a strategic game to cover situations in which each player is imperfectly informed about some aspect of her environment relevant to her choice of an action. The players may be uncertain about the exact state and instead assign a probability distribution over the possible states

The model that we will use, the Window-game (presented in [2]), has a router and N flows, and is played synchronously, in one or more rounds. Every flow is a player that selects in each round the size of its congestion window. The congestion window of a TCP flow is a parameter that determines the maximum number of packets that the flow is allowed to have outstanding in the network at each moment [1]. Real TCP flows adjust their congestion window to control their transmission rate. The router of the Window-game (the mechanism of the game) receives the actions of all the flows and decides how the capacity is allocated. The utility or payoff P(i) of each flow i is equal to the capacity that it obtains (transmitted packets) from the router in each round minus a cost g for each dropped packet, that is:

$$P_i = \text{transmitted}_i - g \cdot \text{dropped}_i \qquad (1)$$

We assume that the cost g for each dropped packet is constant for a certain game and is the same for all players. The general form of the games we study is a Window-game with a router of capacity C and N players. The game is played in one round and each player i chooses its window size $w_i \leq C$. The router policy is MaxMin. The cost for a lost packet is g. The number of the active players (players that can submit packets) of each round is a random variable n. The uncertainty stems from the fact that players-flows may not know the total number of players that are active in each round but only a corresponding probability distribution. Let w=C/n be the Fair Share (FS) of each player when n active players compete for a capacity C. In case of overflow, a MaxMin fair router satisfies any request that does not exceed FS and splits the remaining router capacity (if any) to flows with larger windows. The complete information version of this game has been studied in [2].

In practice, real TCP flows are actually playing a repeated game, which means that the flow will, in most cases, have an estimation of its fair share from the previous rounds. The probability distribution over the possible states of the game is a way of adding this partial information to our model.

We first analyze the NE of some simple toy-case scenarios and then proceed to the main result of this work; an expression for symmetric NE of an interesting class of symmetric Bayesian Window-games. For simplicity, we will assume in this work that all FS values are integers.

2. TOY CASES OF NON-SYMMETRIC GAMES

We present some toy-cases of non-symmetric Bayesian games on a MaxMin router.

Game 1. Two players A and X and router capacity C=12. Player A is always active, while player X receives a signal that informs her weather she is active or not. The probability of each case is 1/2. While both players know all the above information, the contents of X' signal is not revealed to A.

Solution sketch. All strategies with w<FS=6 are strictly dominated by the strategy w=FS and, thus, can be excluded. For g=0, all the remaining strategies (6≤w≤12) give the same constant payoff P(w)=6 for both players, since the whole capacity is used and they both transmit a window size equal to FS=6, with zero cost for the lost packets (if w>6). By definition, player X can be active or not. When we refer to player X without any other specification, we will implicitly refer to the active type of X. The same holds for the rest of this work. For g>0, player X knows that A will play at least the FS. Thus, a capacity equal to FS remains available for X. There is no incentive for X to leave the w=6 strategy, since a greater w will only reduce her payoff due to the cost of lost packets. Player's A payoff is equal to the sum of the products of her payoff at each of the two possible states with the probability of each state. It can be shown that these are the NE (w_A,w_X) of the game:

Table 1. The NE of Game 1

g	g=0	0<g<1	g=1	g>1
NE	$(12, x)\forall x \geq 6$	$(12,6)$	$(x,6)\forall x \geq 6$	$(6,6)$

Game 2. Same as Game 1, but with an additional player B, who, just like A, is always active.

Solution sketch. With an analysis similar to Game 1, it can be shown that these are the NE (w_A, w_B, w_X) of the game:

Table 2. The NE of Game 2

g	g=0	0<g<1	g=1	g>1
NE	(x,y,z), $\forall x,y \geq 6$, $\forall z \geq 4$	(6,6,4)	(x,y,4) $\forall x,y \in [4,6]$	(4,4,4)

Comments. It is clear that with MaxMin, more information leads generally to NE with better outcomes for players with this information. In many similar scenarios, the choice of g>1 generally gives fair NE.

3. A GENERAL SYMMETRIC (GS) GAME

Game GS1. Game with N players in which the number of active players is uniformly distributed in {1, 2, …, N}. In this game, each player receives a signal that informs her if she is active. Every active player assigns equal probability (p=1/N) to each of the N possible game states, where 0,1,2…, or N-1other players are active, respectively.

For each of these N different cases of the Bayesian game, there is a different FS. For example if N=5 and C=60:

Table 3. The possible FS values of Game GS1

n	1	2	3	4	5
FS	60	30	20	15	12

Due to the symmetric nature of the game, we will focus on its symmetric NE. Thus, we assume symmetric profiles for the game, i.e., all active players are using the same strategy w. In case of symmetric profiles, if the window size w is w>FS, then, for each flow, a number of packets equal to the FS will be transmitted, while for the remaining (w–FS) dropped packets there will be a reduction of the payoff proportional to the cost factor g. More precisely, the payoff of each flow will be:

$$\text{payoff} = \text{FS} - (\text{w} - \text{FS}) \cdot \text{g} = \text{FS} \cdot (g+1) - w \cdot g$$

In the cases where w≤FS, the payoff is equal to the window size, so payoff=w.

If we analyze every scenario of this game, we notice that the payoff function (PF) of each flow is a piecewise linear function of w. The following analysis illustrates how to determine the PF for the above example with N=5 and C=60.

1. For window sizes w, such that w≤12:

There are 1,2,3,4 or 5 active players with probability 1/5 for each case. In all these cases, the window size is smaller than the equivalent FS. So in any of these cases payoff=w. Thus, the expected payoff P(w) is:

$$P(w) = \frac{1}{5}w + \frac{1}{5}w + \frac{1}{5}w + \frac{1}{5}w + \frac{1}{5}w = w$$

2. For window sizes w, such that 12<w≤15:

In the case where all 5 players are active, then w>FS=12 and

$$\text{payoff} = \text{FS} \cdot (g+1) - w \cdot g = 12 \cdot (g+1) - w \cdot g$$

In the remaining cases with 4, 3, 2 and 1 active player(s), payoff=w, because the equivalent FS of each case is greater than any possible value of w in (12, 15]. Thus:

$$P(w) = \frac{1}{5}w + \frac{1}{5}w + \frac{1}{5}w + \frac{1}{5}w + \frac{1}{5}[12\,(g+1) - wg] \Rightarrow$$
$$P(w) = \frac{4}{5}w + \frac{12}{5}(g+1) - \frac{1}{5}wg = w\left(\frac{4-g}{5}\right) + \frac{12}{5}(g+1)$$

Working in a similar way, we obtain that:

3. For 15<w≤20 the payoff is:

$$P(w)=\frac{1}{5}w+\frac{1}{5}w+\frac{1}{5}w+\frac{1}{5}[15\,(g+1)-wg]+\frac{1}{5}[12\,(g+1)-wg]\Rightarrow$$
$$P(w)=\frac{3}{5}w+\frac{(12+15)}{5}(g+1)-\frac{2}{5}wg=w\left(\frac{3-2g}{5}\right)+\frac{27}{5}(g+1)$$

4. For 20<w≤30 the payoff is:

$$P(w)=\frac{1}{5}w+\frac{1}{5}w+\frac{1}{5}[20(g+1)-wg]+\frac{1}{5}[15(g+1)-wg]+\frac{1}{5}[12(g+1)-wg]\Rightarrow$$
$$P(w)=\frac{2}{5}w+\frac{(12+15+20)}{5}(g+1)-\frac{3}{5}wg=w\left(\frac{2-3g}{5}\right)+\frac{47}{5}(g+1)$$

5.For 30<w≤60 the payoff is:

$$P(w)=\frac{1}{5}w+\frac{1}{5}[30(g+1)-wg]+\frac{1}{5}[20(g+1)-wg]$$
$$+\frac{1}{5}[15(g+1)-wg]+\frac{1}{5}[12(g+1)-wg]\Rightarrow$$
$$P(w)=\frac{1}{5}w+\frac{(12+15+20+30)}{5}(g+1)-\frac{4}{5}wg=w\left(\frac{1-4g}{5}\right)+\frac{77}{5}(g+1)$$

In the above analysis we can observe how the PF changes with respect to the intervals that are defined by the possible FS values. By generalizing the above analysis to any number of players N, we obtain that:

$$P(w)=w\left(\frac{X-Yg}{N}\right)+\frac{Z}{N}(g+1),\ \forall w>\frac{C}{N}$$
$$P(w)=w,\ \forall w\le\frac{C}{N} \qquad (2)$$

where X,Y and Z are variables, whose values depend on the parameters of the game instance.

Theorem 1: In a MaxMin game with capacity C and cost g, and a random number n of active players, where n is uniformly distributed in 1,2,.., or N, the payoff function (PF) for symmetric profiles is:

$$P(w)=\frac{1}{N}\left[w(a-(N-a)g)+(g+1)\sum_{i=1}^{N-a}\left(\frac{C}{N+1-i}\right)\right],$$
$$\forall a=1,2\ldots N-1\ ,\ \frac{C}{a+1}<w\le\frac{C}{a}$$
$$P(w)=w,\ \forall w\le\frac{C}{N}$$

Proof: We will derive closed form expressions for the terms X, Y and Z of Equation 2. The term X is the number of the "w/N" terms, which is simply the number of the cases (game states) where w<FS. On the other hand, Y is the number of "-wg/N" terms, which corresponds to the number of the cases where w≥FS. So, we have shown that X+Y=N.

For N players, the possible FS values define a set of N-1 consecutive intervals. For α=1,2,..,N-1, let s_α be the interval between the FS's for n=α+1 and n=α active players. That is, s_α is the following interval:

$$\frac{C}{a+1}<w\le\frac{C}{a}$$

Note, that α gives the number of possible FS values (number of the game-states) where w≤FS at the corresponding s_α interval. That is because for $w\in s_\alpha$,

$$w\le\frac{C}{a}<\frac{C}{a-1}<\frac{C}{a-2}<\ldots<\frac{C}{1}$$

As noted earlier, the number of cases where w≤FS is equal to X. Thus, X=α. From X+Y=N we conclude Y=(N-α). In the analysis of the PF for N=5 and C=60, as well as for any N and C, we observe that Z is the coefficient of the term "(g+1)/N" at every interval for w, defined by the possible FS values. The value of Z is equal to the sum of all the FS's whose value is lower than the possible values of w at the specific interval. For example, for N=5, C=60 and 30<w≤60, the coefficient of "(g+1)/N" is (12+15+20+30), which corresponds to the sum

$$\frac{C}{N}+\frac{C}{N-1}+\ldots+\frac{C}{a+1}$$

In general, variable Z is equal to the sum

$$Z=\sum_{i=1}^{N-a}\left(\frac{C}{N+1-i}\right)$$

Theorem 2: For α=1,2,..,N-1, let g_α be

$$g_a=\frac{a}{N-a}$$

Then, the symmetric profile w=C/α gives the best possible payoff, if

$0\le g\le g_1$, when α=1,
$g_{\alpha-1}\le g\le g_\alpha$, when α=2,3,…,N-1.

If g is greater than g_{N-1}, then w=C/N gives the best possible payoff. There is an optimal strategy w for each such interval of g. Note that when g has one of the boundary values g_1, g_2,…,g_{N-1} it belongs to two intervals in the above definition. For boundary values of g, all values of w in the closed interval defined by the two optimal values for w, give the best possible payoff.

Proof: Differentiating P(w) with respect to w (for each interval s_α, since P(w) is a piecewise function) of Theorem 1 and setting it equal to zero we obtain:

$$\frac{dP(w)}{dw}=0\Rightarrow a-(N-a)g=0\Rightarrow g=\frac{a}{N-a}$$

By solving the above equation for each of the s_α intervals, we can find the value of g for which the slope of P(w) is zero. Low values of the cost factor g provide incentives for the flow to play aggressively. Consequently, for very low values of g, starting from zero, the slope of P(w) is positive for all w<C. For larger values of g, strategies with large window sizes will suffer larger penalties for dropped packets. There is a boundary value of g (dP(w)/dw=0) at which P(w) remains the same for all values of $w \in s_1$. Any greater value of g will make the slope of P(w) at s_1 negative.

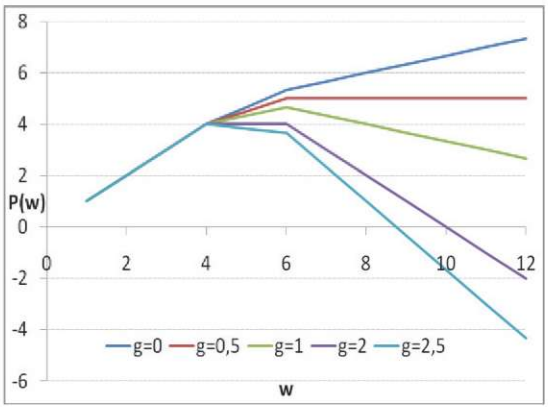


Figure 1. P(w) for several g values (C=12 ,N=3).

In Figure 1, each curve represents the function P(w) for a specific value of g. The curve (g=0) gives the best P(w) for every w among all curves, since it incurs no penalty. For all other curves, we observe that the slopes gradually decrease as the value of w increases, and in some curves the slope becomes negative (e.g. g=1, for w in {6,7,…,12}).

The curves of Figure 1 reveal a simple procedure to find the optimal symmetric profile for each corresponding value of g. Given a specific value for g, we can start from interval s_1, which involves the greatest values of w, and check one-by-one the slope of all the intervals, in search of the interval with the largest values of w with non-negative slope. The upper bound of this interval will give the **best possible payoff.**

In the special cases where the derivate equals zero, all values of w in that interval achieve the best possible payoff. In the rest of this work, when the best payoff is achieved by an interval of values, we will simply work with the upper bound of the interval. The upper bound is always one of the possible FS values.

Theorem 3: The symmetric profiles proposed by Theorem 2 are symmetric NE of the game.

Proof: To prove that these strategy sets are NE, we will show that no player has an incentive to unilaterally differentiate her strategy. Let w_k be the common strategy with the best payoff and player A the only player who changes her strategy from w_k to $w_A \neq w_k$.

1. Let $w_A < w_k$.

Let FS_i be the FS when i players are active. Assume i such that:

$$FS_{i+1} < w_A \leq FS_i$$

Player A will receive a payoff equal to:

- $\mathbf{w_A}$, for each state with j=1,2,…,i active players
- $\mathbf{[FS_j - (w_A - FS_j)\cdot g]}$, for each state with j=i+1,..,N active players.

This is because the transmission of the window size is guaranteed if the window size does not exceed the FS of the state. If w_A exceeds the FS of the state, since the common strategy w_k is $w_A < w_k$, all players play above the FS, therefore they all receive the FS (of the state) minus the cost of their extra packets which are dropped.

The way we calculate the expected payoff of player A as a weighted average of the payoff of each possible state is given as the sum of exactly the same terms that are summed in the general equation of Theorems 1. Thus, when player A unilaterally differentiates her strategy from w_k to w_A, where $w_A < w_k$, she receives the same payoff as if all players had chosen as common strategy the symmetric profile with window size w_A. In Theorem 2 we proved that the symmetric profiles that are proposed give the best possible payoff among all symmetric profiles.

Consequently, for one player who unilaterally differentiates her strategy from the symmetric profile w_k to a strategy w_A it holds:

$$P(w_A) \leq P(w_k), \forall w_A < w_k$$

2. Let $w_A > w_k$.

Let k be the number of players that corresponds to the fair share FS that is equal to the best symmetric strategy w_k ($FS_k = C/k = w_k$), as given by Theorem 2. Let $w_A = w_k + x$. We partition the field of possible values of w_A into intervals.

2.1. Let $w_A \in (w_k, 2w_k]$.

Then $x \in (0, w_k]$. Player A will receive a payoff equal to:

- $\mathbf{w_A}$, for each state with j=1,2,…,k-1 active players.
- $\mathbf{[FS_j - (w_A - FS_j)\cdot g]}$, for each state with j=k, k+1,..,N active players.

Among the states with 1,2,…,k-1 active players, the greatest congestion occurs at the state where k-1 players are active. At this worst case, all the other k-2 players choose FS_k, so the remaining capacity for player A is:

$$2 \cdot w_k \geq w_A = w_k + x$$

Thus, at these states player A has a successful transmission of her window size. On the other hand, when k players are active and all the other k-1 players chose FS_k, the remaining capacity for player A is FS_k. Similarly, for all other states with j=k+1,..,N active players, player A can send

successfully the FS_j of each state and will suffer the cost of her w_A-FS_j dropped packets.

The terms that we add to find the expected payoff of player A are equal to the terms that we need to add to calculate the expected payoff of symmetric profiles at the interval:

$$\frac{C}{k} < w \leq \frac{C}{k-1}$$

If all players chose a symmetric profile in the interval $(FS_k,FS_{k-1}]$, then they would successfully transmit their window size at the states where j=1,2,…,k-1 players were active, while at all other states they would successfully transmit the FS of each state. All symmetric profiles at this interval will give a lower payoff than the symmetric profile w_k, as shown in Theorem 2, and the linear function would be decreasing in this interval. Since the payoff for player A with strategy $w_A \in (w_k,2w_k]$ when all other players stay at w_k is given by the same decreasing linear function that gives the payoff of the symmetric profiles of the interval $(FS_k,FS_{k-1}]$, we conclude that:

When one player A unilaterally differentiates her strategy from the symmetric profile w_k to a strategy w_A, there is no

$$w_A \in (w_k,2w_k], \text{ such that } P(w_A)>P(w_k).$$

Comparing, at each possible state, player's A payoff when she chooses w_A to the payoff she receives when she stays at w_k (let p_{ki} be this payoff, where i is the number of active players of the state), given that all other players play w_k, we notice that with w_A player A receives:

- $\mathbf{p_{ki}+x}$, for all states with i=1,2,..,k-1 active players
- $\mathbf{p_{ki}-x \cdot g}$, for all states with i=k,k+1,…N active players

Since $P(w_A) \leq P(w_k)$,

$$P(w_A)-P(w_k)=[x(k-1)/N]-[x \cdot g(N-k+1)/N] \leq 0. \quad (3)$$

2.2 Let $w_A \in (h \cdot w_k,(h+1)w_k]$, $\forall h \in (2,3,...,k-1]$.

At these k-2 intervals, we compare again player's A payoff when she chooses w_k or w_A, given that all other players play w_k. We notice that with w_A player A receives:

- $\mathbf{p_{ki} + x}$, for all states with i=1,2,..,k-h active players, because even when the greatest congestion occurs at the state where k-h players are active, all the other k-h-1 players choose FS_k, so the remaining capacity for player A is: $(h+1) \cdot w_k \geq w_A = w_k + x$
- $\mathbf{p_{ki} + [C - w_k \cdot (i-1)] < p_{ki} + x}$, but with an extra penalty of $\mathbf{g \cdot [w_A-[C-w_k \cdot (i-1)]]>0}$ for all states with i=k-h+1,…,k-1 active players. Let $\mathbf{Q=g \cdot [w_A-[C-w_k \cdot (i-1)]]}$ be this penalty. Let Y such that $\mathbf{[C-w_k \cdot (i-1)]+Y=x}$. Then we will assume that she receives $\mathbf{p_{ki}+x}$ with an **extra penalty of Z=Q+Y.** (We add and subtract Y).
- $\mathbf{p_{ki}-x \cdot g}$, for all states with i=k,k+1,…N active players. In these states all players receive the FS_i of each state minus the cost of their dropped packets. Strategy w_A loses x more packets than strategy w_k.

So, for $w_A \in (h \cdot w_k,(h+1)w_k]$, there is:

- a linear increase for each extra packet, similar to the increase when $w_A \in (w_k,2w_k]$, for all states with i=1,2,..,k-h active players;
- a linear increase of the penalty for the dropped packets for each extra packet, for all states with i=k,k+1,…N active players;
- an extra penalty, previously named as Z, for all states with i=k+h,…,k-1 active players.

So, comparing the payoff of player A for w_A and w_k, given that all other players play w_k:

$$P(w_A)-P(w_k)=[x(k-1)/N]-[x \cdot g(N-k+1)/N]-Z, \quad (4)$$
$$\text{where } Z>0.$$

The difference of the first two terms $[x(k-1)/N]-[x \cdot g(N-k+1)/N]$ does not exceed zero, as shown in Equation 3. Since we subtract a non-negative value Z from a non-positive difference, we conclude that in Equation 4:

$$P(w_A)-P(w_k) \leq 0.$$

So, there is no $w_A \in (h \cdot w_k,(h+1)w_k]$, $\forall h \in (2,3,...,k-1]$, such that $P(w_A)>P(w_k)$.

Since there is no possible deviation that can offer an improved payoff to any player who unilaterally changes her strategy, **this strategy set (or else strategy profile) is a NE**.

4. CONCLUSION

We examined Bayesian Window-games on a MaxMin router and showed that the MaxMin queue policy leads to fair NE. The strategy profiles of the possible NE depend on the value of g. For plausible values of g (for example g=1), the NE strategy profiles utilize a sufficiently large part of the router capacity, while being at the same time absolutely fair.

Finally, we believe that the NE described in Theorem 3 are the only symmetric NE of the game. In particular, we believe that we can use the machinery of Theorems 1, 2 and 3 to show this result and intend to derive a proof in our future work.

5. REFERENCES

[1] Akella, A., Seshan, S., Karp, R., Shenker, S., and Papadimitriou, C. 2002. Selfish behavior and stability of the internet: a game-theoretic analysis of TCP. In Proceedings of the 2002 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications, SIGCOMM '02, 117-130.Ding, W. and Marchionini, G. 1997. A Study on Video Browsing Strategies. Technical Report. University of Maryland at College Park.

[2] Efraimidis, P. S., Tsavlidis, L., and Mertzios, G. B. Window-games between TCP flows. Theor. Comput. Sci. 411, 31-33, 2798-2817, 2010

[3] Papadimitriou, C. 2001. Algorithms, games, and the internet. In Proceedings of the Thirty-Third Annual ACM Symposium on theory of Computing (Hersonissos, Greece). STOC '01. ACM, New York, NY, 749-753.

[4] Martin J. Osborne, An Introduction to Game Theory, Oxford University Press, 2004, ISBN-10: 0195128958, ISBN-13: 978-0195128956.

An Extensible Probe Architecture for Network Protocol Performance Measurement

David Božič
Univerza na Primorskem
FAMNIT
Glagoljaška 8, Koper, SI 6000
+386 5 615 75 70
david.bozic@student.upr.si

ABSTRACT

The article describes the architecture, implementation, and application of Windmill, a passive network protocol performance measurement tool which enables the experimenters can measure Windmill in a broad range of protocol performance metrics both by reconstructing application-level network protocols and by exposing the underlying protocol layers' events, the range encompasses low-level IP, UDP and TCP events such as packet corruption and length errors, duplications, drops and the level of application performance. By correlating these inter-protocol metrics, the normally hidden interactions between the layers are exposed for examination. A special designed Windmill is a passive component of the larger Internet measurement infrastructure. Unlike most tools that focus on capturing data for post analysis, Windmill was designed to support continuous passive measurements at key network vantage points. The architecture allows application-level protocol data to be distilled at the measurement point for either on-line analysis or further post analysis. The extensible architecture enables experiment managers to vary the number and scope of Windmill's experiments.

Windmill is split into three functional components:

- a dynamically compiled Windmill Protocol Filter (WPF),
- a set of abstract protocol modules,
- an extensible experiment engine.

Fast WPF and the support for both experiment extensibility and high-performance protocol reconstruction are the key contributions of this architecture. Through the combination of dynamic compilation and a fast matching algorithm, Windmill's WPF can match an incoming packet with five components in less than 350 ns. Additionally, WPF addresses some fundamental limitations in past packet filtering technology by correctly handling overlapping filters.

Windmill enables the dynamic placement, management, and removal of long-running experiments, while accommodating the significant demands for protocol reconstruction performance. In order for the rational growth of the Internet to continue, a deeper understanding of the interactions between its protocols is needed.

Windmill can be used as an implementation of a passive application-level protocol performance measurement device and to explore these interactions in real-world settings.

In the article Windmill application is presented as a set of experiments where we can measured a broad range of statistics of an Internet Collaboratory, the experiments show the ability of the Windmill to perform on-line data reduction by extracting application level performance statistics, such as measuring user's actions, and demonstrated the use of the passive probe to drive a complementary active measurement infrastructure of the internet such as servers, without modifying end-host application or operating system code.

Keywords

Windmill, protocol filter, packet module, passive measurement, online analysis.

1. ARCHITECTURE

Windmill's functional components are:

- a dynamically generated protocol filter,
- a set of abstract protocol modules,
- an extensible experiment engine.

The organization of these components is shown in Figure 1.

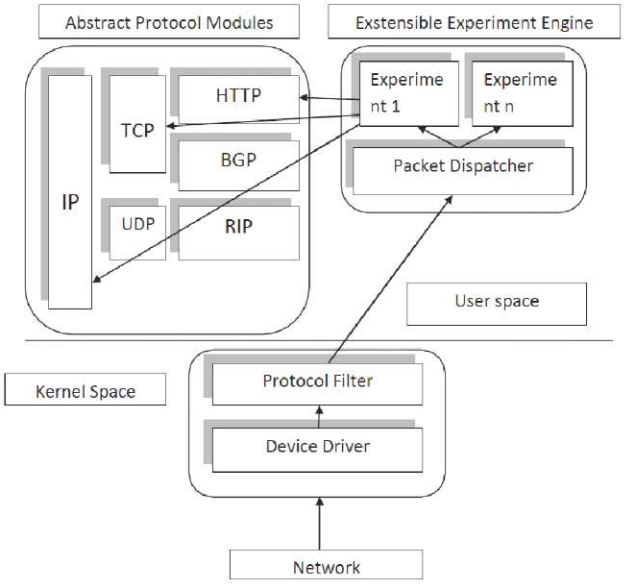


Figure 1. Organization of Windmill's architecture

Dynamically compiled filter matches the underlying network traffic against the WPF. This filter is constructed from the set of outstanding packet subscriptions from the engine's concurrent experiments. These individual subscriptions act as a filter for each experiment, they describe which packets the experiment is interested in receiving. The abstract protocol modules provide both efficient implementations of target protocol layers and interfaces for accessing normally hidden protocol events. Modules are composed to enable the efficient execution of the protocol stack on incoming packets. The extensible experiment engine provides an infrastructure for the loading, modification, and execution of probe experiments. Additionally, the experiment engine provides interfaces for the storage and dissemination of experimental results. Experiments are loaded into the experiment engine through an interface controlled remotely by the probe's administrator, once installed in the engine, an experiment subscribes to packet streams using the abstract protocol modules. Subscriptions are passed to the protocol filter which dynamically recompiles its set of subscriptions into a single block of native machine code. This code is installed in the kernel for fast matching and multiplexing of the underlying network traffic. Upon a packet match the protocol filter sends the packet along with the set of matching experiments to the packet dispatch routine in the experiment engine. The packets are then given to each matching experiment. Each experiment then uses the abstract protocol modules to process the standard protocol information in the packet. Rather than starting with the lowest layer, usually IP, the packet is given to the highest level protocol module, e.g. HTTP or TCP. These higher level protocol modules then recursively call the lower layers of the protocol stack.

After the experiment the results can extract the packet from processing any of the protocol layers. The results include the protocol frame or byte-stream service exported by the lower layers, or protocol events and error conditions triggered by the packet. The current implementation of Windmill is based on off-the-shelf software and hardware. A custom version of the FreeBSD 3.3 kernel serves as the base for the engine's software. Intel-based PCs serve as Windmill's cost-effective hardware platform. Currently, Windmill is being used with broadcast or ring-based datalink layers, including Ethernet and FDDI.

2. WINDMILL PROTOCOL FILTER

Windmill Protocol filter passively examines all the underlying network traffic and performs one-to-many packet multiplexing to the probe experiments this by constructing an intermediate representation of the outstanding subscriptions in the form of a directed-acyclic graph (DAG) which is dynamically compiled to a native machine language module, and is finally installed in the probe machine's kernel. As an example, a request for TCP traffic with source port A from experiment 1 and a request for TCP traffic on source port B from experiment 2 would result in the DAG shown in Figure 2.

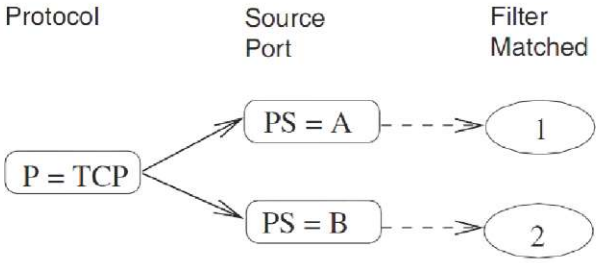


Figure 2. Simple DAG representation of two packet subscriptions.

Windmill Protocol filter can be different from past packet filters [1], where network packets are passively matched to a specification and demultiplexed to a single endpoint, in that it identifies a set of destinations for a packet. By determining a set of end-points, WPF avoids the subtle problem inherent in one-to-one matching algorithms of client starvation from overlapping filters.

One-to-many matching is motivated by the fact that a probe machine may be executing numerous concurrent experiments that are interested in some of the same packet streams. As the streams of packets arrive, the filter for each experiment must be used to determine which packets are

sent to which experiments. This can be done at reception time, where each packet is compared to different experiments' filters. Similar process can be using multiple BPF (Berkeley Packet Filter) devices to do the determination, one for each experiment. Packets can also be matched to experiments by determining a packet's destinations before its reception. Windmill Protocol filter adopts the latter approach in that it recomputed all possible combinations of overlapping filters when they are made, and generates a DAG to reflect these comparisons. Once the DAG is constructed, it is compiled to native machine language on-the-fly and installed in the kernel for matching against incoming packets.

A message header consists of a set of *comparison fields*. A filter is composed of a conjunction of predicates. Each predicate specifies a Boolean comparison for a particular field. An experiment registers a filter by supplying a set of values for one or more of these comparison fields. These fields can correspond to Internet protocol specific values, e.g. IP source address or TCP destination port, or they can be subsets of these values. This allows filtering based on fields such as the first 24 bits of the IP source address, commonly used to examine packets from a specific network.

	Bit comparison elements				
Experiment	IP Src Addr	IP Dst Addr	Protocol	Src Port	Dst Port
Filter 1	AS = X	*	P = T	PS = A	*
Filter 2	*	AD = Y	P = T	*	PD = B
Filter 3	*	AD = Z	P = T	*	PD = C
Packet to match	AS = X	AD = Y	P = T	PS = A	PD = B

Table 1. Three overlapping packet filters and a sample input packet

Table 1 shows how Windmill Protocol filter works, consider the set of filters, with each filter representing the packet subscription from one experiment. The table shows five comparison fields as the basis for three experiments. Each entry in the table specifies the value to be matched, or a "*" representing a wildcard. For example, the sample input packet above matches both filters 1 and 2.

The intermediate representation of these filters as a DAG is shown in Figure 3(a). The vertices represent Boolean operations on the comparison fields; a match results in a transition to the right. Furthermore, each vertex is also labeled with the set of corresponding filters when the Boolean operation associated with the vertex is true. Consider the vertex AS = X, which is labeled with the set {1, 2, 3}.

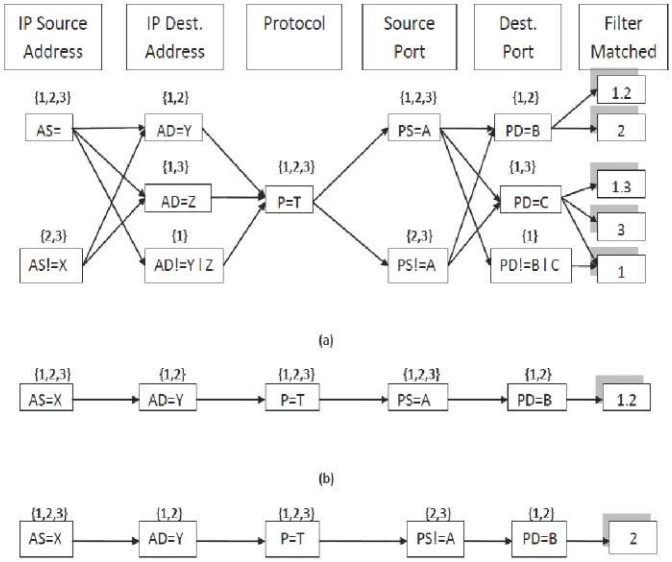


Figure 3. Part (a) represents the DAG generated from the three filters shown in Table 1. Part (b) shows an example of a packet matching filters 1 and 2. Part (c) follows a path through the DAG of a packet that only matches filter 2.

This indicates that if the IP source address (AS) in the input packet is X, then the input packet matches all three filters for the field in question. Consequently, each path through the DAG corresponds to matching the input packet with a unique set of the filters. For example, an input packet that matches the path in Figure 3(b) satisfies both filters 1 and 2, but not filter 3. Similarly, an input packet that matches the path in Figure 3(c) also matches filter 2, but not filters 1 and 3. Observe that the intersection of the set of labels associated with the vertices on a path identifies the unique set of filters that match input packets. To illustrate the subtle problem associated with packet filters that utilize most specific matching, reconsider the example in Table 1. Note that none of the three filters is more specific than the others.

The sample input packet above matches both filters 1 and 2. In both PathFinder and DPF, the packet filter will supply the packet to the experiment that matches first in the corresponding trie data structure [2,3]. This can lead to starvation of packet destinations whose filter is not the first to match an incoming element.

Using one-to-many matching DPF or PathFinder, we would need to use as many trie structures as experiments, resulting in $O(mn)$ time complexity, where m is the number of experiments, and n is the number of comparison fields. Once the subscriptions from each experiment have been combined into a master filter, the master filter is compiled into a single block of machine code and loaded into the running kernel. The goal of this compilation is to reduce the time needed to process each incoming packet.

Depending on the type of requests we expect to receive from the experiments, there are several alternative methods for performing this compilation.

The current version of the WPF assumes that the number of experiments is small enough for us to sacrifice space in the resulting code in exchange for speed. This results in filters that run as fast as other compiled filters.

3. ABSTRACT PROTOCOL MODULES

The abstract protocol modules export interfaces to probe experiments for both protocol reconstruction and the direct access to any protocol layer's events and data structures.

Through these interfaces, the abstract protocol modules provide for the breadth and depth of protocol analysis as well as inter-protocol event correlation. Network protocol layers are typically designed to hide the details of their underlying layers, and provide some type of data frame or byte stream service to the layers above them. The abstract protocol modules are very similar and can be chained together to build the service of higher layer protocols from the bare stream of data packets arriving from the WPF.

The experiment can read a TCP session's byte stream in either direction by only supplying the TCP module with captured packets.

The abstract protocol modules intentionally violate the encapsulation and abstraction of the lower protocol layers by exporting the details of these layers – including protocol events and data structures. The research experiments can then correlate this normally hidden data with the performance of higher layer protocols.

4. COLLABORATORY EXPERIMENTS

Experiments in this article demonstrate the use of Windmill in a real-world setting by instrumenting a key server in the Upper Atmospheric Research Collaboratory (UARC) to gather a broad range of statistics. The use of Windmill for on-line data reduction is illustrated by the collection of application level statistics. Specifically, we extract statistics from the application's data flows that could not be done using post analysis due to the volume of data. The experiments show how Windmill can be used in conjunction with an active measurement infrastructure to obtain snapshots of network metrics that can be temporally correlated with passive statistics. These statistics were gathered without modifying the UARC software or host operation systems; as such it represents an example of utilizing passive techniques for measuring shrink-wrapped systems. UARC is an Internet-based scientific collaboratory [5]. It provides an environment in which a community of space scientists geographically dispersed throughout the world performs real-time experiments at remote facilities.

Essentially, the UARC project enables this group to conduct team science without ever leaving their home institutions. This community has extensively used the UARC system for over four years. During the winter months, a UARC campaign - the scientists use the term campaign to denote one of their typically week-long experiments - occurs around the clock.

The UARC system relies on a custom data distribution service [4] to provide access to both real time and archived scientific data. In these experiments Windmill was deployed to measure one of the system's central data servers during the April 1998 scientific campaign. In order to provide ubiquitous access to the UARC system, users access the system through the Web via a Java applet. One consequence of this decision was the implementation of the data distribution as multiple TCP streams between UARC servers and the client browsers. During this experiment, Windmill intercepted all of the data communications between the main UARC server and its clients. By reconstructing the TCP and application-level sessions from these flows, Windmill extracted the UARC data.

The UARC system provides access to data from over 40 different instruments from around and above the world including the ACE, POLAR, JPL GPS, and WIND satellites, Incoherent Scatter Radar arrays in Greenland, Norway, Puerto Rico, Peru, and Massachusetts, magnetometers, riometers, digisondes and real time supercomputer models. These instruments supply over 170 distinct data streams to the scientists. The goal of our experiments was to obtain user-level performance statistics for analysis by behavioral scientists, such as when and to which instruments the users connected, and what time ranges of data they requested. These statistics can be correlated with chat room logs to model collaboration at a very high level. Similarly, we wanted to determine what effect a user's network connectivity had on her participation. A full analysis of these experiments is outside the scope of this article paper and this section focuses on how

Windmill made these measurements possible and only summarizes the findings. The passive measurement of any serial connection requires hardware intervention. The UARC server was originally connected to the Internet through a 100 Mbps switched Ethernet port on a Cisco 5500 router. For these experiments, we split the switched Ethernet by inserting an Intel Express 10/100 Stackable Hub between the router and the server. The perturbation of the system was the addition of an extremely small amount of latency. Windmill ran on a 300 MHz Pentium-II based PC with 128 MB RAM.

The use of Windmill for on-line data reduction is illustrated by the collection of user-level statistics for the UARC behavioral scientists. These statistics correspond to actions initiated by the users, including addition and removal of subscriptions to data suppliers, as well as requests for archived data.

In order to measure these application-level statistics, the UARC transport protocol was reconstructed. Like BGP, its frames are built on top of TCP and it uses a fixed header size with variable size data payloads. Only a small fraction of the application-level frames exchanged between the client and server describe user actions; the majority of the traffic is scientific data. Three days of continuous campaign throughput was reduced by over five orders of magnitude to approximately 200 KB of statistics.

The following is a subset of the questions that these measurements answered:
• Determination of the amount and level of synchronous collaboration. What are the duration and times when the scientists' view of the data overlapped, enabling concurrent analysis? This corresponds to determining when the scientists were in the same virtual room at the same time.

• Investigation into amount of cross specialization activity. Do the scientists focus only on the instrument and supply types that define their specialty, or do they exploit the wealth of data made available by the system?

• Temporal access patterns of the scientists. An analysis was done to determine whether the scientists have changed their access to their data. In the past, when they were collocated, they would all sit in a quonset hut and engage in science. Does this continue with a dispersed community?

• Access patterns to archived data. How the ability to access does over a year's worth of archived scientific data impact their real-time campaign? The experiment measured the access patterns to the archived data as well as the real-time supplies.

Windmill's passive measurement and analysis of underlying data streams is extremely powerful when coupled with active measurements. Windmill allows experimenters to define trigger events that can be used to initiate active measurements in an external tool. For example, when a flow's bandwidth drops below a threshold, an experiment could generate a trigger event that instructs an external tool to obtain a path snapshot through a mechanism such as traceroute.
If using passive measurement techniques for off-line analysis, it is not possible at analysis time to gather additional data about the state of the network due to the transient characteristics of Internet paths [6].

Pairing passive with active measurements, a broader range of statistics can be obtained. To illustrate this feature, in conjunction with the UARC data reduction experiment, Windmill ran an experiment that classified the different types of client–server connections. Specifically, when the experiment recognized a connection from a previously unconnected host, it sent a message to an active measurement probe running on the same machine that performed a path and ICMP RTT measurements on the client.

5. CONCLUSION

The experiment in this article has proved that the passive measurement techniques will become increasingly important as the commercial shift in the Internet continues. The ability to measure shrink-wrapped protocol implementations is critical due to the overwhelming deployment of commercially based protocol implementations in both the Internet's end-host and infrastructure nodes. Together, the inability to take the system off-line or modify it for study implies the need for increased passive measurement of Internet performance. Windmill was developed for precisely this purpose.

6. REFERENCES

[1] C. Labovitz, G. R. Malan, and F. Jahanian. Internet Routing Instability. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.
[2] K. Lougheed and Y. Reckhter. A Border Gateway Protocol (BGP). RFC 1163, June 1990.
[3] G. Robert Malan, F. Jahanian, and S. Subramanian. Salamander: A Push-based Distribution Substrate for Internet Applications. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.
[4] V. Paxson. End-to-End Routing Behavior in the Internet. In *Proceedings of ACM SIGCOMM '96*, August 1996.
[5] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. The SimOS Approach. *IEEE Parallel and Distributed Technology*, Fall 1995.
[6] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.
[7] J. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, A. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous Profiling: Where Have All the Cycles Gone? In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 1-14, Saint-Malo, France, October 1997.
[8] J. Apisdorf, K. Claffy, K. Thompson, and R. Wilder. OC3MON: Flexible, Affordable, High Performance Statistics Collection. In *Proceedings of INET '97*, Kuala Lumpur, Malaysia, June 1997.
[9] M. L. Bailey, B. Gopal, M. A. Pagels, L. L. Peterson, and P. Sarkar. PathFinder: A Pattern- Based Packet Classifier. In *Proceedings of First USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, November 1994.

[10] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel. Characteristics of Wide-Area TCP/IP Conversations. In *Proceedings of ACM SIGCOMM '91*, September 1991.
[11] K. C. Claffy, H. W. Braun, and G. C. Polyzos. A parameterizable methodology for Internet traffic flow profiling. *IEEE JSAC*, 1995.
[12] C. R. Clauer, J. D. Kelly, T. J. Rosenberg, C. E. Rasmussen, E. Stauning, E. Friis-Christensen, R. J. Niciejewski, T. L. Killeen, S. B. Mende, Y. Zambre, T. E. Weymouth, A. Prakash, G. M. Olson, S. E. McDaniel, T. A. Finholt, and D. E. Atkins. A New Project to Support Scientific Collaboration Electronically. *EOS Transactions on American Geophysical Union*, 75, June 1994.
[13] D. E. Comer and J. C. Lin. Probing TCP Implementations. In *Proceedings of the Summer USENIX Conference*, June 1994.
[14] S. Dawson, F. Jahanian, and T. Mitton. Experiments on Six Commercial TCP Implementations Using a Software Fault Injection Tool. *Journal of Software Practice and Experience*, 27(12):1385-1410, December 1997.
[15] D. Engler and M. F. Kaashoek. DPF: Fast, Flexible Message Demultiplexing using Dynamic Code Generation. In *Proceedings of ACM SIGCOMM '96*, August 1996.

Detection and visualization of visible surfaces

Danijel Žlaus
Student
Nova Cerkev 108
3203 Nova Cerkev
+38631212246
danijel.zlaus@uni-mb.si

ABSTRACT
This paper presents a new visible surface detection algorithm. The algorithm identifies which surfaces are visible from a set viewpoint with the utilization of the graphics processing unit (GPU). How we utilize the GPU is described in detail. This is an experimental approach for precise detection of visible surfaces in 3D scenes.

Categories and Subject Descriptors
I.3.3 [**Computer Graphics**]: Picture/Image Generation - *bitmap and framebuffer operations, display algorithm.*

General terms
Algorithms, Performance

Keywords
visible, hidden, surface, determination

Supervisor: Gregor Klajnšek

1. INTRODUCTION
The application of visible surface detection algorithms is twofold. Firstly, the algorithm can be used to detect which objects are visible and also to approximate the percentage of the visible surface area for each surface. Secondly, the algorithm can be used for removing unnecessary surfaces that are occluded and by doing so we can speed up our applications, because the occluded surfaces could have otherwise gone through heavy processing with advanced effects. Here we must emphasize that this is not a culling method since those tend to operate on groups of surfaces and try to determine visibility of objects. Another property of the culling methods is that they tend to run almost exclusively on the central processing unit (CPU).
The method we propose can determine which surfaces of the object are visible and is more suited for processing on the GPU. We have implemented the method using the OpenGL API.

2. SURFACES
In our method the term surface represents an individual polygon. The simplest polygon is a triangle and since that is sufficient to construct any kind of complex geometry, we saw no need for usage of more complex polygons such as tetragons, pentagons etc. Thus we only use triangle surfaces in our geometric data, but the method also works for arbitrary polygons. It is though recommended to keep the geometry simple by using only triangles, since this reduces a shortcoming of our method, which is described in 4.3.

3. DESCRIPTION OF THE ALGORITHM
The basic idea behind the algorithm is very simple and consists of three steps:

- Render each triangle with a unique color,

- download the rendered 2D image from the GPU,

- iterate though the 2D image and for every pixel map its color back to the triangle it represents and mark the triangle as visible.

3.1 Using the graphics processing unit (GPU)
Our method works by exploiting the color and stencil buffers of the GPU. These give us 39bits of data we can use per pixel. 32 bits come from the color buffer and 8 bits from the stencil buffer. The color buffer is comprised of four 8 bit channels, usually referred to as RGBA channels (red, green, blue and alpha). We reserved 1 bit for future use. The algorithm also exploits the Z-buffer, which automatically performs depth sorting of the geometry. All of these buffers are available on any modern GPU [1].

When the scene is to be rendered we have to attach the object and triangle information to every pixel. This can be accomplished in many ways. We chose a simple system where we first store the surface offset (**0 ... N-1** where N is the number of surfaces for a given object) and then the object offset (**0 ... M-1** where M is the number of objects in our scene) into the color and stencil buffer. This is accomplished by converting the offsets into a RGBA color and storing any remaining data into the accompanying stencil value. Once we have the color and stencil value, we can render that surface and the GPU will, using the Z-buffer, determine if any part of the surface is visible.

After the rendering of the geometry is finished we need to download the data stored in the color and stencil buffer from the GPU memory to the system memory [2] where we process the

result as a 2D image. The color and stencil buffers together carry information from which we can calculate the index of the object and the index of the object's surface to which the pixel belongs. Thus we can easily determine which surfaces are visible.

It is worth noting that these are the minimal steps required for implementing the algorithm and that presented method will also work with scenes that include dynamic objects. In the continuation we describe the techniques required to process scenes with richer geometry.

```
For each model in level
  For each polygon in model
    Generate unique color information
    Add the color to an auxiliary color buffer
  End for
End for
Upload the data to the GPU
```

Pseudo code for initialization

```
Cull the scene using ordinary techniques
Draw onto a hidden screen using auxiliary color buffers
Retrieve the drawn scene
For each pixel in scene
  Decode pixel
  Mark the appropriate polygon as visible
End for
```

Pseudo code for rendering

3.2 Utilizing the limited space

Since we have a fixed amount of available bits per pixel and two things we would like to store in the buffer, we have two ways as to how to divide it. Either we create a fixed division or we attempt to change the division when needed.

If we know the maximum number of objects and triangles per object that will appear in every future scene, beforehand, we can use fixed division. Since that scenario is highly unlikely, it is worth trying to find a better approach.

To use the limited space of the GPU buffers (39 bits) more efficiently we can check how many objects will be sent to the GPU just before rendering and how many surfaces each of these objects contains. Using this information we can dynamically set the division before each rendering. This is done using the following algorithm:

1. Determine the number N' of objects that will be rendered.
2. Determine which of the N' objects has the most surfaces - M' surfaces.
3. Calculate the least amount of bits needed to store numbers N' (n bits) and M' (m bits)
4. Calculate the optimal division point using the numbers n and m.

Using this approach a scene that renders 70000 objects leaves 22 bits available for the surface offsets. The number 70000 requires at least 17 bits to be stored. With the available space left we can have up to 2^{22} (slightly over 4 million) surfaces per object.

3.3 Optimization

It is recommended that we do some basic geometry culling, such as frustum culling [3] and perhaps more advanced techniques such as occlusion culling [4], before executing the surface visibility detection algorithm. This is advised, because it usually reduces the number of objects that need to be rendered and therefore reduces the number of bits required for storing the object index. It is possible that some scenes could not be processed correctly, if the number of objects and surfaces in the geometry is too big. By culling the scene before the rendering, we can potentially reduce the number of occurrences of this issue.

In our frustum culling method we use an axis aligned bounding box (AABB) coupled with a bounding sphere for each object to perform fast intersection testing. In addition, every object also has its geometry further divided with an octree, which can be used to for more precise culling [5].

We performed a test of the efficiency of the algorithm with and without culling. The comparison is shown in Figure 1. In the scene that was used for testing, culling had a mostly negative impact on performance because geometry of the scene was tightly localized. The only noticeable speedup occurred when just a small part of geometry was visible (viewpoint 7) and when no geometry was visible (viewpoint 8). Culling would have a more positive effect on the performance if the objects were more uniformly distributed.

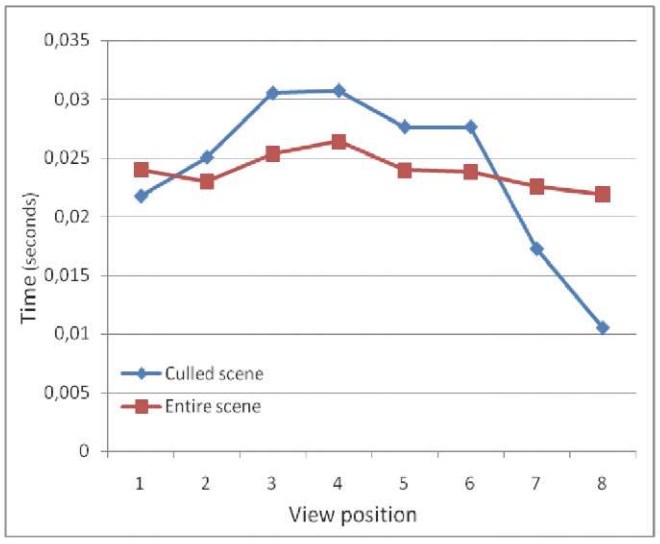


Figure 1 - Comparison of the performance of the algorithm with and without usage of culling.

4. DRAWBACKS

The proposed method also has some drawbacks, which need to be considered when the algorithm is used. In this chapter we give a detailed description of these drawbacks.

4.1 Performance scaling

Retrieving (downloading) data from the GPU memory into system memory is a slow operation and can therefore become a bottleneck of the method [2]. The higher rendering resolution we use the longer we need to wait for the transfer of data to finish, before we can start processing the data. A resolution increase of X times will result in an X^2 increase of the number of pixels in the rendered image. Figure 2 shows how the resolution of the image affects the actual performance of the algorithm.

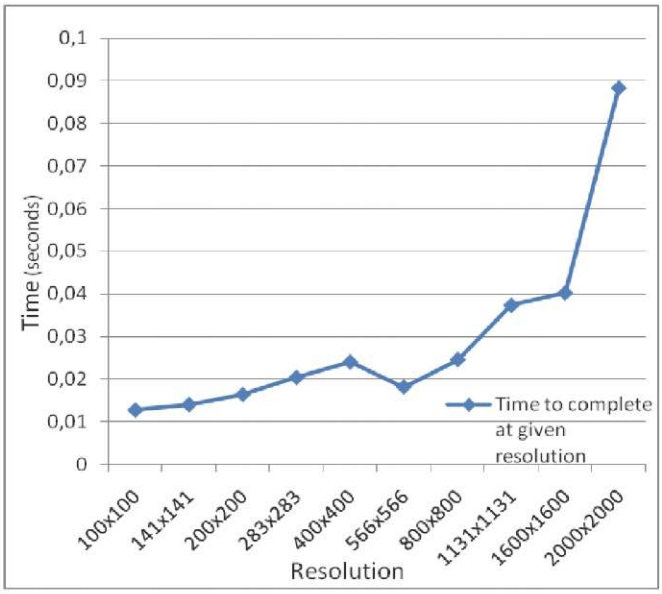


Figure 2 - Performance scaling with resolution increase

In the figure the resolution increases are $\sqrt{2}$ times the previous resolution so that the pixel count increases by a factor of 2 ($X = \sqrt{2}; X^2 = 2$). The graph matches our prediction and we can see that the run time starts to sharply rise at a certain resolution, as we would expect from an exponential function. The exact point when this sharp jump happens depends on the hardware configuration of the host PC and is especially influenced by the speed of the graphics card and the bus bandwidth.

4.2 Size of the viewport

The algorithm we propose requires two rendering pass and two rendering surfaces (images). One render surface is used by the visibility algorithm and the second render surface is used to render the view of the scene from the selected viewpoint. In continuation we will call the first surface the visibility surface and the second one the view surface.

The sizes of the visibility surface and the view surface need not be equal, but this can affect the preciseness of the algorithm. Usage of small visibility surface (smaller than the view surface) should result in a lot of surfaces getting marked as false negatives, since some surfaces will not be rendered to the visibility surface. An example of this is shown in Figure 3. The red surfaces have all gone undetected although they are visible, whilst the light green surfaces have been marked. Figure 4 shows the results of the visibility detection algorithm when the visibility surface is the same size as our view surface. Figure 5 shows the results of the algorithm when the visibility surface is bigger than the view surface. There are some undetected surfaces at the selected viewing resolution, but there are many more marked surfaces that we do not actually see at the selected viewing resolution. Figures 3, 4 and 5 show the same scene rendered from the same viewpoint with the same size of the view surface, only the size of the visibility surface was different.

Generally, there is no reason to use a visibility surface that is smaller than the view surface if the goal is to discover surface visibility. We should use the visibility surface that is at least the same size as our view surface. Usage of larger visibility surface is easily possible and also suggested since by using it we capture all the small surfaces in the distance that we would otherwise miss.

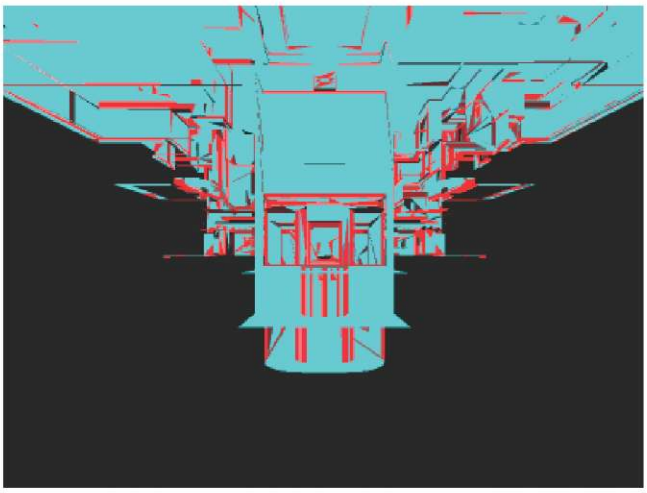


Figure 3 - Usage of the visibility detection algorithm using view window coupled with a smaller visibility results in a lot of false negatives (red colored surfaces)

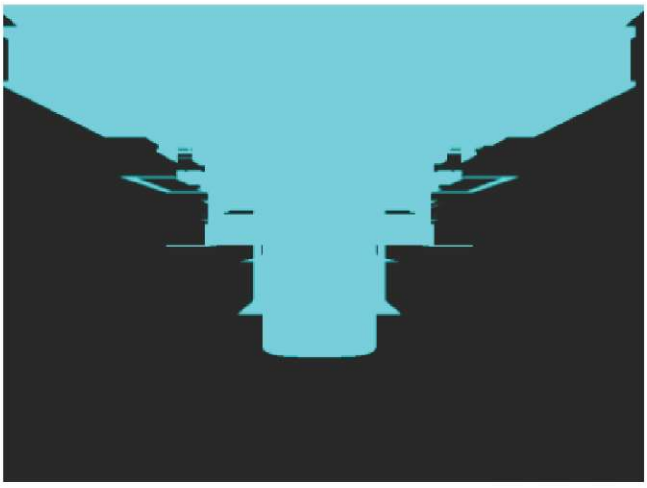


Figure 4 - Result of the visibility detection algorithm using view window coupled with a visibility surface that is the same size. There are no false negatives or false positives.

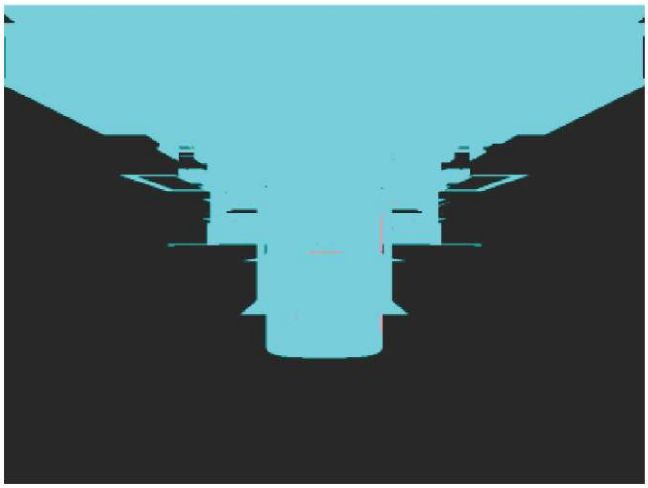


Figure 5 - Result of the visibility detection algorithm using view window coupled with a larger visibility surface. There are a few false negatives and a few false positives, which we do not see here, since the size of the view window is constant.

4.3 Triangle size

The method is also sensitive to the size of surfaces, since just one visible pixel of a large surface is enough to mark the whole surface as visible. This is especially emphasized on surfaces at the edge of the view area, where a large number of surfaces get marked as visible even though a only a tiny part of the surface is visible. If this is undesired there are methods that can be used to reduce this noise:

1. We can attempt suppressing the noise by counting how many pixels belong to each visible surface and then mark the surface as invisible if the number of pixels is under some preset threshold. The threshold would have to be determined depending on the type of geometry and the distance from the geometry. This method needs more post processing of the data, but does not require any changes of geometry of the scene.
2. We can preprocess each surface and break up large surfaces into smaller ones. This way we suppress the amount of noise generated on the expense of more complex scene geometry, but the post processing time is the same as for basic algorithm.

Which method is more appropriate depends on the host hardware and the scene geometry. However, even if we use these methods some noise in the output is unavoidable as it is very unlikely that every triangle detected by the algorithm will be fully contained inside the view frustum. Therefore we can only try to prevent the extreme case where large surfaces get marked as visible although only one pixel of the surface is visible.

5. OTHER METHODS

Today, many algorithms for detection of visible (or hidden) surfaces exist. One of the more popular methods is based on ray casting [6]. The method works by casting rays from the viewpoint into the scene geometry and finds the intersected surface that is closest to the rays point of origin and if such a surface exists, we mark it as visible. It is also possible that a ray does not intersect any geometry. As ray casting usually works in image space so one or more rays is cast into the scene for each pixel of the output image. Because of this the method is subjected to the same problems of resolution scaling as our method, since a resolution increase of X times will require at least X^2 times more rays to be cast. Each ray also has to do a lot of costly intersection testing operations before it finds the closest surface.

The method has several benefits compared to our method:

- does not require a GPU,
- very precise, since it calculates the points of ray-surface intersection,
- not limited by how many objects or triangles it can check,

but it also has several drawbacks:

- computationally heavy,
- complex implementation is needed for a fast solution,
- scales poorly with bigger scenes and resolutions.

5.1 CONCLUSION

We have designed and implemented a fairly robust and fast solution for finding all visible surfaces from a given point of view. Early results are promising and we can achieve real-time visualization of quite complex scenes, although the speed of the execution is very dependent on the host hardware and target resolution of the view window. Our future work will be oriented toward speeding up the method, so that we could perform real time rendering even for higher resolutions.

6. REFERENCES

[1] Buffers on GPU, Website http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson5.php (10.2.2010)

[2] Data readback from GPU, Whitepaper http://developer.nvidia.com/object/fast_texture_transfers.html (15.2.2010)

[3] Frustum culling, Website http://www.flipcode.com/archives/Frustum_Culling.shtml (3.6.2010)

[4] Occlusion culling, Website http://http.developer.nvidia.com/GPUGems/gpugems_ch29.html (5.6.2010)

[5] Octree data structure, Website http://en.wikipedia.org/wiki/Octree (13.5.2010)

[6] M. Berg. Ray shooting, depth orders and hidden surface Removal, Springer-Verlag, Berlin, 1993

Efficient approach for visualization of large cosmological particle datasets

Niko Lukač
University of Maribor
Faculty of Electrical Engineering and Computer Science
Smetanova ulica 17, 2000 Maribor, Slovenia
+386 02 220 7435
niko.lukac@uni-mb.si

ABSTRACT

This paper presents an efficient approach for visualizing large quantities of cosmological particle data. The data is represented as 3D points and originates from various sources. Due to the size of the cosmos, huge amounts of observed and simulated data can cause considerable problems for real-time visualization.

Our approach consists of preprocessing and visualization phases. In the preprocessing phase we organize the data for more efficient visualization. In the visualization phase we perform culling to reduce the number of particles that will be rendered. Compared to brute-force visualization approach, we can render more frames per second (FPS) and in most cases assure real-time visualization, without losing much visual quality.

Categories and Subject Descriptors

I.3.5 [**Computing Methodologies**]: Computational Geometry and Object Modeling – *Boundary representations, Geometric algorithm, Object hierarchies.*

General Terms

Algorithms, Design, Performance, Experimentation.

Keywords

Cosmological visualization, cosmological datasets, geometric algorithms, data preprocessing, octree volume, frustum culling, level of detail.

Supervisor

Borut Žalik
University of Maribor
Faculty of Electrical Engineering and Computer Science
Laboratory for Geometric Modelling and Multimedia Algorithms
Smetanova ulica 17, 2000 Maribor, Slovenia

1. INTRODUCTION

In cosmology it is common for astrophysicists to analyze cosmological data by observation. Computer based visualization enables better understanding of cosmos and extraction of useful information. Scientists often want to systematically analyze large amounts of data, what is impossible to do manually. Because the quantity and resolution of data increases each year, it is almost impossible to visualize large sets of data in real-time using brute-force rendering method, despite the advances in computer technology. Consequently new approaches that support real-time visualization are being developed.

Today there exist advanced interactive visualization toolkits [3, 5] and commonly used approach to achieve real-time visualization of large datasets is parallel based rendering by utilizing clusters of computers or supercomputers to balance the hardware load [1, 11]. Other visualization solutions employ various techniques using level of detail (LOD) methods [2], hierarchical division [10] or exploit advantages of GPU hardware [2, 10, 11].

2. COSMOLOGICAL DATASETS

Cosmological data available today originates from observations and n-body simulations. N-body simulations produce large amounts of artificial datasets using complex algorithms for simulating gravity force influence between the particles. The simulated data represents cosmic evolution and expansion, black matter distribution, galaxy evolution and other cosmological phenomena. Particles in n-body simulations often represent different cosmological objects depending on simulation type, ranging from dark matter fluid, gas clouds, galaxies to galaxy clusters. Simulated datasets are larger than datasets of objects from observable universe, due to the limitations of natural observations (e. g. large distances, radiation noise and cosmic expansion). Therefore computer based cosmological visualizations are mostly focused on million to billion point particle n-body simulated datasets.

The particle datasets are stored in different formats with different properties [8]. Most common properties are particle position (X, Y and Z coordinates), velocity (X, Y, Z) and mass. Using this information it is possible to make point based particle 3D visualization.

3. VISUALIZATION APPROACH

Our solution consists of several efficient methods that enable real time visualization of cosmological particle data by reducing the hardware load. To achieve this, the data is preprocessed before the visualization. In visualization phase, points are culled based on their visibility regarding the position and orientation of the camera. Custom frustum and LOD culling methods are used in order to speed up the rendering process [4].

3.1 Data Preprocessing

In preprocessing phase the data are firstly inserted into an octree structure, based on spatial positions, in order to achieve later visualization optimization. Octree structure recursively divides the space into eight subspaces. The leaf nodes of the tree structure contain the particles that are spatially close to position of the center of the node. Nodes of the octree spatially represent axis aligned bounding boxes (AABB). As Figure 1 shows, the denser the data is at certain subspace, the deeper is the hierarchical space subdivision. In order not to descend too deep into octree, where a single point can eventually become a leaf node, thresholds have been defined. The thresholds control the maximum amount of points per leaf node and maximum tree depth.

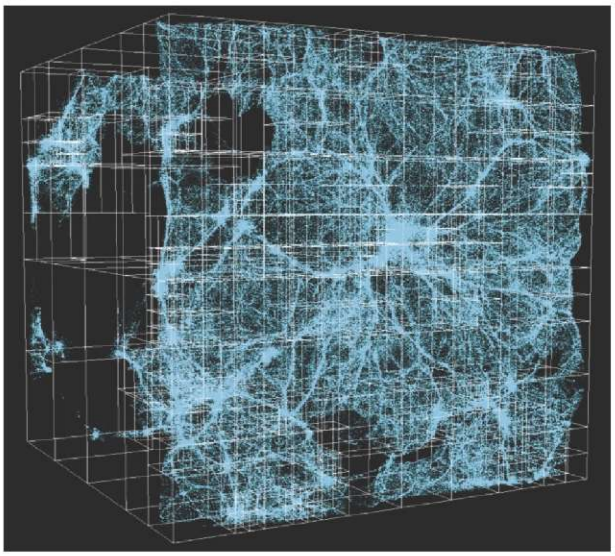


Figure 1. Visualization of sample cosmological particle data aligned in AABB octree nodes.

The next step is to order the data ordering in each node. Here we use a simple scrambling algorithm using random substitutions to achieve good random spatial distribution. This is important for optimization purposes due to the LOD usage later in the visualization phase (see section 3.2.2). Figure 2 presents an example of data ordering before and after scrambling. The scrambling is done until we have good distributed order of points in space, where isolated points are equally important as large clusters of points.

Because the data is often larger than computer system and GPU memory can hold, the data of each node is stored on the hard disk in a separate file. This enables efficient streaming of the data during visualization as the data can be read on the fly into the GPU memory when required.

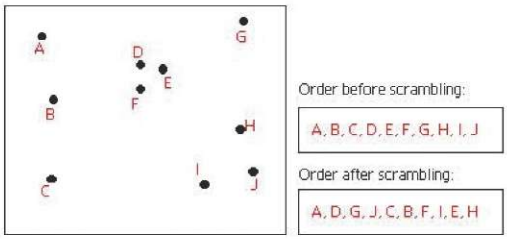


Figure 2. Example of saved data order after scrambling.

3.2 Visualization

In the visualization phase we cull particle points before rendering. Each time the observer's camera moves or orientates differently, we have to execute the culling methods. For optimization purposes we have preprocessed the data, therefore when culling algorithms are used, we only use spatial center points of the octree nodes to represent all other points in its subspace. Checking visibility of each point individually would be too slow and time consuming. For additional performance gain the culling is also performed recursively through the entire octree structure. This is based on a premise that if a parent node is outside viewing frustum all its children nodes are outside the frustum too. This allows skipping many nodes when culling. The culling phase is divided into two subphases: the frustum culling and LOD based culling.

3.2.1 Frustum Culling

This culling method is used to determine which points are within observer's visible space (viewing frustum). Geometric structure of the viewing frustum is shown in Figure 3. In order to perform frustum culling the planes of the viewing frustum are extracted [6] and represented in normalized plane equation form $Ax + By + Cz + D$.

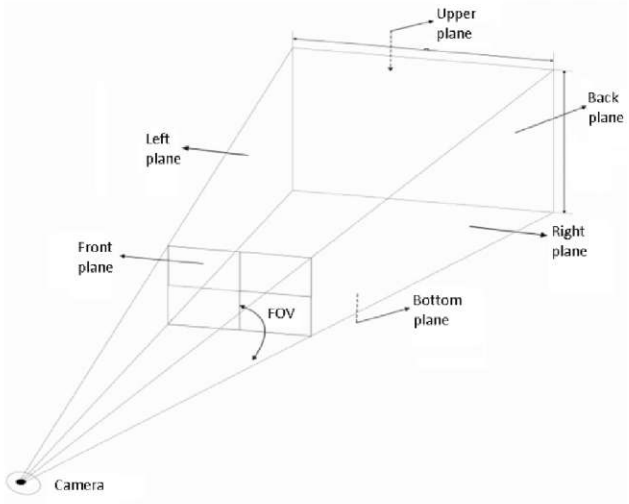


Figure 3. Geometric structure of the viewing frustum.

In order to determine the visibility of a node, custom frustum culling algorithm calculates at first a bounding sphere (BS) of the frustum and BS of the node and performs the intersection test between them. The BS-BS intersection test is very efficient and can quickly dismiss nodes that are outside viewing frustum. To execute the test we calculate the squared Euclidian distance from center point of a given node (point A) to center point of the

BS of the frustum (point B). If the distance is greater than the sum of radiuses of both spheres, then it is known that the given node lies outside frustum's BS and consequently outside the viewing frustum.

If the first culling algorithm determines that a given node intersects or lies inside the frustum's BS, then we perform a second culling algorithm using node's BS and planes of the viewing frustum. This test is executed six times; once for each plane.

If the center point of the node lies inside the viewing frustum we mark the node as visible and stop the frustum culling algorithm. Otherwise we have to perform another test, because part of the node can still be inside the viewing frustum even though its center point is outside.

The last culling test is performed between the node's AABB and the planes of the frustum. We perform the same calculation as in the second step, except that we now use all eight corners points of the node's AABB instead of the center point. This means that the calculation must be performed 48 times in the worst case. However, if we determine that one of the corner points is visible, we mark the node as visible and end the test.

As can be deduced from the description of the frustum culling algorithm, the visibility test is performed only for the nodes and not for single particles. We find it more optimal to end the culling algorithm once we determine that the node is partially visible, because the tests have proven that it is commonly faster to visualize all particles of that node, rather than check which ones are actually visible.

3.2.2 Level of Detail

After the frustum culling is done, we perform further culling using LOD method. With LOD we define how many points will be visible for a given node inside the viewing frustum. Thus we reduce the number of visible points and achieve better performance. As mentioned before, in the preprocessing section, the data is randomly scrambled, which allows employing a LOD culling method that loads the node data in the linear order. The goal of LOD is to visualize the dataset from a given view point by rendering fewer points than originally located in the viewing frustum space, without losing much visual quality by preserving overall structure of particle clusters (see Figure 4).

The used method measures squared Euclidian distance between the center points of the nodes and the position of the camera. With this information we define how many points have to be rendered for a given node with the following equations:

$$points_{\%} = 100 - \frac{current_{distance} * 100}{reference_{distance}} \quad \textbf{(3.1)}$$

$$points_{amont} = points_{N} * (points_{\%} / 100) \quad \textbf{(3.2)}$$

The consequence of our LOD culling method is that further from a given node observer is, the fewer points of the node are visualized. As a result, small details can disappear for the nodes that are far away from the camera. Usually, this is not a problem as these details would not be noticeable due to the density and number of the particles in the data. The reference distance in equation 3.1 is experimentally set to 3 times the radius of the root node of the octree, which represents the AABB of the whole dataset. This means that as a result of the equation 3.2 all points of a node far away from the camera would be culled, which could result in a significant loss of the visual quality. To prevent that, we can set a minimum threshold for the amount of particles in a node that have to be rendered.

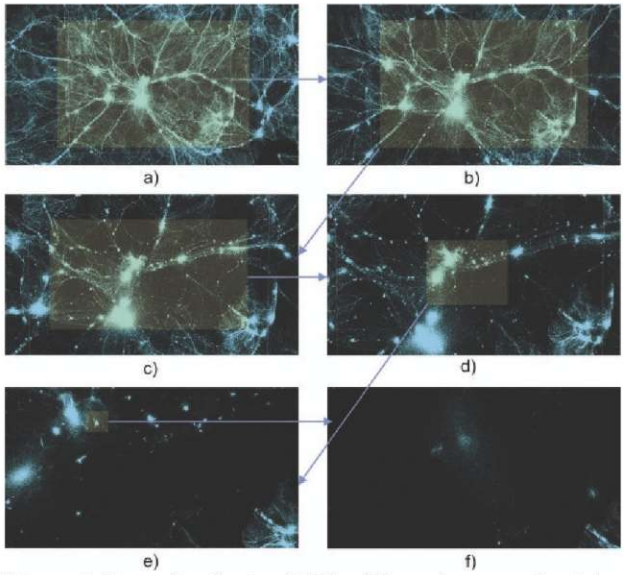


Figure 4. Example of using LOD culling when zooming into a specific region of sample cosmological particle data.

4. RESULTS

We have implemented our approach in C/C++ programming language using OpenGL graphical library for rendering. For GPU memory storage we used OpenGL Vertex Buffer Object (VBO) extension [9]. Each leaf node in the octree structure contains one or more VBO objects that hold the data of visualized points and possibly, if there is enough memory available, even non visualized points, in order to reduce possible data transfer times to GPU.

We compared our visualization method with classic brute-force (BF) approach. The comparison tests were executed on a common desktop PC with 256MB video memory and 4GB system memory. We performed a sample visual comparison between the two approaches as shown in Figure 5 below.

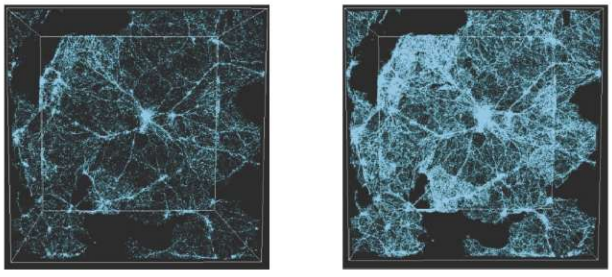


Figure 5. Visual comparison of a) 20% and b) 100% particle visualization.

For further comparison we tested the number of rendered frames per second (FPS) for both approaches. The comparison between the two approaches is shown in Table 1 and Figure 6. For each test different datasets of various simulations [9] with different number of particles have been tried. The FPS test was done

using a predetermined camera fly-through the visualized data. Camera position was initially aligned with the X coordinate axis, and it was away for two times data's highest octree root node radius. The camera was rotated for every 10th degree around the root node center point and after a full rotation the camera moved for 10% of initial distance toward the center point. This fly-through has been repeated until camera position was equal to the center point. By using this approach, different points of views have been tested. All the tests have been performed 10 times. Only small variations in FPS between the different runs have been noticed.

Table 1. Tests comparison results.

Particles #	BF approach [FPS]	Our approach [FPS]	Difference [%]
1.000.000	80	142	77.5%
5.000.000	75	139	85.3%
10.000.000	40	81	102%
15.000.000	33	53	60%
20.000.000	20	35	75%
35.000.000	5	26	420%
50.000.000	2	23	1050%
100.000.000	1	20	1900%

The tests results clearly indicate that our approach is considerably faster in all cases. The BF approach can keep up with real-time visualization as long as there is enough GPU memory. Our approach can still preserve real-time visualization by culling large amounts of points.

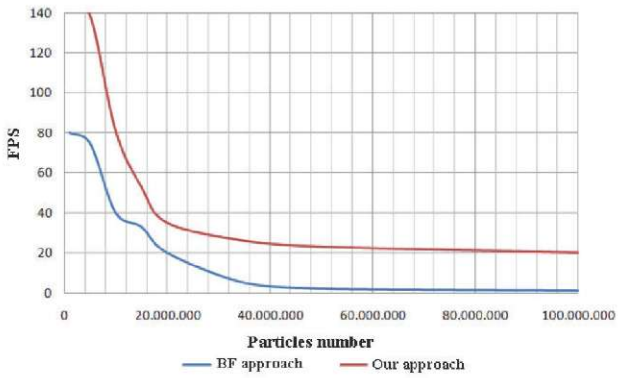


Figure 6. Graph of tests results from table 1.

5. CONCLUSION

In the paper we have presented an approach for efficient rendering of large cosmological particle datasets, which efficiently reduces hardware load. Although our solution can render significantly larger datasets than the brute force method, the number of particles that can be rendered in real-time on a single computer is always limited to some maximum by a hardware bottleneck. This problem is today usually solved by parallelization. We believe that our solution is also suitable for parallelization, which could be done by distributing octree nodes to different computers. Using this approach we could efficiently visualize even datasets that consist of billions of particles. Data ordering could be performed using a clustering algorithm, in order to increase the visualization quality.

6. REFERENCES

[1] Balazs Domonkos, Kristof Ralovich, Parallel Visualization of the Sloan Digital Sky Survey DR6, The 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, University of West Bohemia, Campus Bory, Plzen - Bory, Czech Republic, February 4-7, 2008.

[2] Carl Hultquist, Sameshan Perumal, Patrick Marais, Tony Fairall, Large-Scale Structure in the Universe, Technical Report CS03-16-00, Department of Computer Science, University of Cape Town, October 10, 2003.

[3] Daniela Ferro, Ugo Becciani, Vincenzo Antonuccio Delogu, Angela Germana, Astrophysical Data Analysis and Visualization Toolkit, F. Murtagh, G. Longo, J.-L. Starck, Di Ges´u, Eds. Astronomical Data Analysis III, Saint Agata on the Two Gulfs, 29. April - 1. May 2004.

[4] Fernando Randima ed., GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics, Addison Wesley Professional, (2004).

[5] Giancarlo Amati, Maura Melotti, Daniela Ferro, Gianluca Di Rico, Lugii Paioro, AstroMD and Cosmo.Lab visualizing astrophysicial data, Proceedings of Joint Eurographics - IEEE TCVG Symposium on Visualization, pp. 1-6, 2002.

[6] Gribb Gil, Klaus Hartmann, Fast Extraction of Viewing Frustum Planes from the World-View-Projection Matrix, June 2001, Online document: www2.ravensoft.com/users/ggribb/plane%20extraction.pdf (27.05.2010).

[7] Jasjeet Singh Bagla, Cosmological N-body simulation: Techniques, scope and status, Current science, Vol. 88, No. 7, pp. 1088-1100, 2005.

[8] MPA Numerical Cosmology Data Archive, Online: http://www.mpa-arching.mpg.de/ (20.06.2010).

[9] OpenGL VBO extension, Online: http://www.opengl.org/wiki/Vertex_Buffer_Object (16.03.2010)

[10] Tamas Szalay, Volker Springel, Gerald Lemson, GPU-Based Interactive Visualization of Billion Point Cosmological Simulations, CoRR, November, 2008.

[11] Zhefan Jin, Mel Krokos, Marzia Rivi, Claudio Gheller, Klaus Dolag, Martin Reinecke, High-performance astrophysical visualization using Splotch, Procedia Computer Science, ICS 2010, Vol. 1, No. 1, pp. 1769-1778, May 2010.

Computing the Longest Common Subsequence of Two Strings When One of Them is Run-Length Encoded

Shegufta Bakht Ahsan
AℓEDA Group, Department of CSE, BUET, Dhaka-1000, Bangladesh
plaban777@yahoo.com

Tanaeem M. Moosa
AℓEDA Group, Department of CSE, BUET, Dhaka-1000, Bangladesh
tanaeem@cse.buet.ac.bd

M. Sohel Rahman
AℓEDA Group, Department of CSE, BUET, Dhaka-1000, Bangladesh
msrahman@cse.buet.ac.bd

Shampa Shahriyar
AℓEDA Group, Department of CSE, BUET, Dhaka-1000, Bangladesh
shampa077@gmail.com

ABSTRACT

Given two strings, the longest common subsequence problem computes a common subsequence that has the maximum length. In this paper, we present new and efficient algorithms for solving the LCS problem for two strings one of which is run-length encoded (RLE).

1. INTRODUCTION

Suppose we are given two strings $X[1 \dots N] = X[1]X[2] \dots X[N]$ and $Y[1 \dots P] = Y[1]Y[2] \dots Y[P]$. Without the loss of generality, we can assume that $N \leq P$. A subsequence $S[1 \dots R] = S[1]S[2] \dots S[R], 0 < R \leq N$ of X is obtained by deleting $N - R$ symbols from X. A common subsequence of two strings X and Y is a subsequence common to both X and Y. The longest common subsequence problem for two strings, is to find a common subsequence in both strings, having maximum possible length. An interesting parameter for LCS problem is $\mathcal{R}$, which is the total number of ordered pairs of positions at which the two strings match. More formally, we say a pair (i, j), $1 \leq i \leq N, 1 \leq j \leq P$, defines a match, if $X[i] = Y[j]$. The set of all matches, $\mathcal{M}$, is defined as follows:

$$\mathcal{M} = \{(i, j) \mid X[i] = Y[j], 1 \leq i \leq N, 1 \leq j \leq P\}.$$

Observe that $|\mathcal{M}| = \mathcal{R}$. We use the notation $\mathcal{M}_i$ to denote the set of matches in Row i. Also, for the sake of better exposition we impose a numbering on the matches of a particular row from left to right as follows. If we have $\mathcal{M}_i = \{(i, j_1), (i, j_2), \dots, (i, j_\ell)\}$, such that $1 \leq j_1 < j_2 < \dots < j_\ell$, then, we say that $number((i, j_q)) = q$ and may refer to the match (i, j_q) as the qth match in Row i. Note that $number((i, j_q))$ may or may not be equal to j_q.

In this paper, we are interested to compute a longest common subsequence (LCS) of two strings when one of them is run length encoded (RLE) [12]. The motivation for using compressed strings as input comes from the huge size of biological sequences. In a string, the maximal repeated string of characters is called a run and the number of repetitions is called the run-length. Thus, a string can be encoded more compactly by replacing a run by a single instance of the repeated character along with its run-length. Compressing a string in this way is called run-length encoding and a run-length encoding string is abbreviated as an RLE string.

In what follows, we use the following convention: if X is a (uncompressed) string, then the run length encoding of X will be denoted by $\widetilde{X}$. For example, the RLE string of $X = bdcccaaaaaa$ is $\widetilde{X} = b^1d^1c^3a^6$. Note that for $\widetilde{X}$, we define $\widetilde{X}[1] = b^1, \widetilde{X}[4] = a^6$ and so on. The notation $|X|$ is used to denote its usual meaning, i.e., the length of X; the length of the corresponding RLE string $\widetilde{X}$ is denoted by $|\widetilde{X}|$. We will use small letters to denote the length of an RLE string; whereas capital letters will be used to denote the length of an uncompressed string. For example, if $|X| = N$, then we shall use n to denote the length of $\widetilde{X}$. Also, note that, the notion of a match and hence the definition of $\mathcal{M}$ can be extended in a natural way when one or both the strings involved are RLE. For example, the notion of a match $(i, j) \in \mathcal{M}$, is extended when one input is an RLE string as follows: if $\widetilde{Y}[i] = a^q$ and $X[j] = a$ then we say $(i, j) \in \mathcal{M}$ and $run((i, j)) = q$.

We will use $LCS\_RLE(X, \widetilde{Y})$ to denote an LCS of X and $\widetilde{Y}$. There has been significant research on solving the LCS problem involving RLE strings in the literature. Mitchell proposed an algorithm [7] capable of computing an LCS when both the input are RLE strings. Given two RLE strings $\widetilde{X}[1 \dots n]$ and $\widetilde{Y}[1 \dots p]$, Mitchell's algorithm runs in $O((\mathcal{R} + p + n) \log(\mathcal{R} + p + n))$ time. Apostolico et al. [2] gave another algorithm for solving the same problem in $O(pn \log(pn))$ time whereas the algorithm of Freschi and Bogliolo [4] runs in $O(pN + Pn - pn)$ time. Ann et al. also proposed an algorithm to compute an LCS of two run length

encoded strings [1] in $O(pn + \min\{p_1, p_2\})$ where p_1, p_2 denote the number of elements in the bottom and right boundaries of the matched blocks respectively. The version of the problem where only one string is run length encoded was handled recently by Liua et al. in [6]. Here, the authors proposed an $O(nP)$ time algorithm to solve the problem.

In this paper, we present two novel algorithms to compute $LCS\_RLE(X, \widetilde{Y})$. In particular, we first present a novel and interesting idea to solve the problem and present an algorithm that runs in $O(nP)$ time. This matches the best algorithm in the literature [6] for the same problem. Subsequently, based on the ideas of our above algorithm, we present another algorithm that runs in $O(\mathcal{R} \log\log p + N)$ time. Clearly, for $\mathcal{R} < pN/\log\log p$, our second algorithm outperforms the best algorithms in the literature. Notably, in our setting, Mitchell's algorithm would run in $O((\mathcal{R}+P+n)\log(\mathcal{R}+P+n))$ time, which clearly is worse than ours. Mitchell's algorithm could also be used in our setting with an extra preprocessing step to compress the uncompressed string. In this case, the cost of compression must be taken into account.

2. A NEW ALGORITHM

In this section, we present Algorithm LCS_RLE-I which works in $O(pN)$ time. Since our algorithm depends on some ideas of the algorithm LCS-I of [10, 11], we give a very brief overview of LCS-I in the following subsection.

2.1 Review of LCS-I

Note that LCS-I solves the classic LCS problem for two given strings X and Y. For the ease of exposition, and to remain in line with the description of [10, 11], while reviewing LCS-I (in this section) we will assume that $|X| = |Y| = N$. From the definition of LCS it is clear that, if $(i, j) \in \mathcal{M}$, then we can calculate $\mathcal{T}[i, j], 1 \leq i, j \leq N$ by employing the following equation [8]:

$$\mathcal{T}[i,j] = \begin{cases} \text{Undefined} & \text{if } (i,j) \notin \mathcal{M}, \\ 1 & \text{if } (i = 1 \text{ or } j = 1) \text{ and } (i,j) \in \mathcal{M}, \\ \max\{(\mathcal{T}[\ell_i, \ell_j])\} + 1 & \\ \quad 1 \leq \ell_i < i & \text{if } (i, j \neq 1) \\ \quad 1 \leq \ell_j < j & \text{and } (i,j) \in \mathcal{M} \\ \quad (\ell_i, \ell_j) \in \mathcal{M} & \end{cases} \quad (1)$$

Here we have used the tabular notion $\mathcal{T}[i, j]$ to denote $r(Y[1..i], X[1..j])$. In what follows, we assume that we are given the set of $\mathcal{M}$ in the prescribed order assuming a row by row operation. LCS-I depends on the following facts, problem and results.

Fact 1. ([9, 8]) *Suppose* $(i, j) \in \mathcal{M}$. *Then for all* $(i', j) \in \mathcal{M}$, $i' > i$, (resp. $(i, j') \in \mathcal{M}$, $j' > j$), *we must have* $\mathcal{T}[i', j] \geq \mathcal{T}[i, j]$ (resp. $\mathcal{T}[i, j'] \geq \mathcal{T}[i, j]$). □

Fact 2. ([9, 8]) *The calculation of the entry* $\mathcal{T}[i, j]$, $(i, j) \in \mathcal{M}$, $1 \leq i, j \leq n$, *is independent of any* $\mathcal{T}[\ell, q]$, $(\ell, q) \in \mathcal{M}$, $\ell = i$, $1 \leq q \leq N$. □

Problem 1. Range Maxima Query Problem. *We are given an array* $A = a_1 a_2 \ldots a_n$ *of numbers. We need to preprocess* A *to answer the following form of queries:*
Query: *Given an interval* $I = [i_s..i_e]$, $1 \leq i_s \leq i_e \leq n$, *the goal is to find the index* k *(or the value* $A[k]$ *itself) with maximum value* $A[k]$ *for* $k \in I$. *Ties can be broken arbitrarily, e.g. by taking the one with larger (smaller) index. The query is denoted by* $RMQ_A(i_s, i_e)$

Theorem 1. ([3, 5]) *Range Maxima Query Problem can be solved in* $O(n)$ *preprocessing time and* $O(1)$ *time per query.* □

Now, assume that we are computing the match (i, j). LCS-I maintains an array H of length N, where, for the current value of $i \in [1..N]$ we have, $H[\ell] = \max_{1 \leq k < i, (k,\ell) \in \mathcal{M}}(\mathcal{T}[k, \ell])$, $1 \leq \ell \leq N$. The 'max' operation, here, returns 0, if there does not exist any $(k, \ell) \in \mathcal{M}$ within the range. Now, given the updated array H, LCS-I computes $\mathcal{T}[i, j]$ by using the constant time range maxima query as follows: $maxIndex = RMQ_H(1, j-1)$, $\mathcal{T}[i, j] = H[maxIndex] + 1$. Because of Fact 1, LCS-I is able to maintain the array H on the fly using another array S, of length N, as a temporary storage. After calculating $\mathcal{T}[i, j]$, such that $(i, j) \in \mathcal{M}_i$, LCS-I stores $S[j] = \mathcal{T}[i, j]$. It continues to update S (and not H) as long as the computation continues in the same row. As soon as the processing of a new row begins, it updates H with new values from S. Due to Fact 1, LCS-I does not need to reset S after H is updated (by it) for the next row. Now, for the constant time range maxima query, an $O(N)$ time preprocessing is required as soon as H is updated. But due to Fact 2, it is sufficient to perform this preprocessing once per row. So, the computational effort added for this preprocessing is $O(N^2)$ in total. Therefore, LCS-I runs in $O(N^2)$ time.

2.2 LCS_RLE-I

In this section we present our first algorithm, namely LCS_RLE-I, to solve the LCS problem when one of the strings is an RLE string. Recall that, the notion of a match $(i, j) \in \mathcal{M}$, is extended when one input is an RLE string as follows: if $\widetilde{Y}[i] = a^q$ and $X[j] = a$ then we say $(i, j) \in \mathcal{M}$ and $run((i, j)) = q$. Following the idea of [10, 11], in the LCS_RLE-I algorithm, we maintain the arrays H and S and use them exactly the same way as they are used in LCS-I. We will be using another array $\mathcal{K}$ for the efficient implementation of our algorithm and its use will be clear as we proceed.

Now consider that we have completed the computation for the matches belonging to Row $i-1$ (i.e., $\mathcal{M}_{i-1}$) and we start Row i. Given the updated array H, assume that we are processing the match (i, j). Also assume that when the the computation of the match (i, j) would be complete, i.e. $\mathcal{T}[i, j]$ is completely computed, we would have the result of $LCS\_RLE(\widetilde{Y}'a^q, X')$, where $\widetilde{Y}'a^q$ and X' are prefixes of

$\widetilde{Y}$ and X respectively. Then, clearly, the match is due to the letter a. Now, if $q = 1$, then to compute $\mathcal{T}[i,j]$, we simply need to perform: $\mathcal{T}[i,j] = H[RMQ_H(1,j-1)] + 1$. We also need to update S array to store the new value of $\mathcal{T}[i,j]$ as the current highest value of the j column, i.e. we perform $S[j] = \mathcal{T}[i,j]$. In what follows, we refer to the above operations as the *baseOperation*.

If $q > 1$, then, we require two steps. Firstly, we perform the *baseOperation*. Then, in the second step (referred to as the *weightOperation*) we consider q previous matches in Row i, including the current one. If fewer matches are available we need to consider all of them. Now, note carefully that T-values for these matches have already been computed and reflected in the S array. We copy S to $\mathcal{K}$ and S array is never changed by any *weightOperation*. For Row i, we call $\mathcal{K}[k]$ to be a *match position* if $(i,k) \in \mathcal{M}_i$ and $\mathcal{K}[k]$ and k are referred to as the corresponding $\mathcal{K}$-value and $\mathcal{K}$-index (similar notations are also defined for the arrays H and S). Now, we add a *weight* to each of the corresponding $\mathcal{K}$-values: the *weight* is 0 for the current match, 1 for the previous match, 2 for the match before it and so on. Now observe that $\mathcal{T}[i,j]$ will be the maximum of these values. This is because the k^{th} element of this "window" from right, corresponds to matching k a's from the run a^q with rightmost k a's from X' and then matching the remaining substring with $\widetilde{Y}'$.

We will use array $\mathcal{K}$ to do this computation efficiently. Now recall that we are handling the match $(i,j) \in \mathcal{M}_i$. Clearly, we can implement the *weightOperation* by adding the appropriate weights at the corresponding *match positions* of $\mathcal{K}$ and then performing the query $RMQ_{\mathcal{K}}(u,j)$, such that $\mathcal{K}[u]$ is a *match position* (due to (i,u)) and $q = number((i,j)) - number((i,u)) + 1$. In what follows we will refer to the above range (i.e. the range $[u \ldots j]$) as the *weighted query window*. However, in this strategy, we may need to adjust the weights every time we compute a new match since for each match the *weighted query window* may change. Undoubtedly, this would be costly. In what follows, we discuss how to do this more efficiently.

Rather than adding the appropriate *weight*, for a particular row, we will add *relative weight* to all the *match positions* of $\mathcal{K}$. This would ensure that the position of the maximum value remains the same, although the value may not. To get the correct value, we will finally deduct the appropriate difference from the value. We do it as follows. After the *baseOperation*, we copy the array S to array $\mathcal{K}$. Then, to a match (i,ℓ), $1 \le \ell \le |\mathcal{M}_i|$ we add $|\mathcal{M}_i| - number((i,j)) + 1$ as the *relative weight*. In other words we give weight 1 to the rightmost match, 2 to the next one and so on and finally, $\mathcal{M}_i$ to the first match.

Now recall that we are considering the match $(i,j) \in \mathcal{M}_i$, i.e., we are computing $\mathcal{T}[i,j]$. Assume that $number((i,j)) = |\mathcal{M}_i| - k + 1$, i.e., this is the k^{th} *match position* from right. As before, we execute the query $RMQ_{\mathcal{K}}(q,j)$, such that $\mathcal{K}[u]$ is a *match position* (due to (i,u)) and $q = number((i,j)) - number((i,u)) + 1$. However, this time we need to do some adjustment as follows. It is easy to realize that each of the values of the matched positions in $\mathcal{K}[u..j]$, is k higher than the actual value. So, to correct the computation we perform $T[i,j] = \mathcal{K}[RMQ_{\mathcal{K}}(q,j)] - k$.

The analysis of the algorithm is similar to that of LCS-I algorithm of [10, 11]. As we need to do at most two RMQ preprocessing per row, overall it will cost $O(Np)$ time (using $O(N)$ time preprocessing algorithm). We need two RMQ queries per match which amounts to $O(\mathcal{R})$ (using constant time RMQ query) time. Note that, in the worst case $\mathcal{R} = O(Np)$. Finally, it is easy to see that, the set $\mathcal{M}$ in the prescribed order can be computed easily in $O(Np)$ time. Therefore, LCS_RLE-I solves our problem in $O(Np)$ time.

3. LCS_RLE-II

In this section, we use the ideas of LCS_RLE-I to present our second algorithm, LCS_RLE-II, which runs in $O(\mathcal{R} \log\log p + N)$ time. To achieve this running time, we will use an elegant data structure (referred to as the vEB tree henceforth) invented by van Emde Boas [13] that allows us to maintain a sorted list of integers in the range $[1..n]$ in $O(\log\log n)$ time per insertion and deletion. In addition to that it can return $next(i)$ (successor element of i in the list) and $prev(i)$ (predecessor element of i in the list) in constant time.

We follow the same terminology and assume the same settings of Section 2 to describe LCS_RLE-II. So, assume that we are considering the match $(i,j) \in \mathcal{M}_i$ and recall that when the computation of the match (i,j) would be complete, i.e. $\mathcal{T}[i,j]$ is completely computed, we would have the result of $LCS\_RLE(\widetilde{Y}'a^p, X')$. Note carefully that the *baseOperation* is basically the operation required to compute a normal LCS. We can use the LCS algorithm of [10] or [11] just to do the *baseOperation* for each match. Then, per match we would only need $O(\log\log p)$ time [10, 11], requiring a total of $O(\mathcal{R} \log\log p)$ time to perform all the *baseOperations*.

Now, we focus on the *weightOperations*. Our goal is to completely avoid any RMQ preprocessing. We need to modify the *weightOperation* as follows. We will use the vEB tree for this purpose. Recall that we want to find the maximum value of $\mathcal{K}$ in the *weighted query window*. Furthermore, note that, only the matched positions of $\mathcal{K}$ in the *weighted query window* are important in the calculation. So instead of maintaining the array $\mathcal{K}$, we maintain a vEB tree where always the appropriate number (q in this case) of matches (with values after the addition of the relative weights) are kept. And as the computation moves from one match to the next, to maintain the appropriate *weighted query window*, only one element (corresponding to a match) is added to the vEB tree and at most one element is deleted. Also note that we need not delete any element if the current *weighted query window* has fewer matches than q.

When we need the maximum value of the *weighted query window*, we just find the maximum from the vEB tree which can also be found in $O(\log\log p)$ time (by inserting a fictitious element having infinite value and then deleting it after computing its predecessor). As we need to insert and delete constant number of elements from the vEB tree for each match, this can be done in $O(\mathcal{R} \log\log p)$ time on the whole. Like before, we would need to deduct the appropriate value $((|\mathcal{M}_i| + 1 - number((i,j))$ in this case) from the returned maximum to do the proper adjustment.

Finally, the computation of the set $\mathcal{M}$ in the prescribed order can be done following the preprocessing algorithm of [10, 11] which runs in $O(\mathcal{R} \log\log p + N)$ time. So, we have an algorithm solving our problem in $O(\mathcal{R} \log\log p + N)$ time.

4. CONCLUSIONS

In this paper, we have studied the longest common subsequence problem for two strings, where one of the input strings is run length encoded. We have presented two novel algorithms, namely LCS_RLE-I and LCS_RLE-II to solve the problem. We have first presented LCS_RLE-I combining some new ideas with the techniques used in [10, 11]. LCS_RLE-I runs in $O(nP)$ time, which matches the best algorithm in the literature. Then we present LCS_RLE-II which runs in $O(\mathcal{R} \log\log p + N)$ time. Observe that in the worst case, $\mathcal{R} = O(nP)$ and hence the worst case running time of LCS_RLE-II is slightly worse than the best algorithm in the literature. However, in many cases $\mathcal{R} = O(nP)$, and our algorithm would show superior behavior in these cases. In particular, if $\mathcal{R} < pN/\log\log p$, LCS_RLE-II will outperform the best algorithm in the literature.

Additionally, if we run Mitchell's algorithm (the best algorithm in the literature for two RLE strings) in our setting, the running time would be $O((\mathcal{R} + p + n)\log(\mathcal{R} + p + n))$, which clearly is worse than ours. Also, employing some of the insights of Mitchel [7], we believe, our work can be extended to the version where both the input are RLE strings. Another research direction could be to implement the algorithms in the literature along the new ones proposed here and to compare them against each other from a practical point of view.

5. REFERENCES

[1] H.Y. Ann, C.B. Yang, C.T. Tseng, and C.Y. Hor. A fast and simple algorithm for computing the longest common subsequence of run-length encoded strings. *Information Processing Letters*, 108(6):360–364, 2008.

[2] A. Apostolico, G.M. Landau, and S. Skiena. Matching for run-length encoded strings. *J. Complexity*, 15(1):4–16, 1999.

[3] M.A. Bender and M. Farach-Colton. The lca problem revisited. In *LATIN*, pages 88–94, 2000.

[4] V. Freschi and A. Bogliolo. Longest common subsequence between run-length-encoded strings: a new algorithm with improved parallelism. *Information Processing Letters*, 90(4):167–173, 2004.

[5] H. Gabow, J. Bentley, and R. Tarjan. Scaling and related techniques for geometry problems. In *STOC*, pages 135–143, 1984.

[6] Y.W. Jia Jie Liu and R.C.T. Lee. Finding a longest common subsequence between a run-length-encoded string and an uncompressed string. *J. Complexity*, 24(2):173–184, 2008.

[7] J. Mitchell. A geometric shortest path problem, with application to computing a longest common subsequence in run-length encoded strings. *Technical Report Department of Applied Mathematics, SUNY Stony Brook*, 1997.

[8] M.S. Rahman and C.S. Iliopoulos. Algorithms for computing variants of the longest common subsequence problem. In *ISAAC*, pages 399–408, 2006.

[9] M.S. Rahman and C.S. Iliopoulos. Algorithms for computing variants of the longest common subsequence problem. *Theoritical Computer Science*, 395(2-3):255–267, 2008.

[10] M.S. Rahman and C.S. Iliopoulos. New efficient algorithms for the lcs and constrained lcs problems. *Information Processing Letters*, 106(1):13–18, 2008.

[11] M.S. Rahman and C.S. Iliopoulos. A new efficient algorithm for computing the longest common subsequence. *Theoritical Computer Science*, 45(2):355–371, 2009.

[12] E.E.K. Sayoood. Introduction to data compression. *Morgan Kaufmann Publishers Inc*, 2000.

[13] P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6:80–82, 1977.

Prefix transpositions on binary and ternary strings*

Amit Kumar Dutta
Department of CSE, BUET
Dhaka-1000, Bangladesh
amit@csebuet.org

Masud Hasan
Department of CSE, BUET
Dhaka-1000, Bangladesh
masudhasan-
@cse.buet.ac.bd

M. Sohel Rahman
Department of CSE, BUET
Dhaka-1000, Bangladesh
msrahman-
@cse.buet.ac.bd

ABSTRACT

The problem sorting by Prefix Transpositions asks for the minimum number of prefix transpositions required to sort the elements of a given permutation. In this paper, we study a variant of this problem where the prefix transpositions act not on permutations but on strings over a fixed size alphabet. Here, we determine the minimum number of prefix transpositions required to sort the binary and ternary strings, with polynomial time algorithms for these sorting problems.

1. INTRODUCTION

The *transposition distance* between two permutations (and the related problem of *sorting by transposition*) is used to estimate the number of global mutations between genomes and can be used by molecular biologists to infer evolutionary and functional relationships. A transposition involves swapping two adjacent substrings of the permutation. In a prefix transposition, one of them must be a prefix. *Sorting by prefix transposition* is the problem of finding the minimum number of prefix transpositions needed to transform a given permutation into the identity permutation. In the literature, other interesting problems include sorting by other operations like reversals, prefix reversals, block interchange etc.

A natural variant of the aforementioned sorting problems is to consider them not on permutations but on strings over fixed size alphabets. This shift is inspired by the biological observation that multiple "copies" of the same gene can appear at various places along the genome [4, 5]. Indeed, recent works by Christie and Irving [2], Radcliffe et al. [6] and Hurkens et al. [4, 5] explore the consequences of switching from permutations to strings. Notably, such rearrangement operations on the strings have been found to be interesting and important in the study of orthologous gene assignment [1], especially if the strings have only low level of symbol repetition.

The work of Chen et al. [1], presented for both reversals and transpositions, polynomial-time algorithms for computing the minimum number of operations to sort a given binary string. They also gave exact constructive diameter results on binary strings. Radcliff et al. [6] on the other hand gave refined and generalized reversal diameter results for non fixed size alphabets. Additionally, they gave a polynomial-time algorithm for optimally sorting a ternary (3 letter alphabet) string with reversals. Finally, Hurkens et al. [4, 5] introduced *grouping* (a weaker form of sorting), where identical symbols need only be grouped together, while a group can be in any order. In the sequel, they gave a complete characterization of the minimum number of prefix reversals required to group (and sort) binary and ternary strings.

In this paper, we follow up the work of [4, 5] and consider prefix transposition to group and sort binary and ternary strings. Notably, as a future work in [4, 5], the authors raised the issue of considering other genome arrangement operators. In particular, here, we find the minimum number of prefix transpositions required to group and sort binary or ternary strings. It may be noted that, apart from being an useful aid for sorting, grouping itself is a problem of interest in its own right [3].

The rest of the paper is organized as follows. In Section 2, we discuss the preliminary concepts and discuss some notations we use. Section 3 is devoted to grouping, where we present and prove the corresponding bounds. Then, in Section 4, we extend the results of grouping to get the corresponding bounds for sorting. Finally, we briefly conclude in Section 5.

2. PRELIMINARIES

We follow the notations and definitions used in [4, 5], which are briefly reviewed below for the sake of completeness. We use $[k]$ to denote the first k non-negative integers $\{0, 1, \ldots, k-1\}$. A k-ary string is a string over the alphabet $\Sigma = [k]$. Moreover, a string $s = s_1 s_2 \ldots s_n$ of length n is said to be *fully k-ary*, or to have *arity k*, if the set of symbols occurring in it is $[k]$.

A prefix transposition $f(1, x, y)$ on string s of length n, where $1 < x < y \leq (n+1)$, is an rearrangement event that transforms s into $[s[x] \ldots s[y-1]s[1] \ldots s[x-1]s[y] \ldots s[n]]$.

*This research work was conducted as part of the undergraduate thesis of Dutta under the supervision of Hasan and Rahman.

The prefix transposition distance $d_s(s)$ of a string s is defined as the number of prefix transpositions required to sort the string. Note that, after a transposition operation is performed, the two adjacent symbols of the corresponding string may be identical. We consider two strings to be equivalent if one can be transformed into the other by repeatedly duplicating (by transposing) symbols and eliminating adjacent identical symbols. This elimination of adjacent identical symbols gives us a *reduced string*, i.e., a string of reduced length and this process is referred to as *reduction*. As representatives of the *equivalence classes* we take the shortest string of each class. Clearly, these are the strings where adjacent symbols always differ. The process of transforming a string into the representative string of its equivalence class is sometimes referred to as *normalization*. Therefore, the process of normalization basically comprises of repeated transposition and reduction.

For example, let $s = bababab$ and we want to apply operation $f(1,3,6)$. Now, $s[x]\dots s[y-1] = s[3]\dots s[5] = bab$, $s[1]\dots s[x-1] = s[1]\dots s[2] = ba$, $s[y]\dots s[n] = s[6]\dots s[7] = ab$. Therefore, after applying the operation, we get, $s = s[3]\dots s[5]s[1]\dots s[2]s[6]\dots s[7] = \overline{bab}\underline{ba}ab = ba$ **bbaa** $b =$ ba **ba** b = babab

A reduction that decreases the string length by x is called an x transposition. So, if $x = 0$, then we have a 0 transposition. The above example illustrates a 2 transposition.

3. GROUPING

The task of sorting a string can be divided into two subproblems, namely, *grouping* the identical symbols together and then putting the groups of identical symbols in the right order. The grouping distance $d_g(s)$ of a fully k ary string s is defined as the minimum number of prefix transposition required to reduce the string to one of length k.

3.1 Grouping Binary Strings

As strings are normalized, only 2 kinds of binary strings are possible, namely, 010101...010 and 101010...101. The grouping of binary strings seems to be quite easy and obvious. The following bound is easily achieved.

THEOREM 1. (Bound for Binary strings) *Let s be a fully binary string. Then, $d_g(s) = \lceil \frac{n-k}{2} \rceil$.*

PROOF. We can always have a 2 transposition if $|s|$ is even. However, if $|s|$ is odd, we need an extra 1 transposition. So, the upper bound is $d_g(s)) = \lceil \frac{n-k}{2} \rceil$. □

We illustrate the above result with the help of an example. Let $s = ababab$. Then we can continue as follows: $s = ababab = \overline{ab}\underline{a}bab = $ **aabb**$ab = abab = \overline{ab}\underline{ab} =$ **aabb** $= ab$. Here, we need two 2 transpositions to group this string. So, $d_g(s) = 2$.

3.2 Grouping Ternary Strings

In this section, we focus on ternary strings. As it seems, grouping ternary strings is not as easy as grouping binary strings. We start with the following theorem.

LEMMA 1. *In a fully ternary string, we can always perform a 1 transposition.*

PROOF. We take a ternary string s of length $n > 3$. Now, we take a prefix a of length k. Now if $a[1]$ occurs at the suffix at position i, we can transpose $a[1]\dots a[i-1]$ with $a[i]$. Then, a[1] and a[i] are adjacent and we can eliminate one of the two. Otherwise, if $a[k]$ occurs at the suffix at position i, then we can transpose $a[1]\dots a[k]$ with $a[k+1]\dots a[i-1]$. Then $a[k]$ and $a[i]$ are adjacent and as before, we can eliminate one of them. Since, one of the above cases always occurs for ternary strings, the result follows. □

3.3 Grouping distance for Ternary strings

The lower bound for the *grouping* of a ternary string remains the same as that of binary strings; but, as can be seen from Theorem 2 below, the upper bound differs. We first give an easy but useful lemma.

LEMMA 2. *Suppose $s[1..n]$ is a fully ternary string. If we have a prefix $s[1..i], 1 < i \leq n-2$ such that $s[1] = s[n-1]$ and $s[i] = s[n]$, then we have a 2 transposition.* □

The proof of Lemma 2 is very easy and hence is omitted.

THEOREM 2. (Bound for Ternary strings) *Let s be a fully ternary string. Then, $\lceil \frac{n-k}{2} \rceil \leq d_g(s) \leq \lceil \frac{n-k}{2} \rceil + 1$ where n is the length of the string and k is the arity.*

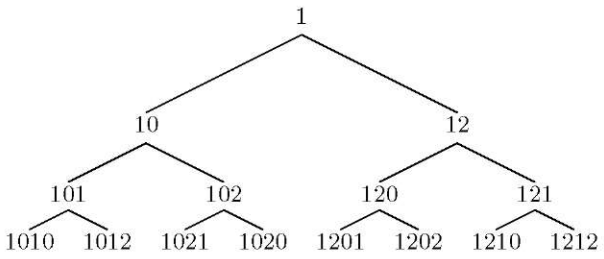
PROOF. First we prove the lower bound. Here, $k = 3$. If we can always give a 2 transposition, then at each operation the string length is decremented by 2 and if $|s|$ is odd, we need an extra 1 transposition. Hence, we have $\lceil \frac{n-k}{2} \rceil \leq d_g(s)$.

Let us concentrate on the upper bound. As strings are fully ternary, we don't need to work with $n \leq 3$. Now, if we apply the upper bound for $n = 4, 5$ and 6, we have $d_g(s) = 2, 2$ and 3 respectively. It is easy to realize that, by Lemma 1, we can always satisfy the above upper bound. Thus the upper bound is proved for $n < 7$.

Now we consider $n \geq 7$. In what follows, we only consider strings starting with 1. This doesn't lose the generality since we can always use relabeling for strings starting with 0 or 2. Now, note carefully that for any string starting with 1, we can only have one of the following eight prefixes of length 4:

$$1012, 1010, 1021, 1020, 1201, 1202, 1210 \text{ and } 1212. \quad (1)$$

Here we give the tree diagram of all strings starting with 1:



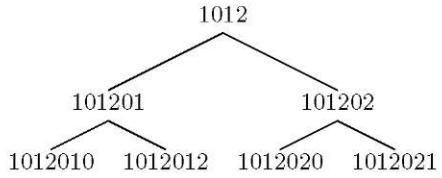
Now note that, the upper bound of Theorem 2 tells us we can give at most three 1 transpositions when n is even (i.e. $n-k$ is odd) and two 1 transpositions when n is odd (i.e. $n-k$ is even). Note that, if we could give a 2 transposition at each step, we would get the bound of $\lceil \frac{n-k}{2} \rceil$. For a n length string, if we can give a 2 transposition, the resulted reduced string may start with 1, 0 or 2. For the latter two cases, we can use relabeling as mentioned before. Therefore we can safely state that the reduced string will have any of the 8 prefixes of List 1. Hence, it suffices to prove the bound considering each of the prefixes of List 1. We will now follow the following strategy:

> We will take each of the prefixes of List 1 and expand it (by adding symbols) to construct all possible strings of length greater than or equal to 7. Strictly speaking, we will not consider all possible strings; rather we will continue to expand until we get a 2 transposition, since afterwards, any further expansion would also guaranty a 2-transposition. Suppose s is one such string. We will take s and try to give a 2 transposition with any of its prefix. If we succeed, then, clearly, we are moving towards the best case and we only need to work with the reduced string. If we can't give a 2 transposition, we specifically deal with s and show that the bound holds. Now if we can give a 2 transposition, the reduced string will have any of the 8 prefixes (using relabeling if needed) and we will show that all strings of these cases will follow the bound.

Firstly it is easy to note that, the prefixes 1010 and 1212 themselves have 2-transpositions (Lemma 2). Therefore, we can safely exclude them from the following discussion. In what follows, when we refer to the prefixes of List 1, we would actually mean all the prefixes excluding 1010 and 1212. Now, to expand the prefixes, if we add 10 or 12, all of them would be able to give a 2 transposition (Lemma 2). Therefore, in what follows, we consider the other cases. Now we analyze each of the prefixes below.

1012

We first give the tree diagram of all string having prefix 1012



If we add 01, we can only add (By 'add' we mean append) 0 or 2. The resulting expanded string becomes 1012010 or 1012012. In both cases, we can give a 2 transposition. On the other hand, if we add 02, we can add 0 or 1 next and the string becomes 1012020 or 1012021. The string 1012020 satisfies the bound as follows: $\overline{1}\underline{01}2020 \Rightarrow 012020 \Rightarrow \overline{012}\underline{0}20 \Rightarrow 0120 \Rightarrow \overline{0}\underline{12}0 \Rightarrow 120$. Here, $n = 7$ and we need only 3 transpositions holding the bound true. Now if we add 1, the string becomes 10120201. For this one as well the bound holds as follows: $\overline{1}\underline{01}20201 \Rightarrow 0120201 \Rightarrow \overline{01}\underline{20}201 \Rightarrow 20201 \Rightarrow \overline{20}\underline{2}01 \Rightarrow 201$. Here, $n = 8$ and we needed 3 transpositions. Now, it can be easily checked that strings like $1012020(20)^*$ or $1012020(20)^*1$ the bound holds using the same strategy as shown above. Adding anything with $1012020(20)^*1$ will also give a 2 transposition as follows. Clearly, we first need to add either 0 or 2 and immediately Lemma 2 would apply.

Now we consider 1012021. The bound holds for this one as well as follows: $\overline{1}\underline{01}2021 \Rightarrow 012020 \Rightarrow \overline{012}\underline{0}21 \Rightarrow 0121 \Rightarrow \overline{01}\underline{2}1 \Rightarrow 201$. Now, strings like $1012(02)^*1$ can also be handled similarly hence the bound holds for them as well. Adding anything with $1012(02)^*1$ will also give a 2 transposition (Lemma 2).

1021

This prefix is ending with 1 and adding anything will give a 2 transposition (Lemma 2).

1020

We first add 21 with this prefix. Then, adding anything with 102021 will give a 2 transposition (Lemma 2). Now, for $102(02)^+1$, we need a 1 transposition to move the initial 1 at the end. Now the upper bound holds because the rest of the string is binary (Theorem 1). Also, adding anything with $102(02)^+1$ will give a 2 transposition (Lemma 2).

Now, if we add 20 with the prefix we get 102020. Next we add 1 or 2 and get 1020201 or 1020202 respectively. For 1020201, the bound holds as follows: $\overline{1}\underline{020201} \Rightarrow 020201 \Rightarrow \overline{02}\underline{0}201 \Rightarrow 0201 \Rightarrow \overline{0}\underline{2}01 \Rightarrow 201$. All strings like $1020(20)^+1$ can also be handled similarly and hence the bound holds for them as well. And adding anything with $1020(20)^+1$ will give a 2 transposition (Lemma 2).

Now for 1020202, the bound holds as follows: $\overline{10}\underline{2}0202 \Rightarrow 210202 \Rightarrow \overline{210}\underline{2}02 \Rightarrow 2102 \Rightarrow \overline{2}\underline{10}2 \Rightarrow 102$. Now, string like $10202(02)^+$ can be handled similarly and hence the bound holds. For $10202(02)^+1$, we need a 1 transposition to move the initial 1 at the end. Now the upper bound holds because the rest of the string is binary (Theorem 1). Also adding anything with $10202(02)^+1$ will give a 2 transposition (Lemma 2).

1201

This prefix is ending with a 1 and adding anything will give a 2 transposition (Lemma 2).

1202

Here, we can employ relabeling and map 2 to 0 and 0 to 2 to get 1020. Now recall that we have already considered 1020 before and hence we are done.

1210

Here we can again employ relabeling and map 2 to 0 and 0 to 2 to get 1012. And since we have already considered 1012, we are done.

And this completes our proof. □

4. SORTING

The sorting distance $d_s(s)$ of a fully k ary string s is defined as the minimum number of prefix transposition required to sort the string to one of length k. We again consider normalized strings.

4.1 Sorting Binary Strings

THEOREM 3. (Bound for Binary strings) *Let s be a fully binary string. Then, $d_s(s) \leq \lceil \frac{n-k}{2} \rceil$.*

PROOF. As binary strings have only 2 letters, after grouping they are already sorted (in ascending or descending order). So, the upper bound is $d_s(s) \leq \lceil \frac{n-k}{2} \rceil$. □

4.2 Sorting Ternary Strings

THEOREM 4. (Bound for Ternary strings) *Let s be a fully Ternary string. Then, upper bound for sorting ternary string is $d_g(s) \leq \lceil \frac{n-k}{2} + 2 \rceil$.*

PROOF. After grouping a ternary string, we have the following grouped strings: 012, 021, 102, 120, 201 and 210. Among these, 012 and 210 are already sorted. We need one more 0 transposition to sort 021, 102, 120 and 201. Hence the result follows. □

5. CONCLUSION

In this paper, we have discussed grouping and sorting of fully binary and ternary strings when the allowed operation is prefix transposition. Following the work of [4, 5], we have handled grouping by prefix transpositions of binary and ternary strings first and extended the results for sorting. In particular we have proved that, for binary strings the grouping distance $d_g(s) = \lceil \frac{n-k}{2} \rceil$ and for ternary string we have $\lceil \frac{n-k}{2} \rceil \leq d_g(s) \leq \lceil \frac{n-k}{2} \rceil + 1$, where n is the length of the string and k is the arity. On the other hand, for sorting binary and ternary strings the sorting distance $d_s(s)$ is upper bounded by $\lceil \frac{n-k}{2} \rceil$ and $\lceil \frac{n-k}{2} \rceil + 2$ respectively. As has already been mentioned, we are now considering the higher-arity alphabets as an extension of our work.

6. REFERENCES

[1] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 2(4):302–315, 2005.

[2] D.A. Christie and R.W. Irving. Sorting strings by reversals and by transpositions. *SIAM J. Discrete Math.*, 14(2):193–206, 2001.

[3] H. Eriksson, K. Eriksson, J. Karlander, L.J. Svensson, and J. Wästlund. Sorting a bridge hand. *Discrete Mathematics*, 241(1-3):289–300, 2001.

[4] C.A.J. Hurkens, L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, and J. Tromp. Prefix reversals on binary and ternary strings. *SIAM J. Discrete Math.*, 21(3):592–611, 2007.

[5] C.A.J. Hurkens, L. van Iersel, J. Keijsper, S. Kelk, L. Stougie, and J. Tromp. Prefix reversals on binary and ternary strings. In *AB*, pages 292–306, 2007.

[6] A.J. Radcliffe, A.D. Scott and E.L. Wilmer Reversals and transpositions over finite alphabets. *SIAM J. Discrete Math.*, 2006.

Embedding of complete and nearly complete binary trees into hypercubes

Aleksander Vesel
University of Maribor
Faculty of Natural Sciences and Mathematics
Koroška c. 160, Maribor, Slovenia
vesel@uni-mb.si

ABSTRACT
A new simple algorithm for optimal embedding of complete binary trees into hypercubes as well as a node-by-node algorithm for embedding of nearly complete binary trees into hypercubes are presented.

Categories and Subject Descriptors
G.2.2 [**Discrete mathematics**]: Algorithms

Keywords
binary trees, algorithms, embedding

1. INTRODUCTION
Hypercubes and binary trees are omnipresent in computer science. In particular, hypercubes are very popular models for parallel computation because of their regularity, recursive structure and the ease of routing. On the other hand, binary trees can represent the basic computational structure of divide-and-conquer or branch-and-bound algorithms. In many cases however, it is more suitable that the internal structure of an algorithm is modeled by a more general structure - a nearly complete binary tree.

In this paper we consider the problem of embedding a (nearly) complete binary tree in a hypercube. This problem occurs during the implementation of divide-and-conquer algorithms in a hypercube network, e.g., see [3, 5]. An embedding is a mapping from the guest graph, representing the communication structure of the processes, into the the host graph, representing the communication network of the processors. Therefore, the problem of allocating processes to processors in a multiprocessor system is also known as the *mapping problem.*

A *tree* is a connected acyclic graph. One vertex is distinguished and called the *root.* A vertex of degree one is called a *leaf* of the tree if it is not the root. The *level* of a vertex v in a tree is the number of vertices on the simple path from the root to v. Note that the level of the root is one. The height of a tree T is the largest level of a vertex in T. A vertex u is called a *child* of v if u is adjacent to v and the level of u is greater than the level of v. If u is a child of v, then v is called the *parent* of u.

A *full binary tree* is a tree in which every node other than the leaves has two children. A full binary tree is ordered, i.e. we distinguish between left and right children. A *complete binary tree* is a full binary tree in which all leaves are at the same level. A *nearly complete binary tree* of height h is composed of a complete binary tree of height $h-1$ and with some nodes at level h (not necessary positioned to the left).

The *hypercube* of order d and denoted Q_d is the graph G where the vertex set $V(G)$ is the set of all binary strings $u_1u_2\dots u_d$, $u_i \in \{0,1\}$. Two vertices $x, y \in V(G)$ are adjacent in Q_d if and only if x and y differ in precisely one place.

An *embedding* of a graph G into a graph H is an injection $f : V(G) \to V(H)$ such that if (u, v) is an edge in $E(G)$ then $(f(u), f(v))$ is an edge in $E(H)$.

For binary vectors $s, t \in \{0,1\}^n$ let $s \oplus t$ denote the coordinate-wise addition modulo two, e.g. $100011 \oplus 000001 = 100010$. Let e_i^n be the binary vector $u_1, u_2, \dots, u_n$ with $u_i = 1$ and $u_j = 0$, $j \neq i$. If s is a binary vector of length n, then we will call the operation $e_i^n \oplus s$ a *reflection.* We will also use the "+" symbol as the concatenation operator, i.e. $s+t$ joins two binary vectors s and t end to end. If we concatenate a binary vector with a single bit (0 or 1), than we call this operation a *projection.* For a binary vector s of length n, $s+0$ and $s+1$ are projections of s into two disjoint subcubes of order n of Q_{n+1}.

The minimum h required for an embedding of a graph G into Q_h is called the *cubical dimension* of G. Deciding whether there exists an embedding of a given tree into a hypercube of a given dimension is known to be NP-complete [4]. Moreover, even in case of trees with bounded degrees, their cubical dimensions are unknown in most cases.

Obviously, if G is a graph such that $2^h \geq |V(G)| > 2^{h-1}$, then the cubical dimension of G is at least h. However, it is well known that the complete binary tree on $2^h - 1$ vertices cannot be embedded into Q_h for $h \geq 3$.

Havel in [1] conjectured that every binary tree T such that $2^h \geq |V(T)| > 2^{h-1}$ has an embedding into Q_{h+1}. The conjecture is still open, but there are many partial results supporting this assertion. It has been shown, for example, that a complete binary tree of height h can be embedded into the hypercube of order $h+1$, e.g. [3, 5].

2. COMPLETE BINARY TREE

Let C_h denote the complete binary tree of height h. Let also r_h denote the root of C_h and let C_h^l and C_h^r denote the left and the right subtree of C_h, respectively. Obviously, C_h^l and C_h^r are both complete binary trees of height $h-1$.

Let $\sigma_l : V(C_h^l) \to V(C_{h-1})$ be the mapping that in the natural way maps each vertex $v \in V(C_h^l)$ to the corresponding vertex of C_{h-1}, e.g. if v is the root of C_h^l, then $\sigma_l(v) = r_{h-1}$. Analogously we define the mapping $\sigma_r : V(C_h^r) \to V(C_{h-1})$.

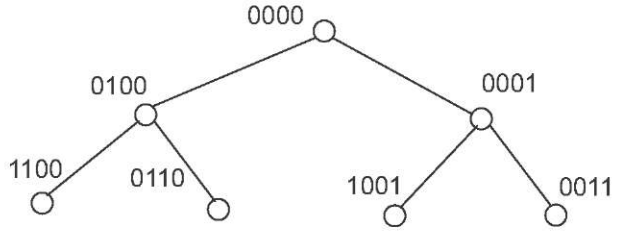


Figure 1: β_3 - the mapping of C_3

For each $h \geq 3$, we define a mapping $\beta_h : V(C_h) \to \{0,1\}^{h+1}$ as follows. The mapping $\beta_3 : V(C_3) \to \{0,1\}^4$ is depicted in Fig. 1, while for $h > 3$ the mapping is given by

$$\beta_h(v) = \begin{cases} 0^{h+1}, & v = r_h \\ \beta_{h-1}(\sigma_r(v)) + 1, & v \in V(C_h^r) \\ \beta_{h-1}(\sigma_l(v)) \oplus 10^{h-1}10, & v \in V(C_h^l), h \text{ even} \\ \beta_{h-1}(\sigma_l(v)) \oplus 0^{h-3}100+0, & v \in V(C_h^l), h \text{ odd}. \end{cases}$$

For any v of C_h, the string $\beta_h(v)$ will be also called a *code word* defined by β_h in the sequel.

THEOREM 1. *Let $h \geq 3$. Then β_h defines an embedding of the complete binary tree of height h in the hypercube of order $h+1$.*

The basis of the proof is the following lemma.

LEMMA 1. *Let $h \geq 3$. Then $\beta_h(v) = 0^{h+1}$ if and only if v is the root of C_h.*

PROOF. If v is the root of C_h, then $\beta_h(v) = 0^{h+1}$ by definition. On the other hand, the recursive definition of β_h implies, that if v is not the root of C_h, then $\beta_h(v)$ is obtained as a sequence of projections and reflections either from the root of the C_{i+1}, i.e. $\beta_{i+1}(r_{i+1}) = 00^i0$, $h > i \geq 3$, or from a code word defined by β_3.

Suppose first that $\beta_h(v)$ derives from 00^i0. Suppose also that i is even. Some of the code words derived from 00^i0 can be seen in the tree depicted in Fig. 2. The root of the tree is 00^i0, while the left and the right child of 00^i0 are obtained as code words derived from it in the left and the right subtree of C_{i+2}, respectively. Analogously, the left and the right children of 10^i000 and 00^i001 are code words in C_{i+3}, etc.

Note that the definition of β_j for $j \geq 4$ implies that $\beta_j(v)$ is derived from a code word s of C_{j-1} such that either the first or the $(j-3)$-th bit of s is reversed. It follows that from a code word s of C_{j-1} with a 1 in at least one of the positions: $2, 3, \ldots, j-4$, a code word of the form 0^{h+1} cannot be derived. Since every leaf of the tree in Fig. 2 in at least one of those positions possesses entry 1, it follows that 0^{h+1} cannot derive from 00^i0 if i is even. It is not difficult to see that a similar tree (having leaves of length j with a 1 in at least one of the positions $2, 3, \ldots, j-4$) can be derived for 00^i0, where i is odd.

For a $\beta_h(v)$ derived from a code word defined by β_3 observe first that all code words of the left subtree of C_3 depicted in Fig. 1 posses 1 at the second place, which implies that 0^{h+1} cannot be derived from any of them. For $\beta_h(v)$ derived from a code word of the right subtree of β_3, observe for example the tree depicted for code word 0001 the in Fig. 3. From the same arguments as above we can conclude that 0^{h+1} cannot derive from any code word defined by β_3 and the proof is complete. □

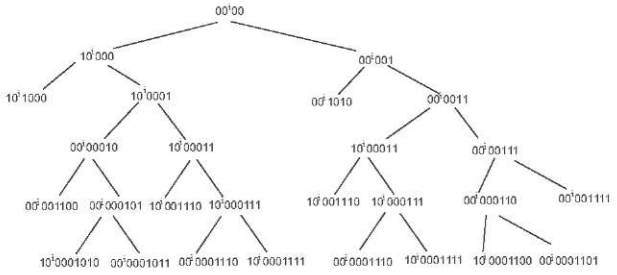


Figure 2: Code words derived from 00^i0 when i is even

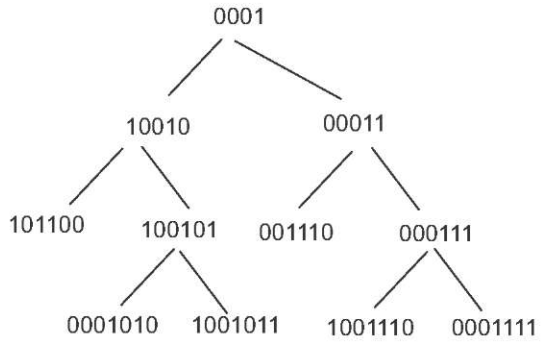


Figure 3: Some code words derived from 0001

PROOF PROOF (OF THEOREM 1). The proof is by induction on h. The claim obviously holds for β_3. Suppose also that the claim holds for h. Note first that projections maps the vertices of C_{h+1}^l and C_{h+1}^r into two disjoint hypercubes.

Moreover, reflections of an embedding in a hypercube preserve Hamming distance. Therefore β_{h+1} is an embedding of C^l_{h+1} and C^r_{h+1} in the hypercube of order $h+1$. Finally, since the root of C_{h+1} and the root of C^l_{h+1} (as well as C^r_{h+1}) differ in precisely one bit, we conclude that β_{h+1} embeds C_{h+1} into Q_{h+1} and the proof is complete. □

Theorem 1 is the basis for the algorithm to compute an optimal embedding of the complete binary tree of height h in Q_{h+1}.

We first present the algorithm to calculate an embedding of the complete binary tree of height h from an embedding of the complete binary tree of height $h-1$. It is assumed in the algorithm, that if v is an arbitrary node of a complete binary tree T, then $b(v)$ is the code word of v and $p(v)$ a parent of v in T. Furthermore, r denotes the root of T and T_l and T_r denote the left and the right subtree of T, respectively.

Procedure NEW TREE
input: h, T, b { $h > 3$, b an embedding of $T = C_{h-1}$ into Q_{h+1} }
output: b {An embedding of $T = C_h$ into Q_{h+1}}
  begin
  traverse T_r from level h to level 2
    for every $v \in T_r$ **do** $b(v) := b(p(v))) + 1$;
  if h mod $2 = 0$ **then**
    traverse T_l from level h to level 2
      for every $v \in T_l$ **do**
        $b(v) := b(p(v)) \oplus 10^{h-1} + 0$;
  else
    traverse T_l from level h to level 2
      for every $v \in T_l$ **do**
        $b(v) := b(p(v))) \oplus 0^{h-3}100 + 0$;
  $r := 0^{h+1}$; { The new root of T }
 end.

We next describe the algorithm to compute an optimal embedding of the complete binary tree of height h in a hypercube.

Procedure CODES
input: h {height of a tree, $h \geq 3$}
output: T, b {T is of height h with an embedding b}
  begin
  $T := C_3$.
  Set $b(v)$ for every $v \in T$ as in Fig. 1;
  for $i := 4$ to h **do begin**
    Augment T with new level of nodes to
    obtain C_i;
    NEW TREE(i, T, b);
 end.

THEOREM 2. *For any $h \geq 3$, CODES embeds C_h into Q_{h+1} in linear time and space.*

PROOF. The correctness of the algorithm is proved by induction on h. If $h = 3$, then the embedding is given in Step 2, the correctness of which can be verified by Fig. 1.

Assume now that for $i = h-1$ the algorithm correctly compute an embedding of T. In other words, when NEW TREE is called in Step 3 for $i = h$, the vector b corresponds to β_{h-1}. Moreover, for a node $v \in T_l$ ($v \in T_r$), the old value of $b(p(v)))$ corresponds to $\beta_{h-1}(\sigma_r(v))$ ($\beta_{h-1}(\sigma_l(v))$). Therefore, since the nodes of T are traversed from the last level to the roots of the subtrees and since NEW TREE accurately follows the definition of β_h, we can conclude that the embedding is correct.

T, p, and b can obviously be represented in linear space, therefore we only consider the time complexity. Let $n := 2^h - 1$ denote the number of nodes of a complete binary tree of height h. Note first that NEW TREE computes an embedding b in time which is linear in the size of T. Since the number of vertices of T in i-th iteration of the **for** loop equals $2^i - 1$, the total number of steps of the algorithm is given by

$$\sum_{i=4}^{h} 2^i - 1 = 2^{h+1} - 12 = O(n) .$$

This argument completes the proof. □

3. NEARLY COMPLETE BINARY TREE

In this section we present a simple node-by-node algorithm for constructing an embedding of a nearly complete binary tree into a hypercube. We assume that in each time step a nearly complete binary tree can grow by one node inserted at the last level of a tree. Note, that in [2] a somewhat similar approach has been studied, where the complete binary tree grows by a complete level of its leaves.

The algorithm presented herein computes the map of the nodes of a new tree using the map of their parent node. Moreover, if a new node does not change the height of a tree, the old nodes need not to be remapped.

The algorithm of the previous section implies that the embedding of C_h can be performed by using an embedding of C_{h-1} in such a way that the embedding of a node v is computed from the "old" embedding of its parent node.

This observation leads to a node-by-node algorithm for embedding of nearly complete binary trees into hypercubes. The algorithm augments a given nearly complete binary tree with one node, which is inserted at the level h and computes the mapping of the augmented tree. We will show that in the majority of cases the algorithm is able to determine the embedding of the augmented tree simply by expanding an embedding with the map of the new node. Moreover, the map of the new node can be computed with ease from the map of its parent node.

If a nearly complete binary tree T of height h with an embedding into Q_{h+1} is augmented with one new node, then for the resulting nearly complete binary tree T' the embed-

ding into Q_{h+1} (or Q_{h+2}, if the height of T' is $h+1$) is computed. The new node can be

(i) a leaf at level h, if T is not complete or

(ii) a leaf at level $h+1$, if T is complete.

In order to obtain an embedding for T' we first show the following lemma.

LEMMA 2. *Let for $h \geq 3$, v be a leaf of C_h and u the parent of v in C_h. Then*

$$\beta_h(v)=\begin{cases}\beta_h(u) \oplus 10^h, & v \text{ is the left child of } u\\ \beta_h(u) \oplus 001^{h-3}0, & v \text{ is the right child of } u\end{cases}$$

PROOF. The proof is by induction with respect to h. The claim obviously holds if $h = 3$ as can be seen in Fig 1. Let us denote v a leaf of C_{h+1} and u the parent of v. If v (and u) is in the right subtree of C_{h+1}, then by inductive hypothesis $\beta_h(\sigma_r(v))$ and $\beta_h(\sigma_r(u))$ differ either in the first bit, if v is the left child of u, or in the third bit, if v is the right child of u. It is straightforward to see now that $\beta_{h+1}(v) = \beta_h(\sigma_r(v)) + 1$ and $\beta_{h+1}(u) = \beta_h(\sigma_r(u)) + 1$ differ either in the first or in the third bit. If v is in the left subtree of C_{h+1}, the proof is analogous. □

In the following algorithm, let for an arbitrary vertex u of a nearly complete binary tree T, $b(u)$ and $p(u)$ denote the code word of u and the parent of u in T, respectively.

Procedure NEW NODE
input: h, T, b, v { b is the embedding of T, $h \geq 3$, v is a new node }
output: T, h, b { An augmented tree of height h with an embedding b }
begin
if T is complete **then begin**
NEW TREE (h, T, b);
$h := h + 1$;
end;
Insert v at the level h in T;
if v is the left child of $p(v)$ **then**
$b(v) := b(p(v))) \oplus 10^h$;
else $b(v) := b(p(v))) \oplus 0010^{h-2}$;
end.

In order to obtain an embedding of a nearly complete binary tree with NEW NODE, before the algorithms is first called, Step 1 and Step 2 of CODES have to be executed. T is then the complete binary tree of height 3 with the embedding b.

THEOREM 3. *If T is a nearly complete binary tree of height $h > 3$ and b an embedding of T into Q_{h+1}, then NEW NODE correctly embeds T' either*

(i) into Q_{h+2}, if T is complete or

(ii) into Q_{h+1}, if T is not complete.

Moreover, if T is C_3 and b an embedding of C_3 into Q_4, then NEW NODE correctly embeds T' into Q_5.

PROOF. Assume that T is either C_3 or an arbitrary nearly complete binary tree of height $h > 3$ and b its embedding into Q_{h+1}. If T is not complete, then the correctness of the algorithm follows from Lemma 2.

Let then T be a complete binary tree. When NEW TREE is called in Step 1, the value of h is not yet incremented, i.e. the output of the procedure is the complete tree of height h with an embedding into Q_{h+2}. However, in Steps 2 and 3, T is first augmented with a new node at level $h+1$ and then an embedding into Q_{h+2} of the resulting nearly complete tree of height $h+1$ is computed. □

4. CONCLUSIONS

The following concluding comment concerning the time complexity of the algorithm is in order. The algorithm remaps the nodes of T only if T is a complete binary tree. In other cases a remapping is not performed. Clearly, the embedding of a new node depends only on the map of its parent node and can be performed in constant time. However, even in the case when remapping is needed, the computation of the new embedding can be done independently in each node v such that only the code word of a parent node of v is used. It follows that the remapping can be computed on the hypercube in parallel in constant time.

5. REFERENCES

[1] I. Havel. On hamiltonian circuits and spanning trees of hypercubes. *Časopis pro Pěstováni Matematiky*, 2(209):135 –152, 1984.

[2] V. Heun and E. W. Mayr. Optimal dynamic embeddings of complete binary trees into hypercubes. *J. Parallel Distrib. Comput.*, 61(8):1110–1125, 2001.

[3] A. S. Wagner. Embedding the complete tree in the hypercube. *J. Parallel Distrib. Comput.*, 20(2):241–247, 1994.

[4] A. S. Wagner and D. G. Corneil. Embedding trees in a hypercube is np-complete. *SIAM J. Comput.*, 19(3):570–590, 1990.

[5] A. Y. Wul. Embedding of tree networks into hypercubes. *J. Parallel Distrib. Comput.*, 2(3):238–249, 1985.

Information set-distance

Joel Ratsaby
Electrical and Electronics Engineering Department
Ariel University Center of Samaria
Ariel 40700, ISRAEL
ratsaby@ariel.ac.il

ABSTRACT

Let $|A|$ denote the cardinality of a finite set A. For any real number x define $t(x) = x$ if $x \geq 1$ and 1 otherwise. For any finite sets A, B let $\delta(A,B) = \log_2\left(t\left(\left|B \cap \overline{A}\right| |A|\right)\right)$. We define a new set distance $d(A,B) = \max\left\{\delta\left(A,B\right), \delta\left(B,A\right)\right\}$ which may be applied to measure the distance between binary strings of different lengths. We prove that d is a semi-metric on the space of sets of size at least 2. The triangle inequality holds for triplets A, B, C that are not strictly contained one in any of the other two.

1. INTRODUCTION

A basic problem in pattern recognition is to find a numerical value that represents the dissimilarity or 'distance' between any two input patterns of the domain. For instance, between two binary sequences that represent document files or between genetic sequences of two living organisms. A good distance is one which picks out only the 'true' dissimilarities and ignores those that arise from irrelevant attributes or due to noise. In most applications defining a meaningful distance requires inside information about the domain, for instance, in the field of information retrieval the distance between two documents is weighted largely by words that appear less frequently since the words which appear more frequently are less informative. Typically, different domains require the design of different distance functions which take such specific prior knowledge into account. It can therefore be an expensive process to acquire expertise in order to formulate a good distance. The pioneering paper of [4] introduced a notion of complexity of finite binary string which does not require any prior knowledge about the domain or context represented by the string (this is sometimes referred to as the *universal* property). This complexity (called the production complexity of a string) is defined as the minimal number of copy-operations needed to produce the string from a starting short-string called the base.

In the current paper we are interested in developing a distance between two binary strings which also possesses this universal property. Our approach is to consider a binary string as a set of substrings. To represent the complexity of such a set we use the notion of combinatorial entropy [3] to introduce a new set-distance function. We start with a distance for general sets and then apply it as a distance between binary strings.

2. THE DISTANCE

In what follows, Ω is a given non-empty set which serves as the domain of interest. The cardinality of any set A is denoted by $|A|$ and the set of all finite subsets of Ω is denoted by $\mathcal{P}_F(\Omega)$. Define $t : \mathbb{R} \to \mathbb{R}$ as follows:

$$t(x) = \begin{cases} x & \text{if} \quad x \geq 1 \\ 1 & \text{otherwise} \end{cases}.$$

Definition 1. For each pair of finite sets $A, B \subset \Omega$ define the following function $\delta : \mathcal{P}_F(\Omega) \times \mathcal{P}_F(\Omega) \to \mathbb{N}_0$ which maps a pair of finite sets into the non-negative integers,

$$\delta(A,B) := \log\left(t\left(\left|B \cap \overline{A}\right| |A|\right)\right)$$

where $\overline{A}$ denotes the complement of the set A and log is with respect to base 2. It is simple to realize that $\delta(A,B)$ equals $\log\left(\left|B \cap \overline{A}\right| |A|\right)$ with the exception when A or B is empty or $B \subseteq A$.

Remark 2. Note that δ is non-symmetric, i.e., $\delta(A,B)$ is not necessarily equal to $\delta(B,A)$. Also, $\delta(A,B) = 0$ when $B \subseteq A$ (not only when $A = B$).

From an information theoretical perspective [2] the value $\log\left|B \cap \overline{A}\right|$ represents the additional description length (in bits) of an element in B given *a priori* knowledge of the set A. Hence we may view A as a partial 'dictionary' while the part of B that is not included in A takes an additional $\log\left|B \cap \overline{A}\right|$ bits of description given A.

The following set will serve as the underlying space on which we will consider our distance function. It is defined as

$$\mathcal{P}_F^+(\Omega) \quad := \quad \mathcal{P}_F(\Omega) \setminus \{A \subset \Omega : |A| \leq 1\}.$$

It is the power set of Ω but without the empty set and singletons. We note that in practice for most domains, as for instance the domain of binary strings considered later, the restriction to sets of size greater than 1 is minor.

The following lemma will be useful in the proof of Theorem 5.

Lemma 3. *The function δ satisfies the triangle inequality on any three elements A, B, $C \in \mathcal{P}_F^+(\Omega)$ none of which is strictly contained in any of the other two.*

PROOF. Suppose A, B, C are any elements of $\mathcal{P}_F^+(\Omega)$ satisfying the given condition. It suffices to show that

$$\delta(A,C) \quad \leq \quad \delta(A,B) + \delta(B,C). \tag{2.1}$$

First we consider the special case where the triplet has an identical pair. If $A = C$ then by Remark 2 it follows that $\delta(A,C) = 0$ which is a trivial lower bound so (2.1) holds. If $A = B$ then $\delta(A,B) = 0$ and both sides of (2.1) are equal hence the inequality holds (similarly for the case of $B = C$).

Next we consider the case where

$$\left|C \cap \overline{A}\right|, \ \left|B \cap \overline{A}\right|, \ \left|C \cap \overline{B}\right| \quad \geq \quad 1. \tag{2.2}$$

By definition of $\mathcal{P}_F^+(\Omega)$ we have $|A| \geq 2$ hence

$$\begin{aligned} \delta(A,C) &= \log\left(t\left(\left|C \cap \overline{A}\right| |A|\right)\right) \\ &= \log\left(\left|C \cap \overline{A}\right| |A|\right) = \log\left|C \cap \overline{A}\right| + \log|A|. \end{aligned}$$

Next, we claim that $C \cap \overline{A} \subseteq \left(B \cap \overline{A}\right) \cup \left(C \cap \overline{B}\right)$. If $x \in C \cap \overline{A}$ then $x \in C$ and $x \in \overline{A}$. Now, either $x \in B$ or $x \in \overline{B}$. If $x \in B$ then because $x \in \overline{A}$ it follows that $x \in B \cap \overline{A}$. If $x \in \overline{B}$ then because $x \in C$ it follows that $x \in C \cap \overline{B}$. This proves the claim. Next, we have

$$\begin{aligned} &\delta(A,B) + \delta(B,C) \\ &\quad = \log|A| + \log\left|B \cap \overline{A}\right| + \log|B| + \log\left|C \cap \overline{B}\right|. \end{aligned}$$

It suffices to show that

$$\log\left|C \cap \overline{A}\right| \leq \log\left|B \cap \overline{A}\right| + \log\left|C \cap \overline{B}\right| + \log|B|. \tag{2.3}$$

We claim that if three non-empty sets X, Y, Z satisfy $X \subseteq Y \cup Z$ then $\log|X| \leq \log(2|Y||Z|)$. To prove this, it suffices to show that $|X| \leq 2|Y||Z|$. That this is true follows from $|X| \leq |Y \cup Z| \leq |Y| + |Z| \leq |Y||Z| + |Z||Y| = 2|Y||Z|$. By (2.2), we may let $X = C \cap \overline{A}$, $Y = B \cap \overline{A}$ and $Z = C \cap \overline{B}$ and from both of the claims it follows that

$$\left|C \cap \overline{A}\right| \leq 2\left|B \cap \overline{A}\right| \left|C \cap \overline{B}\right|. \tag{2.4}$$

Taking the log on both sides of (2.4) and using the inequality $2 \leq |B|$ (which follows from $B \in \mathcal{P}_F^+(\Omega)$) we obtain

$$\begin{aligned} \log\left|C \cap \overline{A}\right| &\leq 1 + \log\left|B \cap \overline{A}\right| + \log\left|C \cap \overline{B}\right| \\ &\leq \log|B| + \log\left|B \cap \overline{A}\right| + \log\left|C \cap \overline{B}\right|. \end{aligned}$$

This proves (2.3).

Next, we define the information set-distance.

Definition 4. For any two finite non-empty sets A, B define the *information set-distance* as

$$d(A,B) := \max\{\delta(A,B), \delta(B,A)\}.$$

In the following result we show that d satisfies the properties of a semi-metric.

Theorem 5. *The distance function d is a semi-metric on $\mathcal{P}_F^+(\Omega)$. It satisfies the triangle inequality for any triplet $A, B, C \in \mathcal{P}_F^+(\Omega)$ such that no element in the triplet is strictly contained in any of the other two.*

PROOF. That the function d is symmetric is clear from its definition. From Remark 2 it is clear that for $A = B$, $\delta(A,B) = \delta(B,A) = 0$ hence $d(A,B) = 0$. Since $\delta(A,B) \geq 0$ and $\delta(A,A) = 0$ for all $A, B \in \mathcal{P}_F^+(\Omega)$ then we also have $d(A,B) \geq 0$ and $d(A,A) = 0$ for all $A, B \in \mathcal{P}_F^+(\Omega)$. This means that d is a semi-metric on $\mathcal{P}_F^+(\Omega)$. Next, we show that it satisfies the triangle inequality for any triplet $A, B, C \in \mathcal{P}_F^+(\Omega)$ such that no element is strictly contained in any of the other two. For any non-negative numbers a_1, a_2, a_3, b_1, b_2, b_3, that satisfy

$$\begin{aligned} a_1 &\leq a_2 + a_3 \\ b_1 &\leq b_1 + b_2, \end{aligned} \tag{2.5}$$

we have

$$\begin{aligned} \max\{a_1, b_1\} &\leq \max\{a_2 + a_3, b_2 + b_3\} \\ &\leq \max\Big\{\max\{a_2, b_2\} + \max\{a_3, b_3\}, \\ &\qquad \max\{b_2, a_2\} + \max\{b_3, a_3\}\Big\} \\ &= \max\{a_2, b_2\} + \max\{a_3, b_3\}. \end{aligned}$$

From Lemma 2 it follows that (2.5) holds for the following: $a_1 = \delta(A,C)$, $b_1 = \delta(C,A)$, $a_2 = \delta(A,B)$, $b_2 = \delta(B,A)$, $a_3 = \delta(B,C)$, $b_3 = \delta(C,B)$. This yields

$$d(A,C) \leq d(A,B) + d(B,C),$$

and hence d satisfies the triangle inequality for such a triplet.

Remark 6. One point about d that needs to be improved in the future is the fact that even if $\left|B \cap \overline{A}\right|$ and $\left|A \cap \overline{B}\right|$ are kept small while $|A|$ and $|B|$ increase, d will increase. It remains to be seen whether some form of normalization of d yields a distance that still satisfies the properties listed in Theorem 5.

3. DISTANCE BETWEEN STRINGS

Let us now define the distance between two binary strings. In this section, we take Ω to be a set $\mathbb{Y}$ of objects. Denote by $\mathbb{X}$ the set of all (finite) binary strings. Our approach to defining a distance between two binary strings $x, x' \in \mathbb{X}$ is to relate them to subsets $Y_x, Y_{x'} \in \mathcal{P}_F^+(\mathbb{Y})$ and measure the distance between the two corresponding subsets. Each string $x \in \mathbb{X}$ is a *description* of a corresponding set $Y_x \in \mathcal{P}_F^+(\Omega)$. Define a function $M : \mathbb{X} \to \mathcal{P}_F^+(\mathbb{Y})$ which dictates how a string x yields a set $M(x) := Y_x \subseteq \mathbb{Y}$. In general, M may be a many-to-one function since there may be several strings (viewed as descriptions of the set) of different lengths for a given set.

Definition 7. Let $\mathbb{X} \times \mathbb{Y}$ be all possible string-object pairs (x, y) and let M be any function $M : \mathbb{X} \to \mathcal{P}_F^+(\mathbb{Y})$. If $x, x' \in \mathbb{X}$ are two binary strings then the *information set-distance* between them is defined as

$$d_M(x,x') := d(M(x), M(x'))$$

where the function d is defined in Definition 4.

The next result follows directly from Theorem 5.

Corollary 8. *Let $\mathbb{Y}$ be a set of objects y and $\mathbb{X}$ a set of all finite binary strings x. Let $M : \mathbb{X} \to \mathcal{P}_F^+(\mathbb{Y})$ be any function that defines the set $Y_x \subseteq \mathbb{Y}$ of cardinality at least 2 described*

by x, for all $x \in \mathbb{X}$. The information set-distance $d_M(x, x')$ is a semi-metric on $\mathbb{X}$ and satisfies the triangle inequality for triplets x, x',x'' whose sets $M(x)$, $M(x')$, $M(x'')$ are not strictly contained in any of the other two.

As an example, consider a mapping M that takes binary strings to subets of $\mathbb{Y} = \{0,1\}^k$ (the k-cube) for some given finite k. The elements of $\mathbb{Y}$ are called k-words. Consider the following scheme for associating finite strings x with sets: given a string x, break it into non-overlapping k-words while, if necessary, appending zeros to complete the last k-word. Let the set $M(x) = Y_x$ be the collection of these k-words. For instance, if $x = 100100110$ then with $k = 4$ we we obtain the set $Y_x = \{1001, 0011, 0000\}$. If a string has $N > 1$ repetitions of some k-word then clearly only a single copy will be in Y_x. In this respect, M eliminates redundancy in a way that is similar to the method of [4] which gives the minimal number of copy operations needed to reproduce a string from a set of its substrings.

Another mapping M may be defined by scanning a fixed window of length k across the string x and collecting each substring (captured in the window) as an element of the generated set Y_x. For instance, suppose an alphabet has 26 letters. Hence there are 26^n possible n-grams (substrings made of n consecutive letters). If x is a document then it can be broken into a *set* $M(x)$ of n-grams. Each letter is represented by 7 bits. We collect words of length $k = 7n$ bits and shift by 7bits, repetitively. In this example d_M measures the distance between two documents. In comparison, the n-gram model in the area of information retrieval [1] represents a document by a binary *vector* of dimensionality 26^n where the i^{th} component is 1 if the document contains the i^{th} particular n-gram and is 0 otherwise. There a similarity (opposite of distance) between two documents is represented by the inner product of their corresponding binary vectors.

Yet another approach which does not need to choose a value for k is to proceed along the line of work of [4]. Here we can collect substrings of x (of possibly different lengths) according to a repetitive procedure in order to form the set $M(x)$ (in [4] the cardinality of the set $M(x)$ is referred to as the complexity of x).

Whichever scheme M is used, to compute the information set-distance $d_M(x, x')$ between two finite strings x and x' we first determine the sets $M(x)$ and $M(x')$ and then evaluate their distance according to Definition 7 to be $d(M(x), M(x'))$.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley, 1999.

[2] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–17, 1965.

[3] J. Ratsaby. On the combinatorial representation of information. In Danny Z. Chen and D. T. Lee, editors, *The Twelfth Annual International Computing and Combinatorics Conference (COCOON'06)*, volume LNCS 4112, pages 479–488. Springer-Verlag, 2006.

[4] J. Ziv and A. Lempel. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22 (3):75–81, 1976.

Better bounds for the bin packing problem with the "Largest Item in the Bottom" constraint

György Dósa* Zsolt Tuza† Deshi Ye‡

ABSTRACT

The (online) bin packing problem with LIB constraint is stated as follows: The items arrive one by one, and must be packed into unit capacity bins, but a bigger item can not be packed into a bin which already contains a smaller item. The number of used bins is minimized as usually.

We show that the performance bound of algorithm First Fit is not worse than $2 + 1/6$ for the problem, improving the previous best upper bound 2.5. If all items are not bigger than $1/d$, then we improve the previous best result $2 + 1/d$ to $2 + 1/d(d + 2)$, for any $d \geq 2$.

Moreover we define a problem with the generalized LIB constraint, where incoming items can not be packed into the bins of some already packed items. The incompatibility of the incoming item with some already packed items becomes to be known only at the arrival of the actual item, and is given by an undirected graph. We show that 3 is an upper bound for this general problem if some natural transitivity constraint is satisfied, but without this constraint no online algorithm can be c-competitive with any constant c.

1. INTRODUCTION

The classical bin packing problem has been extensively studied, where the problem is to pack a set of items, whose sizes are in $(0, 1]$, into the minimum number of bins with capacity 1. In this paper, we consider the bin packing problem with LIB (largest item in bottom) constraint. It is the classical bin packing problem with the additional requirement that in every bin larger (or longer) items must be placed below smaller (or shorter) items. In other words, items $p_1, p_2, ..., p_n$ arrive one by one (where $0 < p_i \leq 1$) and must be packed into bins of size 1, and the total size of items packed into a bin can not be bigger than 1 . With the additional LIB constraint, a later item p_i can not be packed into a bin already containing p_j, if $p_j < p_i$. (If all the information regarding the items is known in advance, then the problem is called *off-line problem*. The problem is called *online* if the items arrive one by one and we must make a decision upon the arrival of an item without knowing any information on future items.)

The LIB version of Bin Packing arises in the transportation applications where it is requested to have safe and stable packing (bigger item is not allowed to be packed on the top of some smaller item). The classical bin packing problem has many applications in computer science. With LIB constraint, the applications in computer science can be easily extended to the applications with priority. For example, in a multi-core system the CPU scheduler needs to assign the task with larger priority before the smaller priority in each core. However, each core has a fixed capacity of priority and the question is to minimize the number of cores used for serving all the tasks.

While applying to bin packing with LIB constraint, we describe the First Fit algorithm: FF places any incoming item i into the first bin B into which it fits, and no item j has already been packed into B such that $p_j < p_i$. If no such bin exists, a new bin will be opened and item i is packed into it.

1.1 Previous results regarding the LIB constraint

The bin packing problem with LIB constraint was introduced by Manyem, [15]. It was shown that NF is not constant approximated (algorithm NF or Next Fit at any time keeps open only one bin, and packs the next item into the open bin if fits, otherwise the bin is closed and the next item is packed into a new bin), but the competitive ratio of First Fit (FF) is at most 3. (The competitive ratio of an online algorithm A means the maximum of ratio $A(I)/OPT(I)$, where OPT is an optimal offline algorithm, and I is arbitrary input for the problem.) Epstein [5] improved the upper bound to 2.5, furthermore, Epstein [5] proved that the parametric upper bound of FF (where the size of each item is at most $1/d$) is at most $2 + 1/d$, for any integer $d \geq 2$. It was also shown that the competitive ratio of any online al-

*Department of Mathematics, University of Pannonia, Veszprem, Hungary, dosagy@almos.vein.hu.

†Computer and Automation Institute, Hungarian Academy of Science, Budapest, Hungary, and Department of Computer Science and Systems Engineering, University of Pannonia, Hungary, tuza@sztaki.hu

‡College of Computer Science, Zhejiang University, Hangzhou, China, yedeshi@zju.edu.cn

gorithm for bin packing with LIB constraint is at least 2, for any parameter d.

1.2 Our contribution

In this paper, we revisit algorithm FF, and give a more careful analysis. We show that the competitive ratio of algorithm FF is not worse than $2 + 1/6$, improving the previous best upper bound 2.5 of Epstein [5]. Moreover, we show that the parametric competitive ratio of FF is at most $2+1/d(d+2)$, again improving the previous upper bound $2 + 1/d$ of [5]. Finally, we also treat the problem in a generalized version, and prove upper and lower bounds. We do not present here the complete proofs, they will be published in the journal version of this paper [4].

1.3 Notation and preliminaries

Let a be any item. The bin where a is packed will be denoted by $B(a)$. The level of a bin B will be denoted as $l(B)$. After making a packing, we say that a bin B_{i_1} is earlier than bin B_{i_2}, or B_{i_2} is latter than bin B_{i_1}, if $i_1 < i_2$ holds.

We say that items $x_v, x_{v-1}, ..., x_1$ form a *chain*, if for any $1 \leq k < v$, x_{k+1} arrives before x_k, and $x_{k+1} < x_k$. Then from the LIB constraint follows that each element of the chain must be packed into different bins.

There are three lower bounds in the literature for the problem, namely: The total sum of the items and the length v of any chain are natural lower bounds on OPT, as easily seen. These lower bounds (total sum and length of longest chain) will be denoted by LB_1 and LB_2, respectively. The third lower bound [5] can be stated as follows:

LEMMA 1. *Let $x_v, x_{v-1}, ..., x_1$ form a chain, $x_1 \leq 1/2$, and suppose that there are u items arriving after x_1, each bigger than a half. Then $LB_3 = v + u$ is a lower bound for the problem.*

2. THE IMPROVED BOUNDS

Due to space limitation, here we can give only an outline of the proof of the statement that $FF/OPT \leq 2 + \frac{1}{d(d+2)}$ holds for all $d \geq 2$.

Let us run algorithm FF, the created bins will be called as FF-bins, furthermore we denote the number of used bins simply by FF, while let OPT denote the optimum number of bins in case of an offline solution. Let a be an arbitrary item, being packed not into the first FF-bin, and let B be an FF-bin with smaller index than $B(a)$. Then we say that a is **not** packed into bin B by overflowing, if at the moment when item a arrives, $l(B) + a > 1$ holds where $l(B)$ means the actual level of bin B, and at this time there is no item packed into bin B smaller than a. Furthermore, we say that a is **not** packed into bin B by the LIB constraint, if there is an item $b < a$, already packed into bin B, just at the moment when item a is revealed (no matter that $l(B) + a$ is bigger than one or not). Now we classify the FF-bins as follows. Let k be an arbitrary large positive integer.

Class H (bins with high level) will be the set of those bins where the level of the bin is at least $\frac{k(d+1)-1}{k(d+1)}$, the other bins form Class R (remaining bins).

If there is no R-bin at all, then our results follows easily, since then the total size of the items is at least $\frac{k(d+1)-1}{k(d+1)} \cdot FF > \frac{FF}{2}$, then $FF/OPT \leq 2$. Thus we suppose that there exist R-bins (at least one), then the level of each R-bin is smaller than $\frac{k(d+1)-1}{k(d+1)}$. Bins of these classes will be called as H-bins and R-bins, respectively.

Next we define a special chain. Let x_1 be an arbitrary item being packed into the **last** R-bin. Suppose that $x_1, x_2, ..., x_k$ are already defined. Then let x_{k+1} be last item being packed into some R-bin B before the arrival of item x_k, such that B is the last R-bin where x_k is not packed by the LIB constraint, if there exists such item. Let the length of the chain be $v \geq 1$; we define the chain in the reverse order $x_v, x_{v-1}, ..., x_1$. We call items $x_v, x_{v-1}, ..., x_1$ as chain-items, the bins of these items as chain-bins, let this set of bins be Class C (chain bins). Let the class of the other R-bins be Class M (bins of medium level). The bins of these classes will be called as C-bins and M-bins, respectively. Then it can be shown that the level of any M-bin is at least $\frac{d}{d+1}$, and we have already seen that the level of any M-bin is smaller than $\frac{k(d+1)-1}{k(d+1)}$.

Now we divide the M-bins into k subclasses, according to their levels, as follows: subclass M_i is defined as the set of M-bins having level at least $\frac{(d+1)k-i}{(d+1)k}$ but smaller than $\frac{(d+1)k-i+1}{(d+1)k}$, for $1 \leq i \leq k$. Since the level of any M-bin is at least $\frac{d}{d+1}$, these are all the M-bins. We also classify the chain bins, let C_i be the set of C-bins with level at least $\frac{i-1}{(d+1)k}$ and smaller than $\frac{i}{(d+1)k}$, for $1 \leq i \leq k$. Since the level of a chain-bin can be $\frac{1}{d+1}$ or bigger, let simply C_+ denote the C-bins with level at least $\frac{1}{d+1}$. In order to prove $FF/OPT \leq 2 + \frac{1}{d(d+2)}$, let h, c_i and m_i be the number of H-bins, C_i-bins, and M_i-bins, respectively, for $1 \leq i \leq k$ (then $m_1 = 0$), and let c_+ be the number of C_+-bins. Then we gain the next lower bounds for the optimum value:

$$OPT \geq \frac{(d+1)\,k-1}{(d+1)\,k} h + \sum_{i=1}^{k} \left(\frac{i-1}{(d+1)\,k} c_i + \frac{(d+1)\,k-i}{(d+1)\,k} m_i \right) + \frac{1}{d+1} c_+, \quad (1)$$

$$OPT \geq c_1 + c_2 + ... + c_k + c_+ \quad (2)$$

$$OPT \geq c_1 + ... + c_j + \sum_{i=j+1}^{k} \frac{(d+1)\,k-i}{(d+1)\,k} m_i, \quad (3)$$

for any $1 \leq j \leq k-1$.

After this treatment, we got an inequality from (1), also one inequality from (2), and $k-1$ inequalities from (3). We show in Table 1 the structure of the inequalities in case of $k = 4$. The variables are described on the top of the columns in the table, and the coefficients are written into the table.

Now we need to multiply each row by some multiplicator.

Then, if the sum of the multiplied coefficients is at least 1 in each column, we obtain the desired result. The question is: What is the best choice of the multiplicators? We want to ensure that after the multiplication of the rows, the sum of the numbers in each column be at least 1, and on the other hand, we want to keep the sum of the multiplicators as low as possible, since this sum is an upper estimate of the ratio FF/OPT. In case of the best choice, the coefficients are the optimal solutions of the dual of the Linear Program which consists of the mentioned inequalities, the nonnegativity assumption of the variables, and the objective is the sum of the variables: $z = h + \sum_{i=1}^{k} (c_i + m_i) + c_+$. We now make a little simplification on this optimal solution, to carry out a simplified analysis. The "cost" of this simplification is that the FF/OPT ratio is slightly increased, but the increased ratio still tends to the same limit as k tends to infinity. Using the optimal choice of the coefficients, we could not get better upper estimate for the FF/OPT ratio, only the analysis would be a bit harder.

c_1	h	c_2	m_2	c_3	m_3	c_4	m_4	c_+
	11/12	1/12	5/6	1/6	3/4	1/4	2/3	1/3
1			5/6		3/4		2/3	
1		1			3/4		2/3	
1		1		1			2/3	
1		1		1		1		1

Table 1: the structure of the lower bounds when big items can not occur

Here (in case $d = 2$ and $k = 4$) the chosen multiplicators are the following: We multiply the first row by $\alpha = \frac{6}{5}$, the next three rows by $\beta = \frac{1}{10}$, and the last row by $\gamma = \frac{41}{30}$. Then the sum of the multiplicators is $\alpha + (k-1)\beta + \gamma = \frac{6}{5} + 3 \cdot \frac{1}{10} + \frac{41}{30} = \frac{43}{15} \approx 2.8667$; generally, $\alpha = \frac{(d+1)^2 k}{d(d+2)k-d}$, $\beta = \frac{d+1}{d(d+2)k-d}$, and $\gamma = \frac{kd^2+kd-k+1}{d(d+2)k-d}$. Then it is easy to check that the sum of the multiplicators (in case of $d = 2$) tends to $2 + \frac{1}{d(d+2)} = 2 + \frac{1}{8} = 2.125$ as $k \to \infty$, (for $k = 6$ this slightly bigger upper estimate is already $\alpha + (k-1)\beta + \gamma = \frac{54}{46} + 5 \cdot \frac{3}{46} + \frac{31}{46} = \frac{50}{23} \approx 2.173\,9$) and the case is the same for any d: we slightly modify the best solution of the dual, the sum of the multiplied rows is at least 1 in each column, and the sum of the multiplicators tends to $2 + \frac{1}{d(d+2)}$, for any $d \geq 2$.

We gain the next theorem:

THEOREM 2. *The parametric competitive ratio of algorithm FF is not bigger than* $2 + \frac{1}{d(d+2)}$, *for any* $d \geq 2$.

2.1 If items bigger than a half can occur

Now let us consider the case $d = 1$, i.e. there can be items bigger than a half. Note that the previous best upper bound is 2.5. Note, that by the previous treatment which is given in the previous chapter for $d \geq 2$, we already could have a better upper bound, since $2 + \frac{1}{d(d+2)} = 2 + 1/3$, if $d = 1$, and this general proof also works for $d = 1$. But with a little bit more careful analysis than that we made for cases $d \geq 2$, we can derive a better bound, namely we can show that if items bigger than a half can occur, then $FF/OPT \leq 2+1/6$ still holds. The proof is quite similar to that in the previous section but we need one more type of lower bound (used never before) on the optimum value.

3. THE GENERALIZED LIB CONSTRAINT

We define a constraint here, which is more general than the (pure) LIB scenario. We still assume that the items with sizes $p_1, p_2, \ldots$ arrive in a sequence revealing one item at a time. Receiving p_j we also get the list of those p_i $(i < j)$ which cannot occur in the same bin as p_j, but nothing is known about later items. Each item must be packed into a bin immediately when it arrives. The case can be totally described with the help of an *incompatibility graph* G whose vertices have weights representing item sizes, and two vertices are adjacent if and only if the corresponding two items cannot be packed into the same bin. In this situation we say that (p_i, p_j) is an *incompatible pair*, or p_i is incompatible with p_j (and vice versa). Also here, FF packs the next item to the bin of smallest index where the item is not incompatible with any previous item there, and does not increase the level of the bin above 1.

Note that the incompatibility graph is known by any offline algorithm, thus it is also known by the optimal algorithm, to which the online algorithm is compared, but in the online case only the already arrived part of the graph is known by the online algorithm.

3.1 With transitivity assumption

Here we consider a subcase that we call the *generalized transitive LIB constraint*. It assumes for any $i < j < k$ that if the pairs (p_i, p_j) and (p_j, p_k) are incompatible then so is (p_i, p_k), too.

PROPOSITION 3. *Under the generalized transitive LIB constraint,* $FF/OPT \leq 3$ *holds for any input.*

The proof can be found in the journal version [4].

3.2 Without transitivity

Here we observe that the generalized LIB constraint, without any restriction make it impossible to put any constant upper bound on competitive ratio.

PROPOSITION 4. *Under the generalized LIB constraint, no online algorithm can be c-competitive, for any constant c. Moreover, this fact holds true even if the incompatibility graphs of input sequences are required to be trees, the length n of input sequence is known in advance, and all items are of size* $p_i = 1/n$. *In particular,* FF/OPT *is unbounded.*

The proof is based on a result of [1,9], dealing with online coloring of trees. □

4. CONCLUSION AND OPEN QUESTIONS

We have addressed the bin packing problem with the so-called "LIB" constraint, where a bigger item cannot be packed into a bin which already contains a smaller item. We showed that the competitive ratio of the algorithm First Fit (FF) is not worse than $2 + 1/6$. Furthermore, in the parametric version of the problem, where all items are not bigger than $1/d$, we proved that the competitive ratio is at most of $2 + 1/d(d + 2)$, for any $d \geq 2$. The above results significantly improve the previous upper bound 2.5 for the original problem and $2+1/d$ for the parametric version, respectively [5]. Finally, we investigated a generalized version called "bin packing with incompatibility graph" under the transitivity constraint. This means that if the pairs (p_i, p_j) and (p_j, p_k) are incompatible for some $i < j < k$, then so is (p_i, p_k), too. We showed that in this case the competitive ratio of algorithm FF is at most 3. There remain gaps between the upper and lower bounds for the problem bin packing with LIB constraint. It is worth investigating other problems of interest, too, for example some special cases of the general problem with conflicts but without transitivity. It remains open for later research to discover relaxations of transitivity which still admit a constant competitive ratio. These questions remain for further research.

5. REFERENCES

[1] D.R. Bean, Effective coloration, *J. Symbolic Logic* 41:2 (1976), 469–480.

[2] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[3] E.G. Coffman jr, M.R. Garey, and D.S. Johnson, Approximation algorithms for bin packing: A survey, In: *Approximation algorithms for NP-hard problems*, pages 46–93, PWS Publishing Co., 1996.

[4] Gy. Dosa, Zs. Tuza, D. Ye, Bin packing with "Largest In Bottom" constraint: Tighter bounds and a generalization, to appear.

[5] L. Epstein, On online bin packing with LIB constraints, *Naval Research Logistics*, 56(8):780–786, 2009.

[6] L. Epstein and A. Levin, On bin packing with conflicts, *Proceedings of Workshop on Approximation and Online Algorithms*, pages 160–173, 2007.

[7] L. Epstein, A. Levin, R. Van Stee, Multi-dimensional packing with conflicts, *Fundamentals of Computation Theory*, pages 288–299, 2007.

[8] L. Finlay and P. Manyem, Online LIB problems: Heuristics for Bin Covering and lower bounds for Bin Packing, *RAIRO Oper. Res*, 39:163–183, 2005.

[9] A. Gyárfás and J. Lehel, On-line and first fit colorings of graphs, *J. Graph Theory* 12(2): 217–227, 1998.

[10] K. Jansen, An approximation scheme for bin packing with conflicts, *Journal of combinatorial optimization*, 3(4):363–377, 1999.

[11] K. Jansen, S. Öhring, Approximation algorithms for time constrained scheduling, *Information and Computation*, 132:85¨C108, 1997.

[12] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham, Worst-case performance bounds for simple one-dimensional packing algorithms, *SIAM Journal on Computing*, 3:25–278, 1974.

[13] C. Lund and M. Yannakakis, On the hardness of approximating minimization problem, *25th Symposium on the Theory of Computing*, pages 286–293, 1993.

[14] P. Manyem, Uniform sized bin packing and covering: Online version, *Topics in Industrial Mathematics, J.C. Misra (Editor), Narosa Publishing House, New Delhi*, pages 447–485, 2003.

[15] P. Manyem, R. Salt, and M. Visser, Approximation lower bounds in online LIB bin packing and covering, *Journal of Automata, Languages and Combinatorics*, 8(4):663–674, 2003.

[16] B. McCloskey, and A. Shankar, Approaches to bin packing with clique-graph conflicts, *Technical Report UCB/CSD-05-1378, EECS Department, University of California, Berkeley*, 2005.

[17] S. Seiden, On the online bin packing problem, *Journal of the ACM (JACM)*, 49(5):640–671, 2002.

[18] A. van Vliet, An improved lower bound for on-line bin packing algorithms, *Information Processing Letters*, 43(5):277–284, 1992.

Semi-on-line bin packing: an overview and an improved lower bound

János Balogh
Department of Applied Informatics
Juhász Gyula Faculty of Education
University of Szeged
Boldogasszony sgt. 6, H-6725 Szeged, Hungary
balogh@jgypk.u-szeged.hu

József Békési
Department of Applied Informatics
Juhász Gyula Faculty of Education
University of Szeged
Boldogasszony sgt. 6, H-6725 Szeged, Hungary
bekesi@jgypk.u-szeged.hu

ABSTRACT

Here we review the main results in the area of semi-on-line bin packing. We describe our bounds for semi-on-line bin packing problems based on our papers. Then we present a new lower bound for the asymptotic competitive ratio of any on-line bin packing algorithm which knows the optimum value in advance.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems—*Computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

General Terms

Theory

Keywords

Bin packing, semi-on-line algorithms, asymptotic competitive ratio, lower bound

1. INTRODUCTION

In computer science the classical one-dimensional bin packing problem is among the most frequently studied combinatorial optimization problems. In its traditional form a list $L = x_1, x_2, \ldots, x_n$ of elements (also called items) with sizes in the interval $(0, 1]$ and an infinite list of unit capacity bins are given. Each element x_i from the list L has to be assigned to a unique bin such that the sum of the sizes of the elements in a bin does not exceed the bin capacity. The size of an element is also denoted by x_i. The *bin packing problem* involves packing the items to the bins in such a way that as few bins as possible are used.

It is well known that finding an optimal packing is NP-hard [20]. Consequently, a large number of papers have been published which look for polynomial time algorithms that find feasible solutions with an acceptable approximation quality.

One class of bin packing algorithms is the so-called *on-line algorithms*. An on-line algorithm packs the elements in order of their arrival, without any knowledge of the subsequent elements. (Neither the size nor the number of the elements are known.) The packed elements cannot be removed. In contrast *off-line algorithms* have complete information about the list in advance, which they can take into consideration during their operation.

The so-called *semi-on-line algorithms* [6, 10] lie between the well-known on-line and off-line ones. For such algorithms, at least one of the following operations is allowed: repacking of some items (e.g. [19, 24]), lookahead of the next several elements [21], or some kind of pre-ordering.

The structure of this article is the following. In Section 2 we review the results of semi-on-line bin packing, then in Section 3 we improve the lower bound for the asymptotic competitive ratio of any on-line (one-dimensional) algorithm which knows the optimum value in advance. For the interested reader, we recommend [13], where a motivation of this problem can be found as well. In the bin packing literature, a method for packing patterns is often used to provide new lower bounds for different on-line bin packing problems (see e.g. [1, 16, 35]). Here we extend the method of packing patterns, giving a new construction for the LP based on packing patterns. The new construction is based on "branching" (input) list sequences.

2. DEFINITIONS AND EARLIER RESULTS

To measure the efficiency of algorithms, two general methods are available: an investigation of the worst-case behavior or, assuming some probability measures, a probabilistic analysis. Here we will concentrate on the asymptotic worst-case behavior of an algorithm. For a given list L, let $A(L)$ and $OPT(L)$ stand for the number of bins used by algorithm A and the number of bins used in an optimal packing, respectively. Then the *asymptotic competitive ratio (ACR)* of algorithm A is

$$R(A) := \limsup_{l \to \infty} \left\{ \max_{L} \left\{ \frac{A(L)}{l} \middle| OPT(L) = l \right\} \right\}. \qquad (1)$$

If an algorithm has a finite ACR, and its value is less than or equal to t, then we say that the algorithm is t-competitive.

For off-line algorithms Fernandez de la Vega and Lueker [14] provided an APTAS (Asymptotic Polynomial Time Approximation Scheme), while Karmakar and Karp [28] developed the first AFPTAS (Asymptotic Fully Polynomial Time Approximation Scheme). In [14] for any $\varepsilon > 0$ an algorithm A_ε is given such that each A_ε runs in polynomial time in the length of the input list L (but exponential in $1/\varepsilon$) and has $A_\varepsilon(L) = (1+\varepsilon)\, OPT(L) + 1$. In [28], a more complex algorithm is given. The running time of this depends on n and $1/\varepsilon$ polynomially and $A_\varepsilon(L) \leq OPT(L) + \log^2(OPT(L))$ holds for this.

The current best on-line algorithm with the best known ACR was defined by Seiden [34] in 2002, and it is called *Harmonic++*. Seiden (improving Richey's earlier method and analysis [34]) proved that the ACR of his algorithm is at most 1.58889. Harmonic++ belongs to the class of *Super Harmonic* algorithms defined in the same paper. For each algorithm from this class, a lower bound of 1.58333 is valid [32], so $1.58333 \leq R(\text{Harmonic++}) \leq 1.58889$. Note that the last previous actual best algorithms were Super Harmonic [29, 32, 34] as well. For on-line algorithms the best known lower bound is 1.5403 [1].

In this paper we will focus on semi-on-line (SOL) bin packing algorithms. The most famous algorithms based on pre-ordering are FFD and BFD. They sort the elements in decreasing order and pack them using the First Fit (Best Fit) strategy. Johnson proved in [27] that $R(\text{FFD}) = R(\text{BFD}) = 11/9 = 1.22222....$ More precisely, Johnson proved that for each list L $\text{FFD}(L) \leq (11/9)\, OPT(L) + 4$. The tight value 6/9 of the additive term was given by Dósa [11], proving that $\text{FFD(L)} \leq 11/9\, OPT(L) + 6/9$. For on-line algorithms running on pre-ordered lists we improved the previous lower bound from $8/7 = 1.1428...$ [9] in [1] to $54/47 = 1.1489....$ The first semi-on-line bin packing algorithm with repacking was given by Galambos [15] for the case when a restricted number of bins can be open. A bin is called closed if we cannot pack elements into it later. This algorithm uses two buffer bins for storing the elements temporarily. Improving on this, Galambos and Woeginger [17] defined a semi-on-line repacking algorithm using 3 buffer bins. Its ACR is 1.69103..., which is optimal among the algorithms with restricted number of open bins.

To describe the subclasses of semi-on-line bin packing algorithms, we need to introduce the role of the "scheduler" and the "packer". The role of the scheduler is to produce the input list, while the role of the packer is to pack the items (that is, to implement the packing algorithm).

In the above-mentioned problem classes, the role of the scheduler is trivial: to supply the elements one by one and to mark the end of the (whole) list.

Gutin et al. introduced [22] the so-called *batched bin packing problem* (BBPP) in 2005, which is a semi-on-line bin packing problem. The classical problem is modified in such a way that the input is split into parts (called batches) by the scheduler. For each step the scheduler either gives a new element, or marks the end of the current batch. Each batch is available only after the processing of the previous one. The algorithm has to pack each batch as an off-line one (that is, a lookahead is possible within the current batch). But lookahead is not allowed outside the current batch, so each batch has to be packed in an on-line manner, where during the packing of a new batch the elements of the earlier packed batches cannot be moved. One batch can consist of more elements or can be empty. If every batch has exactly one element, then we get the classical on-line bin packing problem as a special case. If an input consists of exactly m batches, then we call this BBPP as *m-BBPP*. The ACR of the algorithms for this problem version can be defined the same way as before. Gutin et al. study the 2-BBPP problem and its variants in [22] in detail. For the 2-BBPP problem, they proved an 1.3871... lower bound here and they derived bounds for those special cases of the 2-BBPP problem, where the number of the different sizes of the elements of the list are bounded from above by a given $p \geq 2$ positive integer. They proved, using the results of their paper [23], that if $p = 2$ then their bound is optimal. They raise an open question of whether for different $p(> 2)$ values their bound is optimal or not.

There is a connection between bin packing and memory allocation scheduling task of computer programs (jobs). The size of a job corresponds to the size of an element, while a memory partition corresponds to the capacity of a bin. The most important difference is that a memory area ordered to a job can be released after finishing the job. To model this, a new class of bin packing algorithms can be defined; the algorithms for *dynamic bin packing problem* (DBP), where the scheduler can specify the arrival of an element (*Insert* operation) or removal of a previously inserted element (*Delete* operation). Each step of the input is one of these specifications, thus an input list is a finite series of Insert and Delete operations. We should add that a removal can only be a part of the input, i.e. the algorithm, or the packer cannot delete an element; only the input provider (the scheduler) can do this. The number of bins used by a dynamic bin packing algorithm A can be defined as the maximum of the nonempty bins used during the steps. Then the asymptotic competitive ratio $R(A)$ of a dynamic bin packing algorithm A can be defined in a similar way as earlier by formula (1). It is not hard to see that the original bin packing problem is the special case of the dynamic problem, where the list consists of inserts only. The dynamic bin packing problem was defined and analyzed by Coffman, Garey and Johnson in [7]. They gave and analyzed approximation algorithms for the problem.

When the scheduler can apply only Insert operations (no Delete) with repacking, Ivkovič and Lloyd proved in [26] that for any $\varepsilon > 0$ there is a $(1+\varepsilon)$-competitive approximation scheme A, that requires $O(\log n)$ amortized time per Insert operation and there is a $(1+\varepsilon)$-competitive fully polylogarithmic approximation scheme A that requires $O(\log^2 n)$ amortized time per Insert operation.

For the same bin packing problem, Epstein and Levin [12] gave an APTAS. In their model the total size of the elements moved per step (Insert operation) is bounded by β times the size of the arriving element. Their *Algorithm Dynamic APTAS* uses at most $(1+\varepsilon)\, OPT(L) + 1$ bins.

A special case of the dynamic bin packing problem is the so-

called *fully dynamic bin packing* (FDBP, [24, 25]) problem. The difference between this and dynamic bin packing problem is that the packer is allowed to perform repacking as well. If the repacking is restricted, i.e. for each step the packer can repack at most c items, then it is called *c-repacking fully dynamic bin packing problem* (c-repacking FDBP). Here c is a fixed positive integer. Obviously, the case $c = 0$ is simply the pure on-line bin packing problem.

The classical on-line bin packing can also be relaxed by allowing the repacking of at most c elements for each step. This version of the problem is called c-repacking semi-on-line bin packing (c-repacking SOL). Obviously, the case $c = 0$ again gives the pure on-line bin packing problem.

Many years after Galambos's first paper on repacking algorithms, Gambosi et al. [18, 19] returned to the analysis of certain semi-on-line algorithms. Their algorithm used repacking, but the application and the cost of the algorithm was defined in a special way. Their method packs the large elements one by one, while it composes bundles (groups) of the small items. Then one group is moved in one step, and this kind of movement counts as 1-repacking. In this sense these algorithms may move even $O(n)$ items in one step. In paper [19] the authors analyzed two algorithms. The faster, linear time algorithm A_1 has an ACR of 3/2, while the other algorithm A_2 with $O(n \log n)$ running time has an ACR of 4/3.

Ivkovič and Lloyd investigated the FDBP problem. Their algorithm, similar to the technique presented in [19], uses the bundle technique for the small items. Its ACR was 5/4 [25].

Up to the last decade no lower bounds were given for the efficiency of the semi-on-line algorithms. The first paper from this point of view was published by Ivkovič and Lloyd in 1996 [24]. They proved that there is no c-repacking FDBP algorithm which has a better asymptotic competitive ratio than 4/3. With a small modification the construction can be applied for the c-repacking SOL problem as well. This means that the lower bound 4/3 becomes valid for this problem, as was mentioned in [10] by Csirik and Woeginger. Note that the bound of 4/3 is valid for both problems for any c.

2.1 Our earlier results

In [3], we improved the best-known lower bound from 1.3333 [24] to 1.3871 for the c-repacking semi-on-line problem. We presented our proof for the c-repacking SOL problem, and we showed that it remains valid for the c-repacking FDBP problem as well. The results obtained are valid for any c. We proved the lower bound by analyzing and solving a specific optimization problem. For the analysis of the construction we used different methods: LP-techniques were combined with results from linear algebra, and then we solved and analyzed a special non-linear optimization problem. We expressed the exact value of the lower bound in terms of the Lambert W function. Note that our construction is a generalization of the constructions described in [22] and [24].

We proved some lower bound results for the special case of the problem as well. In [2] we focused on the special case of the problem, where there is another condition: the maximal number of the different elements is restricted by a given $p (p \geq 2)$ constant. The lower bounds are valid for the special cases of both above-mentioned semi-on-line problems. Furthermore, they are valid for the 2-BBPP problem defined in [22]. The bounds improve the lower bounds for the case $p \geq 3$ given in [22] (and they are valid for both the c-repacking and classical versions of the problem). Hence we answered the open question in [22] for the 2-batched bin packing problem, allowing at most p different item sizes. In [2] we proved that our construction can improve these bounds, giving a negative answer for their optimality. The construction works for specific p ($p > 2$), and the lower bounds are valid for the above two problems, the c-repacking SOL problem and the c-repacking FDBP.

In [4] we proved that although our bounds are given in one dimension, the above-mentioned lower bounds are valid for every d-dimension ($d \geq 1$) for multidimensional (geometric) semi-on-line bin packing problems. To the best of our knowledge this is the first multidimensional semi-on-line bin packing result. The lower bound construction is given by hypercubes and so it is valid for the classical d-dimensional on-line hypercube packing problem, even when repacking is not allowed. The bound improves the lower bound of 4/3 ([8]), which was the best known lower bound for cases $d \geq 4$.

The interesting aspect of the lower bounds presented in [2] and [3] is that although the problem is clearly discrete, the solution requires that one solve specific nonlinear (continuous) optimization problems. To do this, we applied global optimization methods. So the construction furnished a nice connection between combinatorial and global optimization. In [2], we solved the nonlinear optimization problem by a reliable Branch and Bound method based on interval arithmetic [30, 31]. Reliability means that the proof is produced by a computer, but the results are checked and the inaccuracy arising from rounding errors is eliminated.

In another paper [5], we improved our previous results and gave new upper bounds for the c-repacking SOL problem. We analyzed our algorithm using classical methods (like the weighting function technique). We gave a series of algorithms HFR-c for any positive integer value c. We proved that if c goes to infinity, then R(HFR-c) goes to 3/2. More precisely, we proved that the ACR for a given c is not larger than $3/2 + bc/(1 - bc)$, where bc is in the interval $(0, 1/(6c)]$. The given upper bounds show that repacking really can help. To demonstrate this we state some cases of these. In the case of $c = 1$, the result of our algorithm is irrelevant from the point of view that the best on-line algorithm is more competitive [34]. However, our 1-repacking algorithm uses far fewer bin classes. The ACR of our algorithm HFR-2 for the case of $c = 2$ is smaller or equal to 1.5728..., which is less than the best known ACR for on-line algorithms [34]. Another interesting aspect of the case $c = 2$ is that the given ACR is smaller than the lower bound 1.58333 proved for Harmonic Fit type on-line algorithms [32]. The same holds for our algorithm HFR-3 in the case $c = 3$: R(HFR-3) $\leq 1.5507...$. Another important issue is that for the case of $c = 4$ our algorithm has a better ACR than the best lower bound for on-line algorithms [1]. This means that it is more competitive than any pure on-line algorithm. Of course, this is due to the semi-on-line property, which permits repacking.

This shows us that it is worth investigating such algorithms.

Many open issues remain, of which we shall mention three. The first is to give a 1-repacking algorithm with an ACR better than 1.58333. The second is to provide a semi-on-line c-repacking algorithm with an ACR better than 1.5403 — for $c < 4$. The third is to improve the lower bounds for small c values ($c = 1, 2$).

3. IMPROVED LOWER BOUND FOR A SEMI-ON-LINE PROBLEM

Epstein and Levin considered a subtask in [13], when the on-line bin packing algorithm knows the value of the optimum in advance. In their paper they proved a lower bound of 1.30556. Here we raise this lower bound to 1.32312.

The packing pattern technique is described in the literature of bin packing ([1, 16, 35]). It is generally used in the lower bound constructions for bin packing problems. The set of packing patterns of a given sequence of items can be found. Based on this set, an LP can be constructed and the optimum value of the LP is found for the lower bound of the sequence.

We extend this LP-method in an adaptive way to the so-called "branching" list sequences. The purpose of this extension is to get an automatic method for constructing the LP in a new way, so we provide a technique for computing the lower bound for these kind of lists. Similar to the case of on-line bin packing algorithms, each list sequence contains a finite number of different lists. The lower bound will be the optimum value of the constructed LP. And like the on-line case it always exists.

As far as we know there is no known similar extension of the LP-method. In their construction, Epstein and Levin [13] used this branching list technique, but their result was proved using combinatorial methods. We think that our new method can result in new lower bounds for many different bin packing problems. As a by-product, we slightly improve the lower bound given in [13] for the bin packing problem with conflicts for interval graphs. The latter fact follows from a result presented in [13].

3.1 Packing patterns for branching lists and the construction of the LP

Let us consider r different list sequences. Denote these by $L_1, \ldots, L_r$, respectively. Let L_i be the concatenation of n_i number of lists, i.e. $L_i = L_{i,1} \ldots L_{i,n_i}$ $(i = 1, \ldots, r)$. Let $L_{i,j}$ also be a concatenation of $n_{i,j} = c_{i,j}n$ elements such a way that each element of $L_{i,j}$ has the same size. Denote this size by $s_{i,j}$ ($c_{i,j} > 0$ and let $0 < s_{i,j} \leq 1$ be real numbers and $j = 1, \ldots, n_i$, $i = 1, \ldots, r$ are integers). Denote the optimum values of the list sequences by $OPT(L_1), \ldots, OPT(L_r)$, respectively. In our construction all the optimum values will be equal, whose value will be denoted by N^*.

Let us suppose that a one-dimensional deterministic on-line bin packing algorithm A is executed on the lists $L_1, \ldots, L_r$. The value of N^* is known by A in advance. As a first step we will find the bin types which can be used by the algorithm. Next, we are interested in finding the number of bins of each bin type.

The packing patterns describe how the items of a list sequence L_i $(i = 1, \ldots, r)$ are distributed among the bins. The set of packing patterns for a fixed list L_i is denoted by P^i $(i = 1, \ldots, r)$. The elements of P^i are n_i-dimensional vectors. Each $p_i = (p_{i,1}, \ldots, p_{i,n_i})$ is a non-negative integer vector, for which $p_{i,1}s_{i,1} + \ldots + p_{i,n_i}s_{i,n_i} \leq 1$ holds. Here we also define a class of a packing pattern $p_i \in P^i$: $class(p_i)$ as the smallest j for which $p_{i,j} \neq 0$ in p_i.

Our procedure is the following:

1. First for each list sequence L_i, we will construct equations for the number of elements in the packing. Each such equation describes all the elements of $L_{i,j}$ can be found in the packing. (Each element is anywhere in a bin of the packing.) For each sublist $L_{i,j}$ we have one equation of the form

   $$\sum_{p_i \in P^i} p_{i,j} n(p_i) = n_{i,j}, j = 1, \ldots, n_i, i = 1, \ldots, r,$$

   where $n(p_i)$ is the number of bins used by the algorithm A while packing the elements of the sublist L_i.

   Thus for each list L_i we have n_i equations, which means that the total number of such constraints is $n_1 + \ldots + n_r$.

2. Algorithm A packs each list L_i, $i = 1, \ldots, r$, so, for $R(A)$

   $$R(A) \geq \limsup_{n \to \infty} \frac{A(L_i)}{OPT(L_i)}$$

   holds. It produces one constraint for each list L_i, giving additional r inequalities. We note that there is a difference here compared to the classical on-line algorithms, because we do not add these inequalities for the real sublists of a list L_i (i.e. for the lists $L_{i,1}, \ldots L_{i,t}$ of a list L_i, if $t < n_i$). This arises from the nature of the problem (since it is the case of the known optimum value).

   It means that

   $$\sum_{p_i \in P^i} n(p_i) \leq R(A)\, OPT(L_i),\ i = 1, \ldots r.$$

   Substituting the previous expression into $OPT(L_i)$, we get that

   $$\sum_{p_i \in P^i} n(p_i) \leq R(A) N^*,\ i = 1, \ldots, r,$$

   where the values of $n(p_i)$-s are non-negative integers.

3. To obtain good lower bounds we have to find some connections among the lists. From this reason we consider a kind of sequences that can have common prefix lists. We call these kinds of lists branching lists. A branching list can be represented by a tree, which can be constructed based on the common prefix portions of the different lists.

   Let us now ask the following question of how these connections can appear in the constraints of the LP, which is assigned to the list construction.

Consider all pairs L_i, L_j of the lists ($1 \leq i < j \leq 1$). For every such pair we consider the largest index k, for which $L_{i,1} = L_{j,1}, \ldots, L_{i,k} = L_{j,k}$ holds, It means that if $L_i = L_{i,1} \ldots L_{i,k} L_{i,k+1} \ldots L_{i,n_i}$, then L_j can be written in the form $L_j = L_{i,1} \ldots L_{i,k}\ L_{j,k+1} \ldots L_{j,n_j}$, where $L_{i,k+1} \neq L_{j,k+1}$. (at most one of the lists $L_{i,k+1}$ and $L_{j,k+1}$ can even be empty).

The LP contains additional constraints (equations) for every such i, j pair of indices. More precisely, the number of these additional equations for a given pair i, j is equal to the number of different packing patterns containing an element of the common prefix sublist $L_{i,1} \ldots L_{i,k}$. The set $P^{i,k}$ of these packing patterns contains vectors of dimension n_i. This is the subset of P^i that contains the kind of elements of P^i containing an element of the first k sublist of L_i. Therefore

$$P^{i,k} = \{p_i \mid class(p_i) \leq k\},$$

where, of course, $P^i = P^{i,n_i}$,

$$\sum_{p_i \in P^{i,k}, p_i = (p_{i,1}, \ldots, p_{i,k}, p_{i,k+1}, \ldots, p_{i,n_i})} n(p_i) = \sum_{p_j \in P^{j,k}, p_j = (p_{i,1}, \ldots, p_{i,k}, p_{j,k+1}, \ldots, p_{j,n_j})} n(p_j),$$

for all packing patterns $p_i \in P_{i,k}$, $0 \leq i < j \leq 1$, where $n(p_i)$ and $n(p_j)$ are the number of bins of type p_i and p_j used by the algorithm in the packing of the lists L_i and L_j, respectively. Each equation expresses the equality of the number of bins packed using packing patterns $p_i \in P^{i,k}$ and $p_j \in P^{j,k}$, where the first k values are the same, i.e.

$$p_i = (p_{i,1}, \ldots, p_{i,k}, p_{i,k+1}, \ldots, p_{i,n_i}),$$
$$p_j = (p_{i,1}, \ldots, p_{i,k}, p_{j,k+1}, \ldots, p_{j,n_j}),$$

where

$$p_{j,1} = p_{i,1}, \ldots, p_{j,k} = p_{i,k}.$$

We note that the number of this kind of constrains may be large.

4. Next, we will consider an algorithm A which gives a minimum value of $R(A)$ subject to all of the above constraints. Our aim is to compute the value

$$R = \min_A \max_{i=1,\ldots,k} \limsup_{n \to \infty} \frac{A(L_i)}{OPT(L_i)}.$$

This value will be a lower bound for the ACR of any on-line algorithm.

Summarizing the previous constraints, the LP can be given in the following form:

$$\min\ R, \tag{2}$$

subject to

$$\sum_{p_i \in P^i} p_{i,j} n(p_i) = n_{i,j}, j = 1, \ldots, n_i, i = 1, \ldots, r, \tag{3}$$

$$R \geq \frac{A(L_i)}{OPT(L_i)} = \frac{\sum_{p_i \in P^i} n(p_i)}{N^*},\ i = 1, \ldots, r, \tag{4}$$

$$\sum_{p_i \in P^{i,k}, p_i = (p_{i,1}, \ldots, p_{i,k}, p_{i,k+1}, \ldots, p_{i,n_i})} n(p_i) = \sum_{p_j \in P^{j,k}, p_j = (p_{i,1}, \ldots, p_{i,k}, p_{j,k+1}, \ldots, p_{j,n_j})} n(p_j),$$
$$\forall p_i \in P_{i,k}, 0 \leq i < j \leq 1. \tag{5}$$

This is a mixed integer programming problem, which is solvable.

3.2 The list construction

After these preliminaries we are ready to present an improved lower bound for the bin packing problem with known optimum. Consider the following branching list construction:

$L_1 = L_{1,1}, L_{1,2}$, where
$L_{1,1}$ contains n items, each with size $\frac{1}{40} - 4\varepsilon$, $L_{1,2}$ contains $\frac{39}{40}n$ items with size 1,

$L_2 = L_{2,1}, L_{2,2}, L_{2,3}$, where
$L_{2,1}$ contains n items of size $\frac{1}{40} - 4\varepsilon$, $L_{2,2}$ contains n items of size $\frac{1}{40} + \varepsilon$, $L_{2,3}$ contains $\frac{19}{20}n$ items of size 1,

$L_3 = L_{3,1}, L_{3,2}, L_{3,3}, L_{3,4}$, where
$L_{3,1}$: n items of size $\frac{1}{40} - 4\varepsilon$, $L_{3,2}$: n items of size $\frac{1}{40} + \varepsilon$, $L_{3,2}$: n items of size $\frac{1}{5} + \varepsilon$, $L_{4,3}$: $\frac{3}{4}n$ items of size 1,

$L_4 = L_{4,1}, L_{4,2}, L_{4,3}, L_{4,4}, L_{4,5}$, where
$L_{4,1}$: n items of size $\frac{1}{40} - 4\varepsilon$, $L_{4,2}$: n items of size $\frac{1}{40} + \varepsilon$, $L_{4,3}$: n items of size $\frac{1}{5} + \varepsilon$, $L_{4,4}$: n items of size $\frac{1}{4} + \varepsilon$, $L_{4,5}$: $\frac{1}{2}n$ items of size 1.

$L_5 = L_{5,1}, L_{5,2}, L_{5,3}, L_{5,4}, L_{5,5}$, where
$L_{5,1}$: n items of size $\frac{1}{40} - 4\varepsilon$, $L_{5,2}$: n items of size $\frac{1}{40} + \varepsilon$, $L_{5,3}$: n items of size $\frac{1}{5} + \varepsilon$, $L_{5,4}$: n items of size $\frac{1}{4} + \varepsilon$, $L_{5,5}$: n items of size $\frac{1}{2}$.

It can be seen that after the common portions the lists are filled with padding items 1 to attain the common optimum value. It is not hard to see that for the optimum values $OPT(L_1) = \ldots = OPT(L_5) = N^* = n$ holds, so this value can be given for the algorithm in advance. Constructing and solving the LP (2)-(5) leads to optimum value of 1.3231. This means that for the ACR of any on-line algorithms with known optimum value we have a lower bound of 1.3231. This raises the earlier bound of 1.30556 of Epstein and Levin for this problem. It provides a new lower bound for the competitive ratio of any on-line algorithm for the bin packing with conflicts problem on interval graphs. For the latter problem the lower bound 4.30556 is raised to 4.3231.

4. SUMMARY

Here we provided an overview of known semi-on-line bin packing results and presented a new lower bound for those

on-line bin packing problem where the algorithm knows the value of the optimum in advance. We introduced the concept of branching input lists and we gave a new method for constructing the LP. It can be generally used to give lower bounds for bin packing algorithms based on the method of packing patterns. We extended the LP method to the so-called branching lists and of course we hope that this type of construction can be applied to other problems as well.

5. REFERENCES

[1] J. Balogh, J. Békési, and G. Galambos. New lower bounds for certain bin packing algorithms. In *WAOA 2010, LNCS 6534*, 25–36, 2011.

[2] J. Balogh, J. Békési, G. Galambos, and M.C. Markót. Improved lower bounds for semi-on-line bin packing problems. *Computing*, 84:139–148, 2009.

[3] J. Balogh, J. Békési, G. Galambos, and G. Reinelt. Lower bound for bin packing problem with restricted repacking. *SIAM Journal on Computing*, 38:398–410, 2008.

[4] J. Balogh, J. Békési, G. Galambos, and G. Reinelt. On a multidimensional semi-online bin packing problem. In *Proceedings for ICAI 2010 - 8th International Conference on Applied Informatics, in print*, 2011.

[5] J. Balogh, J. Békési, G. Galambos, and G. Reinelt. On-line bin packing with restricted repacking. *manuscipt*, 2011.

[6] E.G. Coffman, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: Combinatorial analysis. In *Handbook of Combinatorial Optimization Supplement Volume*, Kluwer Academic Publishers, 151–208, 1999.

[7] E.G. Coffmann, M.R. Garey, and D.S. Johnson. Dynamic bin packing. *SIAM Journal on Computing*, 12:227–260, 1983.

[8] D. Coppersmith and P. Raghavan. Multidimensional on-line bin packing: Algorithms and worst-case analysis. *Operations Research Letters*, 8:17–20, 1989.

[9] J. Csirik, G. Galambos, and Gy. Turán. Some results on bin packing. In *Proceedings of EURO VI.*, Vienna, 1983.

[10] J. Csirik and G.J. Woeginger. On-line packing and covering problems. In: *On-line algorithms. Lecture Notes in Computer Science*, Vol. 1442, Berlin, 147–177, 1998.

[11] G. Dósa. The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is $FFD(I) <= 11/9\ OPT(I) +6/9$. In *Proc. ESCAPE*, 1–11, 2007.

[12] L. Epstein and A. Levin. A robust APTAS for the classical bin packing problem. *Mathematical Programming*, 119:33–49, 2009.

[13] L. Epstein and A. Levin. On bin packing with conflicts. *SIAM Journal on Optimization*, 19:1270–1298, 2008.

[14] W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1:349–355, 1981.

[15] G. Galambos. *A new heurisic for the classical bin packing problem.* Technical Report 82. Institute für Mathematik, Augsburg, 1985.

[16] G. Galambos and A van Vliet. Lower bounds for 1-, 2- and 3-dimensional on-line bin packing algorithms. *Computing*, 52:281–297, 1994.

[17] G. Galambos and G.J. Woeginger. Repacking helps in bounded space on-line bin packing, *Computing*, 49:329–338, 1993.

[18] G. Gambosi, A. Postiglione, and M. Talamo. New algorithms for on-line bin packing. In *Algorithms and Complexity, Proceedings of the First Italian Conference*, World Scientific, Singapore, 44–59, 1990.

[19] G. Gambosi, A. Postiglione, and M. Talamo. Algorithms for the relaxed on-line bin-packing model. *SIAM Journal on Computing*, 30:1532–1551, 2000.

[20] M.R. Garey and D.S. Johnson. *Computers and Intractability (A Guide to the theory of NP-Completeness)*, W.H. Freeman and Company, San Francisco, 1979.

[21] E.F. Grove. On-line bin packing with lookahead. In *Proc. SODA*, 430–436, 1995.

[22] G. Gutin, T. Jensen, and A. Yeo. Batched bin packing. *Discrete Optimization*, 2:71–82, 2005.

[23] G. Gutin, T. Jensen, and A. Yeo. Optimal on-line bin packing with two item sizes. *Algorithmic Operations Research*, 1:72–78, 2006.

[24] Z. Ivkovič and E.L. Lloyd. A fundamental restriction on fully dynamic maintenance of bin packing. *Information Processing Letters*, 59:229–232, 1996.

[25] Z. Ivkovič and E.L. Lloyd. Fully dynamic algorithms for bin packing: being (mostly) myopic helps. *SIAM Journal on Computing*, 28:574–611, 1998.

[26] Z. Ivkovič and E. Lloyd. Partially dynamic bin packing can be solved within $1 + \varepsilon$ in (amortized) polylogarithmic time. *Information Processing Letters*, 63:45–50, 1997.

[27] D.S. Johnson. *Near-optimal bin packing algorithms.* PhD thesis. MIT, Cambridge, MA, 1973.

[28] N. Karmarkar and R. Karp. An efficient approximation scheme for the one-dimensional bin packing problem. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS'82)*, 312–320, 1982.

[29] C.C. and D.T. Lee. A simple on-line bin packing algorithm. *Journal of the ACM*, 32:562–572, 1985.

[30] M.C. Markót and T. Csendes. A new verified optimization technique for the "packing circles in a unit square" problems. *SIAM Journal on Optimization*, 16:193–219, 2005.

[31] M.C. Markót, T. Csendes, and A.E. Csallner. Multisection in interval branch-and-bound methods for global optimization. II. Numerical tests, *Journal of Global Optimization*, 16:219–228, 2000.

[32] P. Ramanan, D.J. Brown, C.C. Lee, and D.T. Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10:305–326, 1989.

[33] M.B. Richey. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics*, 34:203–227, 1991.

[34] S.S. Seiden. On the on-line bin packing problem. *Journal of the ACM*, 49:640–671, 2002.

[35] A. van Vliet. An improved lower bound for on-line bin packing algorithms. *Information Processing Letters*, 43:277–284, 1992.

Determining the expected runtime of an exact graph coloring algorithm

Zoltán Ádám Mann
Budapest University of Technology and Economics
Department of Computer Science and Information Theory
Magyar tudósok körútja 2., 1117 Budapest, Hungary
zoltan.mann@cs.bme.hu

Anikó Szajkó
Budapest University of Technology and Economics
Department of Computer Science and Information Theory
Magyar tudósok körútja 2., 1117 Budapest, Hungary
szajko.aniko@gmail.com

ABSTRACT

Exact algorithms for graph coloring tend to have high variance in their runtime, posing a significant obstacle to their practical application. The problem could be mitigated by appropriate prediction of the runtime. For this purpose, we devise an algorithm to efficiently compute the expected runtime of an exact graph coloring algorithm as a function of the parameters of the problem instance: the graph's size, edge density, and the number of available colors. Specifically, we investigate the complexity of a typical backtracking algorithm for coloring random graphs with k colors. Using the expected size of the search tree as the measure of complexity, we devise a polynomial-time algorithm for predicting algorithm complexity depending on the parameters of the problem instance. Our method also delivers the expected number of solutions (i.e., number of valid colorings) of the given problem instance, which can help us decide whether the given problem instance is likely to be feasible or not. Based on our algorithm, we also show in accordance with previous results that increasing the number of vertices of the graph does not increase the complexity beyond some complexity limit. However, this complexity limit grows rapidly when the number of colors increases.

1. INTRODUCTION AND PREVIOUS WORK

Graph coloring[1] is one of the most fundamental problems in algorithmic graph theory, with many practical applications, such as register allocation, frequency assignment, pattern matching, and scheduling [22, 6, 19]. Unfortunately, graph coloring is NP-complete [11]. Moreover, if $P \neq NP$, then no polynomial-time approximation algorithm with an approximation factor smaller than 2 can exist for graph coloring [10].

[1]See Section 2 for detailed definitions.

Exact graph coloring algorithms are often variants of the usual backtrack algorithm. The backtrack algorithm has the advantage that, by pruning large parts of the search tree, it can be significantly more efficient than checking the whole search space exhaustively. In the worst case, the backtrack algorithm requires an exponential number of steps, but its average-case complexity is $O(1)$ [27]. The runtime can vary significantly: both very short and very long runs have non-negligible probability [16].

The probabilistic analysis of the coloring of random graphs was first suggested in the seminal paper of Erdős and Rényi [9]. Subsequent work of Grimmett and McDiarmid [13], Bollobás [4], and Luczak [17], lead to an understanding of the order of magnitude of the expected chromatic number of random graphs. Through the recent work of Shamir and Spencer [24], Luczak [18], Alon and Krivelevich [2], and Achlioptas and Naor [1], we can determine almost exactly the expected chromatic number of a random graph in the limit: with probability tending to 1 when the size of the graph tends to infinity, the expected chromatic number of a random graph is one of two possible values.

In terms of the running time of the backtracking algorithm on random graphs, much less is known. Bender and Wilf gave lower and upper bounds on the runtime of backtracking in the non-k-colorable case [3]. Asymptotically, these bounds are quite good, but in practical cases, there can be several orders of magnitude difference between the lower and upper bounds. In a recent paper, we improved these bounds [20], but there still exists a range of the input parameters, in which there is a non-negligible gap between the lower and upper bounds (see Figure 1, and note also the exponential scale on the vertical axis). Hence, accurate prediction of the algorithm's runtime is still only partially possible.

Predicting the runtime of the algorithm would greatly improve its practical usability, by informing the user in advance about the estimated runtime. This would let the user decide if the exact solution of the problem is realistic in the available time frame, or a heuristic solution should be used instead. More generally, it allows the manual or automated selection of the most suitable algorithm from an algorithm portfolio [12]. It also enhances load balancing when

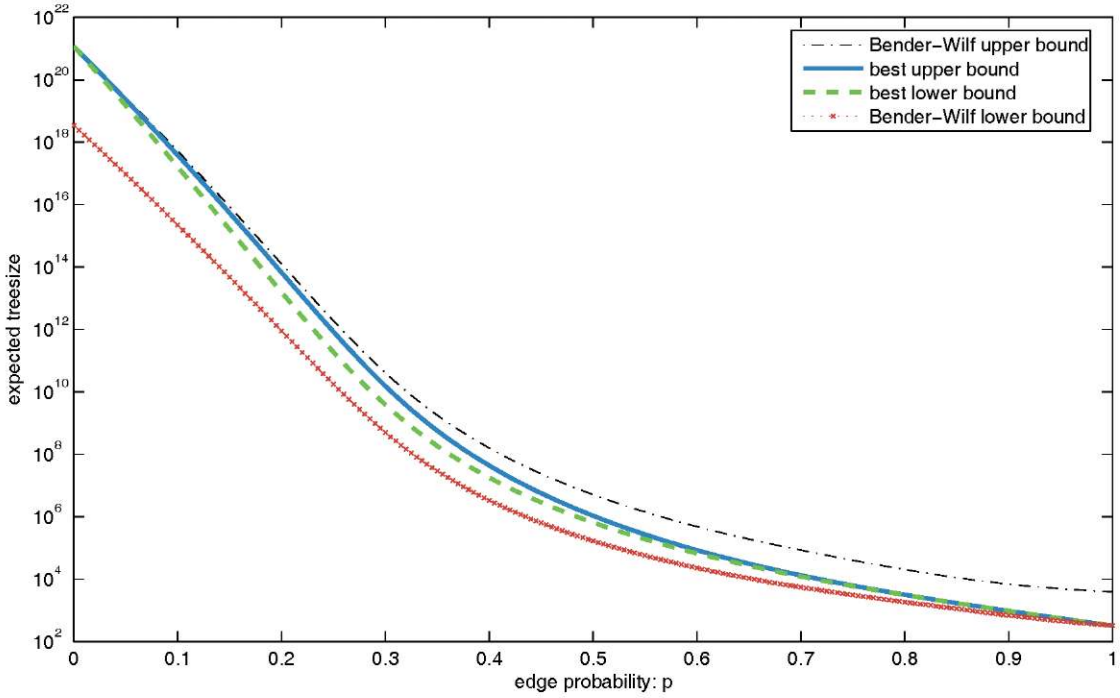


Figure 1: Lower and upper bounds on algorithm runtime of Bender and Wilf [3] and the best known bounds from our recent paper [20]

several problem instances are solved in parallel on multiple machines.

Empirical study of the behaviour of search algorithms and the complexity of graph coloring problem instances has lead to the discovery of a phase transition phenomenon with an accompanying easy-hard-easy pattern [7, 15, 14, 8]. Briefly, this means that, given k colors, for small values of the edge density (underconstrained case), almost all random graphs are colorable. When the edge density of the graph increases, the ratio of k-colorable graphs abruptly drops from almost 1 to almost 0 (phase transition). After this critical region, almost all graphs are non-k-colorable (overconstrained case). In the underconstrained case, coloring is easy: even the simplest heuristics usually find a proper coloring [26, 5]. In the overconstrained case, it is easy for backtracking algorithms to prove uncolorability because they quickly reach contradiction [23]. The hardest instances lie in the critical region [7]. This phenomenon is exemplified in Figure 2, showing our own empirical findings, experimenting with a backtrack graph coloring algorithm [21].

Summarizing these results, one can state that we have a good *quantitative* understanding of graph coloring *in the limit* (when the size of the graph tends to infinity) and a good *qualitative* understanding of it in the finite case. Our aim in this paper is to study the hardness of graph coloring *quantitatively* with accurate results for *finite* graphs.

Hence, our aim is to devise an algorithm for obtaining accurate results on the expected runtime of the backtrack algorithm in coloring random graphs. Like [3] and [20], we restrict ourselves to the non-k-colorable case (see Section 2). More specifically, our complexity results are accurate in the non-k-colorable case, but only an upper bound in the k-colorable case. To use a machine independent measure of algorithm complexity, we analyze the expected size of the search tree as a function of problem instance paramteres: the size of the graph, the edge density and the number of available colors (Section 3). Our contribution is an algorithm for determining the expected size of the search tree exactly (Section 4). The algorithm uses dynamic programming, and its runtime is polynomial in the size of the graph. As a by-product, we also obtain the exact value of the expected number of solutions as a function of input parameters (Section 5). We also present our empirical findings on how the complexity of the problem and the number of solutions depend on the input parameters (Section 6). Finally, Section 7 concludes the paper.

2. PRELIMINARIES

We consider the decision version of the graph coloring problem, in which the input consists of an undirected graph $G = (V, E)$ and a number k, and the task is to decide whether the vertices of G can be colored with k colors such that adjacent vertices are not assigned the same color. The input graph is a random graph from $G_{n,p}$, i.e. it has n vertices and each pair of vertices is connected by an edge with probability p independently from each other. The vertices of the graph will be denoted by $v_1, \ldots, v_n$, the colors by

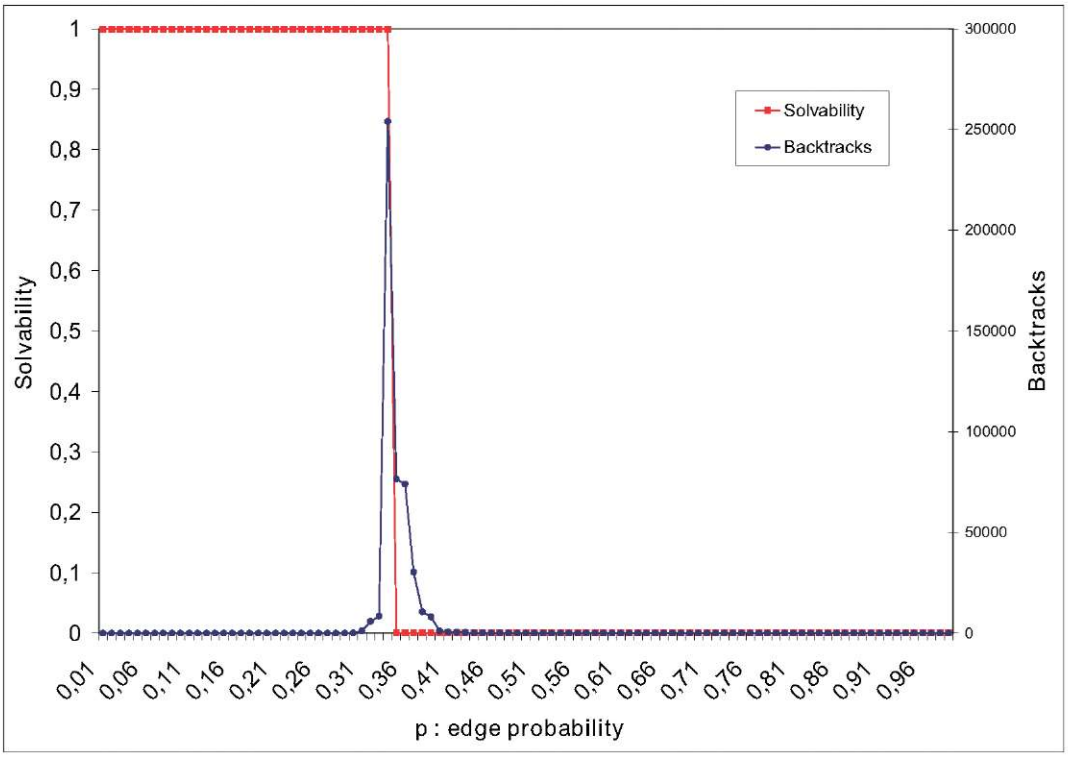


Figure 2: The runtime complexity of a backtrack algorithm (right y-axis: number of backtracks) and the ratio of feasible problem instances (left y-axis: solvability) in the coloring of $G_{n,p}$ random graphs for $n = 70$, as a function of the edge probability (p). The number of colors is $k = 8$.

$1, \ldots, k$. A *coloring* assigns a color to each vertex; a *partial coloring* assigns a color to some of the vertices. A (partial) coloring is *invalid* if there is a pair of adjacent vertices with the same color, otherwise the (partial) coloring is *valid*.

The backtrack algorithm considers partial colorings. It starts with the empty partial coloring, in which no vertex has color. This is the root – that is, the single node on level 0 – of the search tree[2]. Level t of the search tree contains the k^t possible partial colorings of $v_1, \ldots, v_t$. The search tree, denoted by T, has $n + 1$ levels $(0, 1, \ldots, n)$, the last level containing the k^n colorings of the graph. For simplicity of notation, we use $w \in T$ to denote that the partial coloring w is a node of the search tree. Furthermore, let T_t denote the set of partial colorings on level t of T. If $t < n$ and $w \in T_t$, then w has k children in the search tree: those partial colorings of $v_1, \ldots, v_{t+1}$ that assign to the first t vertices the same colors as w.

A node $w \in T_t$ is a partial coloring, i.e. it can also be regarded as a function $w : \{v_1, v_2, \ldots v_t\} \to \{1, 2, \ldots, k\}$. That is, for $w \in T_t$ and $v \in \{v_1, v_2, \ldots v_t\}$, $w(v)$ denotes the color of vertex v in the partial coloring w. In other cases, i.e. when $t < i \leq n$, then $w(v_i)$ is undefined as w assigns no color to v_i.

[2]In order to avoid misunderstandings, we use the term 'vertex' in the case of the input graph and the term 'node' in the case of the search tree.

In each partial coloring w, the backtrack algorithm considers the children of w and visits only those that are valid. Invalid children are not visited, and this way, the whole subtree under an invalid child of the current node is pruned. This is correct because all nodes in such a subtree are also certainly invalid.

T depends only on n and k, not on the specific input graph. However, the algorithm visits only a subset of the nodes of T, depending on which vertices of G are actually connected. The number of actually visited nodes of T will be used to measure the complexity of the algorithm on the given problem instance. Moreover, the number of actually visited nodes on the nth level of T yields the number of solutions.

Of course, this is a simplified algorithm model. In practice, a backtracking graph coloring algorithm can be enhanced with several techniques, e.g. heuristics for the choice of the next vertex to color and the order in which the colors should be considered, symmetry breaking, consistency propagation etc. [25]. Nevertheless, this simplified model captures well the main phenomena of any backtracking-style algorithm (i.e., branching as well as pruning invalid subtrees of the search tree), especially in the non-k-colorable case. This is because in the non-k-colorable case, the search space to be traversed is to a large extent given, and the algorithm must traverse all of it (except for the pruned subtrees of the search tree). In the k-colorable case, the ideas mentioned above for speeding up the algorithm can be leveraged more intensively.

E.g., with lucky choices of the vertices to color and the colors to assign to them, the algorithm might manage to color the graph with a very small amount of – or even zero – backtracking. In contrast, in the non-k-colorable case, the order in which the colors to assign to a given vertex are tried does not matter because all of them have to be tried anyway. Therefore, our model is realistic in the non-k-colorable case; in the k-colorable case, our complexity results can be seen as upper bounds on real algorithm complexity.

3. THE EXPECTED NUMBER OF VISITED NODES OF THE SEARCH TREE

For each $w \in T$, we define the following random variable (the value of which depends on the choice of G):

$$Y_w = \begin{cases} 1 & \text{if } w \text{ is valid,} \\ 0 & \text{else.} \end{cases}$$

Let $p_w = Pr(Y_w = 1)$. Moreover, we define one more random variable (whose value also depends on the choice of G): $Y =$ the number of visited nodes of T.

Since the algorithm visits exactly the valid partial colorings, it follows that $Y = \sum_{w \in T} Y_w$, and thus $E(Y) = \sum_{w \in T} E(Y_w)$. Moreover, it is clear that $E(Y_w) = p_w$. It follows that the expected number of visited nodes in T is:

$$E(Y) = \sum_{w \in T} p_w.$$

For $w \in T_t$, let

$$Q(w) := \big\{\{x, y\} : x, y \in \{v_1, \ldots, v_t\}, x \neq y, w(x) = w(y)\big\}$$

be the set of pairs of vertices with identical colors, and let $q(w) := |Q(w)|$. Clearly, w is valid if and only if, for all $\{x, y\} \in Q(w)$, x and y are not adjacent. It follows that $p_w = (1-p)^{q(w)}$ and thus the expected number of visited nodes of T is:

$$E(Y) = \sum_{w \in T} (1-p)^{q(w)}.$$

Note that computing $E(Y)$ directly through this formula is not tractable since $|T|$ is exponentially large in n. In the following, we devise a way to overcome this hurdle by a smart grouping of the terms of this sum.

4. EFFICIENT CALCULATION USING DYNAMIC PROGRAMMING

Before presenting our algorithm, we need to introduce some further notions. Our first aim is to compute the maximum possible value of $q(w)$ within T_t.

We denote by $s(w, i)$ (or simply s_i if it is clear which partial coloring is considered) the number of vertices of G that are assigned color i in the partial coloring w.

PROPOSITION 1. *For all $w \in T_t$, $q(w) \leq \binom{t}{2}$.*

PROOF.

$$\begin{aligned} q(w) &= \sum_{i=1}^{k} \binom{s_i}{2} = \frac{1}{2}\left(\sum_{i=1}^{k} s_i^2 - \sum_{i=1}^{k} s_i\right) \leq \\ &\leq \frac{1}{2}\left(\left(\sum_{i=1}^{k} s_i\right)^2 - \sum_{i=1}^{k} s_i\right) = \frac{1}{2}\left(t^2 - t\right) = \binom{t}{2}. \end{aligned}$$

□

We will denote this value as $q_{max}(t)$ or simply q_{max}. It is also possible to derive a formula for the minimum of $q(w)$ [20], depending on the value of t and k. This value will be denoted by $q_{min}(t, k)$ or simply q_{min}. The exact formula for $q_{min}(t, k)$ is not necessary for our purposes.

Let $R(q, t, k) := |\{w \in T_t : q(w) = q\}|$ denote the frequency of value q among the $q(w)$ values of the nodes in T_t, given k colors. (The right-hand-side of the definition of $R(q, t, k)$ does not seem to depend on k. However, w inherently depends on k.)

In the sum $\sum_{w \in T_t} (1-p)^{q(w)}$, we can group the terms according to the q values. Since $R(q, t, k)$ is the frequency of the value q among the $q(w)$ values of nodes in T_l, we obtain

$$\sum_{w \in T_t} (1-p)^{q(w)} = \sum_{q=q_{min}(t)}^{q_{max}(t)} R(q, t, k)(1-p)^q.$$

Therefore,

$$E(Y) = \sum_{w \in T} (1-p)^{q(w)} = \sum_{t=0}^{n} \sum_{q=q_{min}(t)}^{q_{max}(t)} R(q, t, k)(1-p)^q.$$

If we could determine all the $R(q, t, k)$ values explicitly, this would enable us to efficiently calculate the exact value of $E(Y)$ using this formula. Determining the $R(q, t, k)$ values is possible with the following recursion (we write ℓ instead of k as third parameter, so that the meaning of k is not affected):

PROPOSITION 2.

$$R(q, t, \ell) = \sum_{j=0}^{t} \binom{t}{j} R\left(q - \binom{j}{2}, t - j, \ell - 1\right).$$

PROOF. Assume that color class 1 contains j vertices. There are $\binom{t}{j}$ possibilities to choose these j vertices. The remaining $t - j$ vertices must be colored with $\ell - 1$ colors. Moreover, the j vertices of color 1 already account for $\binom{j}{2}$ vertex pairs with identical colors. Hence, the remaining $t - j$ vertices must be colored in such a way that the number of vertex pairs with identical colors out of these $t - j$ vertices equals $q - \binom{j}{2}$. For this, there are exactly $R\left(q - \binom{j}{2}, t - j, \ell - 1\right)$ possibilities. □

Based on this recursive formula, we can use dynamic programming to compute the $R(q, t, \ell)$ values and store them in

Algorithm 1 Dynamic programming algorithm to compute $E(Y)$

```
//Set R values for ℓ = 1
for t=0 to n
{
    R(C(t,2), t, 1) = 1
}

//Set R for higher values of ℓ
for ℓ=2 to k
{
    for t=0 to n
    {
        for q=q_min to q_max
        {
            //Use the recursive formula to compute R
            R(q,t,ℓ) = 0
            for j=0 to t
            {
                //Consider the current term only if non-zero
                if q − C(j,2) ≥ q_min(t − j, ℓ − 1)
                {
                    term = C(t,j) R(q − C(j,2), t − j, ℓ − 1)
                    R(q,t,ℓ) = R(q,t,ℓ) + term
                }
            }
        }
    }
}

//Compute E(Y)
result=0
for t=0 to n
{
    for q=q_min to q_max
    {
        result=result+R(q,t,k)(1 − p)^q
    }
}
E(Y)=result
```

a 3-dimensional table. We fill this table according to increasing values of ℓ. This works because computing $R(q,t,\ell)$ requires only already computed values of the form $R(q',t',\ell-1)$. For a given ℓ, we must iterate through the possible values of t from 0 to n, and for each such t, we must fill the table for all possible values of q from q_{min} to q_{max}. See Algorithm 1 for details.

As a starting point, if $\ell = 1$, then for all values of t, T_t consists of a single partial coloring in which all vertices are assigned the same single color. Therefore, if $\ell = 1$, then $q_{min} = q_{max} = \binom{t}{2}$ and for this value of q we have $R(q,t,1) = 1$. As additional boundary conditions, we have $R(q,t,\ell) = 0$ in all cases when $t < 0$ or $q < q_{min}$.

Since $t = O(n)$, $j = O(n)$, $q_{max} = O(n^2)$, and $\ell = O(k)$, the runtime of Algorithm 1 is $O(kn^4)$. This is polynomial in the size of the graph, though quite high. On the other hand, the calculation of the $R(q,t,\ell)$ values is the most time-consuming part of the algorithm, and these values can be pre-computed and stored. Afterwards, we can compute $E(Y)$ more quickly – namely in $O(n^3)$ steps – for different values of n, p, k.

5. THE EXPECTED VALUE OF THE NUMBER OF SOLUTIONS

As a by-product of the presented model for algorithm performance, we also obtain results on the expected number of solutions. This is because the number of solutions is exactly $S = \sum_{w \in T_n} Y_w$, and thus the expected number of solutions is $E(S) = \sum_{w \in T_n} (1-p)^{q(w)}$. As previously, this sum has exponentially many terms, but again, the terms can be grouped according to the value q among the $q(w)$ values. With the notation introduced previously, we can write it as

$$E(S) = \sum_{q=q_{min}}^{q_{max}} R(q,n,k)(1-p)^q.$$

Thus we can compute $E(S)$ with a slight modification of Algorithm 1 in $O(kn^4)$ time. If the $R(q,t,k)$ values are already pre-computed, then computing $E(S)$ takes only $O(n^2)$ time, since the above formula for $E(S)$ has less than q_{max} terms and $q_{max} = O(n^2)$.

Recalling that our runtime prediction is accurate for non-k-colorable graphs only, the expected number of solutions can help us to decide, in what range of the parameters our runtime prediction is accurate. If the expected number of solutions is very small, then probably there is no solution and hence our runtime estimation is accurate, whereas if the expected number of solutions is high, then probably the graph is k-colorable, and our runtime prediction is only an upper bound on the real value. More precisely, knowing the expected number of solutions allows us to estimate the probability that a problem instance is solvable using Markov's inequality:

$$Pr(\text{solvable}) = Pr(S \geq 1) \leq E(S).$$

What is more, the probability of solvability can also be bounded from below using the first and second moments of S [1]. Practically, if the problem instance parameters are such that $E(S)$ is significantly less than 1, then such problem instances are probably unsolvable. If, on the other hand, $E(S)$ is significantly above 1, then the problem instances are probably solvable. That is, the phase transition will be near the point where $E(S) \approx 1$.

6. NUMERICAL RESULTS

This section shows some simulation results based on the presented method.

6.1 Size of the search tree

The method presented in Section 4 enables us to gain some insight as to how the complexity of graph coloring changes for different values of the parameters n, k, and p. Figure 3 shows an example: $E(Y)$ as a function of n and k, for fixed p. We can conclude from the figure that for small values of k, the problem is easy, even if n becomes large. This is consistent with previous results on the relatively low average-case complexity of graph coloring [27, 26]: although the complexity is exponential in n in the worst case, but it is $O(1)$

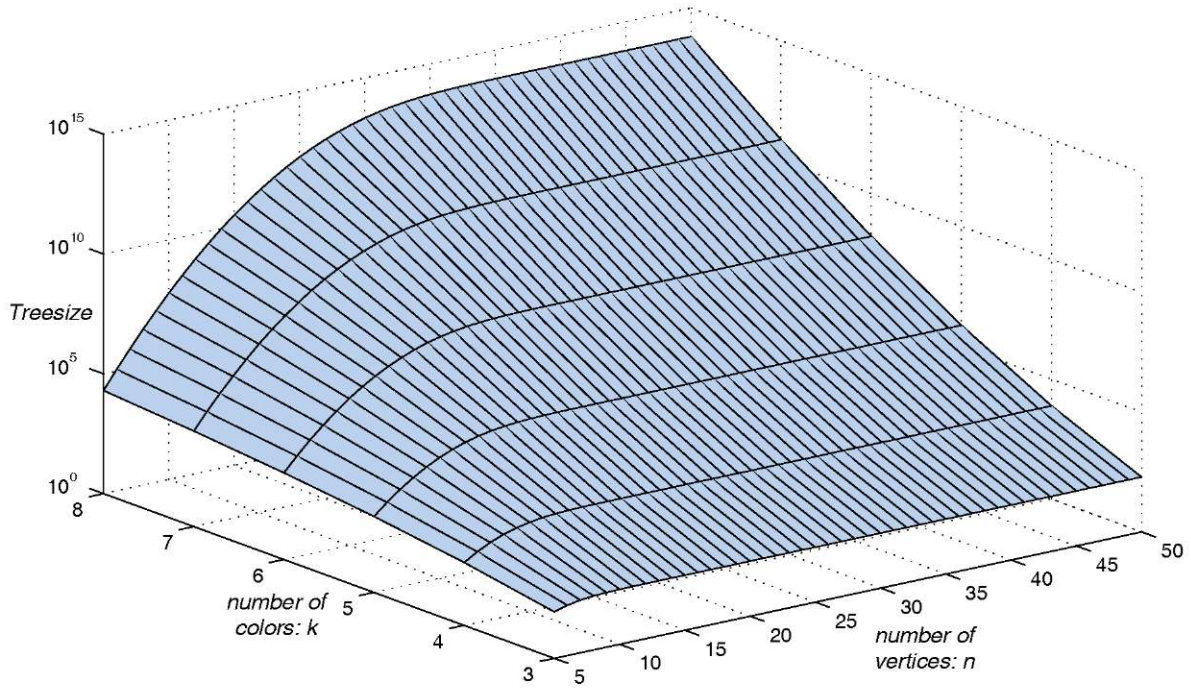


Figure 3: Expected size of the search tree for $p = 0.5$, as a function of n and k.

in the average case. However, as k increases, this increases the complexity of the problem dramatically (note the exponential scale on the vertical axis). It is still true that the complexity saturates, i.e. increasing n does not increase the complexity significantly after some threshold. However, this saturation takes place at a much higher value than in the case of small k.

The same phenomenon is depicted in Figure 4 from a different perspective. Here, n is fix, and p and k are varied. Again, it can be seen from the figure, that the complexity is in many cases quite low and hardly increasing with growing k. However, there is again a range of the parameters in which the complexity explodes. This is in line with the practical experience of high variability in the runtime of the algorithm. It is also clear that the curve must be monotonously decreasing in p: this is because in the non-k-colorable case, where our algorithm model is accurate, increasing p makes it easier for the algorithm to prove uncolorability, as more edges are likely to make the contradiction apparent earlier on (at a higher level of the search tree).

6.2 Number of solutions

Using the method presented in Section 5, we can also look at the expected number of solutions, and thus the picture can be further refined. Figure 5 depicts the expected number of solutions together with the expected size of the search tree for fix n and k, as a function of p. Since the complexity is exactly the number of all valid partial colorings in the search tree, and the number of solutions is the number of valid colorings on the nth level of the search tree, the figure shows clearly the changing contribution of the nth level of the search tree to the total search tree size. As can be seen, for small values of p, the search tree is dominated by the nth level.

However, as p increases, the contribution of the nth level decreases rapidly. This is again a consequence of the fact that the increased number of edges let the algorithm detect inconsistencies earlier on, thus it becomes rare that the algorithm actually reaches the nth level.

As mentioned earlier, our results are only accurate for non-k-colorable graphs, and the transition between k-colorability and non-k-colorability occurs roughly where the expected number of solutions is 1. Figure 5 also shows where this happens: for the used parameters, it is around $p \approx 0.5$. Hence, for the given values of n and k, our results are accurate in the $p > 0.5$ region. In other words, the curve in Figure 5 that shows the expected size of the search tree is accurate only in the right half of the diagram; in the left half, is is only an upper bound on the algorithm's runtime complextiy.

Finally, Figure 6 depicts the expected number of solutions together with the expected size of the search tree for fix p and k, as a function of n. As can be seen, for small values of n (in the range of $2k \ldots 3k$), the expected number of solutions is relatively high, i.e. the contribution of the nth

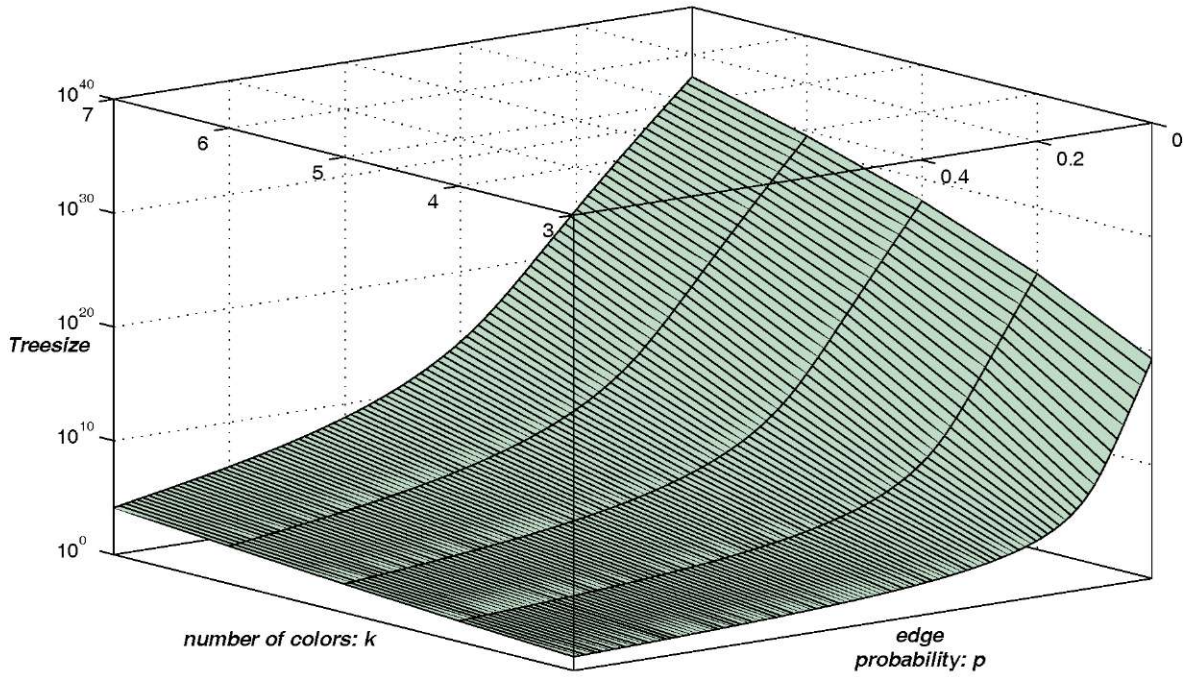


Figure 4: Expected size of the search tree for $n = 40$, as a function of p and k.

level of the search tree is high. In this region, increasing n significantly increases the algorithm's runtime. This is logical, because if n increases, then the algorithm will also visit many nodes on level $n+1$ of the search tree. However, after a while, the already mentioned saturation takes place. The expected number of solutions becomes very small, indicating that the algorithm rarely gets to the nth level of the search tree, as it usually finds a contradiction much earlier, without descending that far in the search tree. Accordingly, further increasing n does not significantly increase the complexity anymore, because the algorithm visits the lower parts of the search tree rarely anyway. This phenomenon reveals an interesting and quite complex connection between the algorithm's runtime complexity and the number of solutions.

7. CONCLUSION AND FUTURE WORK

In this paper, we have investigated the runtime complexity of a typical backtracking algorithm for coloring random graphs of the class $G_{n,p}$ with k colors. Using the expected size of the search tree as the measure of complexity, we devised a polynomial-time algorithm for predicting the backtrack algorithm's runtime complexity. As a by-product, our method also delivers the expected number of solutions of the given problem instance, which is interesting in its own right, but also helps to quantify when our model of runtime complexity is accurate.

Using the developed methods, we analyzed numerically how the algorithm's runtime complexity depends on the input parameters n, p, and k. We obtained a rich picture with regions of very low and very high complexity, and varying sensitivity with respect to changes in the input parameters. This way, our model can explain several of the phenomena that had been discovered before about the behaviour of backtrack-style optimization algorithms on graph coloring and related problems.

We also showed the multifaceted connection between the expected complexity of the problem and the expected number of solutions.

The most important limitation of the approach presented in this paper is that it is only accurate for non-k-colorable problem instances. Our future work will focus on extending the presented results to k-colorable problem instances. The main challenge of this is to take into account the order in which the algorithm visits the children of a node of the search tree, because this can have significant impact on the algorithm's running time. This difference might make it necessary to use different and/or more sophisticated methods to derive similar results for the case of k-colorable graphs as well.

Acknowledgements

This work was partially supported by the Hungarian National Research Fund and the National Office for Research and Technology (Grant Nr. OTKA 67651).

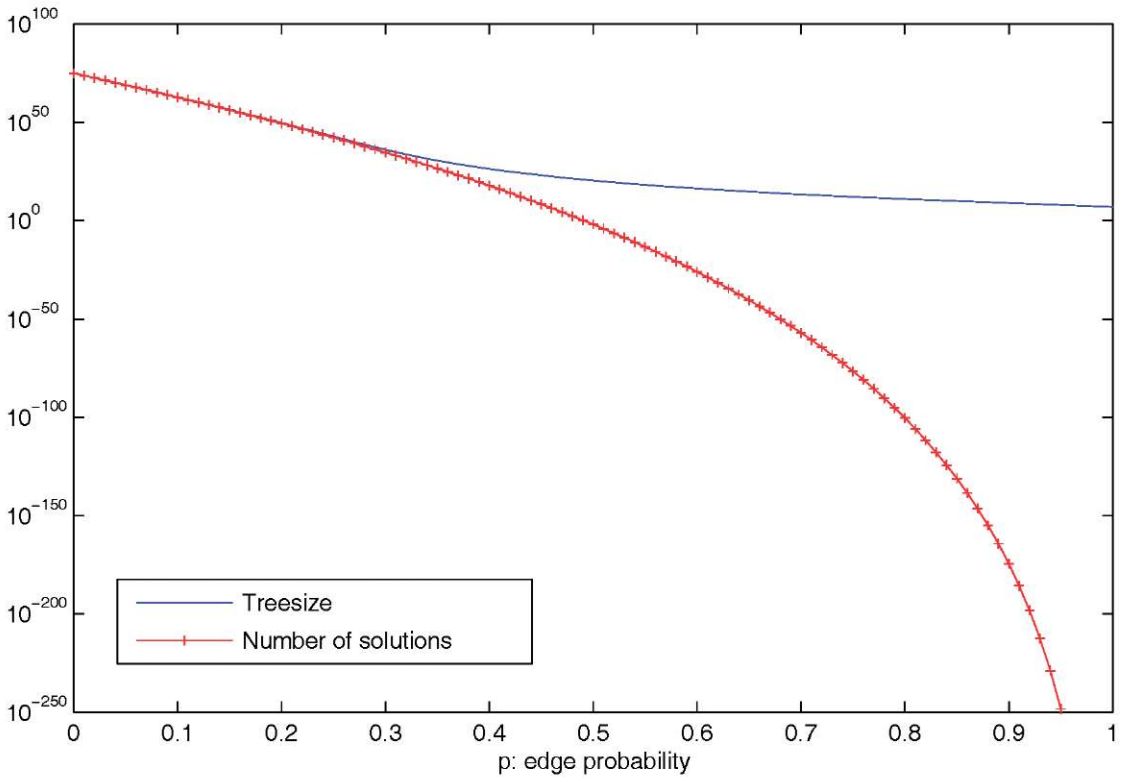


Figure 5: Expected number of solutions and expected search tree size for $n = 75$ and $k = 10$, as a function of p.

8. REFERENCES

[1] D. Achlioptas and A. Naor. The two possible values of the chromatic number of a random graph. In *36th ACM Symposium on Theory of Computing (STOC '04)*, pages 587–593, 2004.

[2] N. Alon and M. Krivelevich. The concentration of the chromatic number of random graphs. *Combinatorica*, 17(3):303–313, 1997.

[3] E. A. Bender and H. S. Wilf. A theoretical analysis of backtracking in the graph coloring problem. *Journal of Algorithms*, 6(2):275–282, 1985.

[4] B. Bollobás. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.

[5] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[6] P. Briggs, K. D. Cooper, and L. Torczon. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems*, 16(3):428–455, 1994.

[7] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *12th International Joint Conference on Artificial Intelligence (IJCAI '91)*, pages 331–337, 1991.

[8] J. Culberson and I. Gent. Frozen development in graph coloring. *Theoretical Computer Science*, 265(1-2):227–264, 2001.

[9] P. Erdős and A. Rényi. On the evolution of random graphs. *Magyar Tud. Akad. Mat. Kutató Int. Közl*, 5:17–61, 1960.

[10] M. R. Garey and D. S. Johnson. The complexity of near-optimal graph coloring. *Journal of the ACM*, 23:43–49, 1976.

[11] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[12] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[13] G. R. Grimmett and C. J. H. McDiarmid. On colouring random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 77(2):313–324, 1975.

[14] T. Hogg. Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81(1-2):127 – 154, 1996.

[15] T. Hogg and C. P. Williams. The hardest constraint problems: A double phase transition. *Artificial Intelligence*, 69(1-2):359–377, 1994.

[16] H. Jia and C. Moore. How much backtracking does it take to color random graphs? rigorous results on heavy tails. In *Principles and Practice of Constraint Programming (CP 2004)*, pages 742–746, 2004.

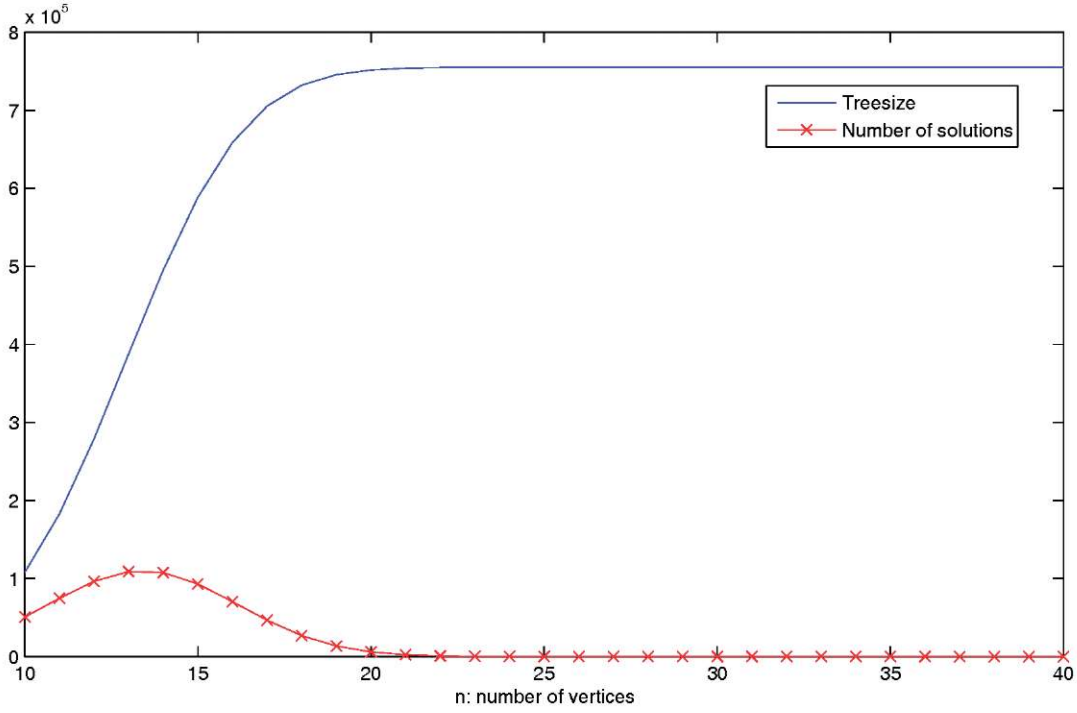


Figure 6: Expected number of solutions and expected search tree size for $p = 0.5$ and $k = 5$, as a function of n.

[17] T. Luczak. The chromatic number of random graphs. *Combinatorica*, 11(1):45–54, 1991.

[18] T. Luczak. A note on the sharp concentration of the chromatic number of random graphs. *Combinatorica*, 11(3):295–297, 1991.

[19] Z. Mann and A. Orbán. Optimization problems in system-level synthesis. In *3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, pages 222–231, 2003.

[20] Z. Mann and A. Szajkó. Improved bounds on the complexity of graph coloring. In *12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, 2010.

[21] Z. Mann and T. Szép. BCAT: A framework for analyzing the complexity of algorithms. In *8th IEEE International Symposium on Intelligent Systems and Informatics*, pages 297–302, 2010.

[22] N. K. Mehta. The application of a graph coloring method to an examination scheduling problem. *Interfaces*, 11(5):57–65, 1981.

[23] R. Monasson. On the analysis of backtrack procedures for the coloring of random graphs. In E. Ben-Naim, H. Frauenfelder, and Z. Toroczkai, editors, *Complex Networks*, pages 235–254. Springer, 2004.

[24] E. Shamir and J. Spencer. Sharp concentration of the chromatic number on random graphs $G_{n,p}$. *Combinatorica*, 7(1):121–129, 1987.

[25] T. Szép and Z. Mann. Graph coloring: the more colors, the better? In *11th IEEE International Symposium on Computational Intelligence and Informatics*, 2010.

[26] J. S. Turner. Almost all k-colorable graphs are easy to color. *Journal of Algorithms*, 9(1):63–82, 1988.

[27] H. S. Wilf. Backtrack: an O(1) expected time algorithm for the graph coloring problem. *Information Processing Letters*, 18:119–121, 1984.

Speeding up exact cover algorithms by preprocessing and parallel computation

Sándor Szabó
Institute of Mathematics and Informatics
University of Pécs
Ifjúság út 6.
7624 Pécs, Hungary
sszabo7@hotmail.com

ABSTRACT

D. Knuth [2] has proposed an exact cover search algorithm that works well on nontrivial size instances. In this paper we try to find ways to speed up the search. From this reason we will consider preprocessing the initial data of the problem to construct certain tables we can look up during the computation. Again to speed up the exact cover search we will propose two ways to carry out the computations in a parallel fashion. Basically we are interested in practical solutions of a computationally hard problem motivated by algebraic applications from the author's practice.

Keywords

Exact cover, k-cover, maximum clique, k-clique problems, dancing links implementation, speed up by preconditioning, practical algorithm for NP complete problem in high performance computing environment,

1. INTRODUCTION

Let U be a ground set and let $A_1, \ldots, A_k$ be subsets of U. If $A_1 \cup \cdots \cup A_k = U$, then we say that the sets $A_1, \ldots, A_k$ form a covering of U. If $A_i \cap A_j = \emptyset$ for each i, j, $i \neq j$, $1 \leq i, j \leq k$, then we say that the sets $A_1, \ldots, A_k$ form a packing of U. If the sets $A_1, \ldots, A_k$ form a covering and a packing of U simultaneously, then we say that they form a partition (or exact cover or tiling) of U. The next toy example illustrates these concepts.

Example 1. Let the ground set U be defined by $U = \{1, \ldots, 7\}$. The members of the family of subsets F of U are the following

$$A_1 = \{2,3,5\}, \quad A_2 = \{1,3,6\}, \quad A_3 = \{1,6\},$$
$$A_4 = \{3,4,5,7\}, \quad A_5 = \{4,7\}, \quad A_6 = \{2,6\}.$$

A routine consideration shows that the sets A_1, A_3, A_5 form a partition of U. The sets A_3, A_4 form a packing of U, that cannot be extended to a partition of U. The sets A_3, A_4, A_6 form a covering of U which does not contain a partition of U.

MATCOS 2011 Koper, Slovenia

PROBLEM 1. *Given a ground set U and a family of subsets F of U. Decide if there are elements $A_1, \ldots, A_k$ of F that form a partition of U.*

This is the decision version of the so-called exact cover (or set partition) problem. The problem is known to be NP complete. At some occasions a simple "yes" or "no" is not enough and one needs an actual exact cover or all possible exact covers. Plainly these versions of the exact cover problem cannot be computationally less demanding than the decision version of the problem.

PROBLEM 2. *Given a ground set U, a family of subsets F of U and a positive integer k. Decide if there are subsets $A_1, \ldots, A_k$ of F that form a partition of U.*

This problem is called the decision version of the k-cover problem. Again one might be interested in exhibiting a k-cover of U or listing all possible k-covers of U. This problem is called the enumeration version of the k-cover problem.

Let Γ be a graph. A subgraph Δ is called a clique in Γ if each two distinct nodes in Δ are adjacent. If Δ has k nodes, then we simply call it a k-clique.

PROBLEM 3. *Given a graph Γ and a positive integer k. Decide if Γ has a k-clique.*

This is the decision version of the k-clique problem. In certain applications one wants to exhibit a k-clique or to list all possible k-cliques of Γ.

There is an industrial strength algorithm for solving Problem 1 described in [2]. This algorithm can be used to solve Problem 2 too. Indeed it was used for instance in [5] and [6]. There is a time tested algorithm for solving Problem 3 proposed in [3]. An improved algorithm is presented in [4]. Another type of improvement is described in [8]. In this paper we will consider Problems 1 and 2. A k-clique problem can be associated with the k-cover problem naturally. In general a graph can be associated with each exact cover problem. We will point out that keeping this graph available during the computation provides an extra flexibility that can be exploited in parallel computations.

2. EXACT COVER

The initial data (U, F) of Problem 1 can be visualized by introducing a bipartite graph Ω. The nodes of Ω are

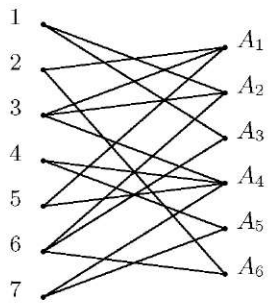


Figure 1: The bipartite graph Ω in Example 1.

Table 1: The incidence matrix I of the graph Ω in Example 1 and the extended incidence matrix of I'.

	1	2	3	4	5	6	7
A_1		•	•		•		
A_2	•		•			•	
A_3	•					•	
A_4			•	•	•		•
A_5				•			•
A_6		•				•	

	1	2	3	4	5	6	7	a	b	c	d
A_1		•	•		•						
A_2	•		•			•		○			
A_3	•					•			○		
A_4			•	•	•		•		○	○	
A_5				•			•	○		○	○
A_6		•				•					○

partitioned into two disjoint sets V_1, V_2. Here $V_1 = U$, $V_2 = F$ and $u \in U$ is adjacent to an $A \in F$ if $u \in A$. In order to construct an algorithm for locating exact covers let us consider a partition (or exact cover) P of U such that $P \subseteq F$. This means that the set of neighbors of the elements of P form a partition of V_1.

Pick an $A \in F$. We distinguish two cases. If $A \notin P$, then let Ω_1 be the graph we get from Ω after deleting A from Ω but do not delete the neighbors of A. If $A \in P$, then let Ω_2 be the graph we get from Ω after deleting A and its neighbors from Ω. In both cases the original instance is reduced to two smaller instances of the problem. In this way one can build up a binary searching tree. Then traversing the tree one can find a solution for the problem.

The branching of the search tree depends on the choice of the subset A of F. This means that one has to specify the choice of A to get an algorithm.

2.1 Knuth's algorithm

The algorithm Knuth [2] suggests builds a non-binary search tree. Let P be a partition of U such that $P \subseteq F$. Pick a $u \in U$. Let $B_1, \ldots, B_r$ be all the neighbors of u in Ω. For each i, $1 \leq i \leq r$ let Ω_i be the graph we get from Ω after deleting the node B_i together with its neighbors. In this way the original instance is reduced to r smaller ones. One then can build a (non-binary) search tree and use it to find a solution of the original instance.

Of course the branching of the tree depends on the particular rule one uses to choose the element u of U. Knuth [2] recommends to choose an $u \in U$ for which r is minimal.

One also has to make a decision on the data structures to represent the data. The initial data (U, F) of Problem 1 can be summarized in an incidence matrix I. The rows of I are labeled by the elements of F and the columns of I are labeled by the elements of U. Thus I is an $|F|$ by $|U|$ matrix consisting of $|F||U|$ cells. Let $A \in F$, $u \in U$. The cell where row A crosses column u will be referred as the (A, u) cell. The entry in the (A, u) cell is one if $u \in A$ and the cell contains zero if $u \notin A$.

We further particularize the initial data structure (U, F) of the algorithm. Let $|F| = m$ and $|U| = n$. In this way I is an m by n matrix with zero and one entries. We may assume that $U = \{1, \ldots, n\}$. Let β_j be the number of ones in the j-th column of the incidence matrix I. We will call the numbers $\beta_1, \ldots, \beta_n$ branching factors. Let β_j be a smallest one among $\beta_1, \ldots, \beta_n$ for which j is minimal.

If $\beta_j = 0$, then $A_1 \cup \cdots \cup A_m \neq U$ and so one cannot form a partition of U using the elements of F.

If $\beta_1 = \cdots = \beta_n = 1$, then the sets $A_1, \ldots, A_m$ form a partition of U.

For the remaining case we may suppose that $\beta_j \neq 0$ and $(\beta_1, \ldots, \beta_n) \neq (1, \ldots, 1)$. Set $r = \beta_j$ and assume that the non-zero entries in the j-th column of I are in the rows indexed by $i(1), \ldots, i(r)$. In the partition P of U we are looking for one member of F must contain the element j of U and consequently exactly one of the sets $A_{i(1)}, \ldots, A_{i(r)}$ must be in P. At this point the search procedure forks into r branches. In the k-th branch the set $A_{i(k)}$ will be a member of P. We replace U by $U \setminus A_{i(k)}$ and from the family F we delete each set that is not disjoint to $A_{i(k)}$. We end up with r smaller instances of the exact cover problem.

Choosing a minimal among the branching factor is a heuristic principle to try to minimize the size of the search tree. In many application the members of the family F has the same number of elements and so the number of the columns of the incidence matrix reduces with the same extent at each branching. In this case one can expect the heuristic principle performing well. However, if there are large differences in the sizes of the members of F, then the greedy choice picking the large members of F first may reduce the size of the incidence matrix I rapidly and reducing the size of the search tree better. We do not pursue this issue in this paper.

2.2 Secondary columns

Problem 1 can be formulated in terms of solving a system of linear equations in zero-one variables. Let P be a partition of U and consider the $x_1, \ldots, x_m$ zero-one variables. Let $x_i = 1$ if $A_i \in P$ and let $x_i = 0$ if $A_i \notin P$. Now $Ax = b$, where $A = I^T$, x is the m-dimensional column vector with entries $x_1, \ldots, x_m$ and b is the m-dimensional column vector with entries $1, \ldots, 1$. Conversely, each zero-one solution of the system of linear equations $Ax = b$ corresponds to a partition P of the ground set U. In [2] Knuth pointed out that after a minor modification his algorithm handles the more general problem of finding all zero-one solutions of the system $Ax = b$, $A_1 x_1 \leq b_1$. Here A_1 is an l by m matrix with zero-one entries, b_1 is an l-dimensional column vector with components $1, \ldots, 1$. In other words we can add certain inequality constraints to the original $Ax = b$ problem.

Set $I_1 = A_1^T$ and using this set $I' = [A^T, A_1^T] = [I, I_1]$. Shortly we added l additional columns to the incidence matrix I. The columns of I are called the primarily columns of I' and the columns of I_1 are called the secondary columns of I'. The columns of incidence matrix I' correspond to the elements of a new ground set $U' = U \cup U_1$. The rows of

the incidence matrix I' correspond to the members of a new family of subsets of F' of U'. We are looking for all subsets P' of F' for which the restrictions of the members of P' to U form a partition of U and the restrictions of the members of P' to U_1 form a packing of U_1. The algorithm for this new generalized situation goes almost in the same way as in the original version. We compute the branching factors only for the primary columns. But we check if two sets in F' are disjoint or not instead of checking if their restriction to U is disjoint or not.

For the sake of easier reference we would like to spell out the problem that Knuth's algorithm solves.

PROBLEM 4. *Given a ground set U and a partition U_1, U_2 of U. Further given a family of subsets F of U. Decide if there is a subset P of F such that $P_{|U_1}$ forms a partition of U_1 and $P_{|U_2}$ forms a packing of U_2.*

Here $P_{|U_i} = \{A \cap U_i : A \in P\}$ is the restriction of P to U_i, for each i, $1 \leq i \leq 2$. In the $U_2 = \emptyset$ particular case the problem reduces to Problem 1. If $U_1 = \emptyset$, then one cannot compute the branching factors. So we should pay attention to handle this particular case correctly in the algorithm. During execution the algorithm generates various subproblems and among these quite naturally subproblems with $U_1 = \emptyset$ may occur.

Our first observation can be summarized in the following manner. Suppose we face Problem 1. Before starting Knuth's algorithm first we systematically inspect the incidence matrix I associated with the problem. We construct secondary columns and we add them to I. In a typical situation the search tree of the modified problem is smaller than the search tree in the original problem as at each step more rows are deleted from the incidence matrix. Consequently the branching factors are smaller. The secondary columns are constructed only once at the beginning of the algorithm. In this sense we use a preprocessing to speed up the search. There are various possibilities to construct secondary columns. For Problem 2 there seem to be more ways than for Problem 1.

Using the terminology of the zero-one linear programming one can say that when we generate secondary columns in a sense we generate certain cuts to slice down pieces from the domain of feasible solutions.

We close this section noticing that Knuth's original algorithm for solving Problem 1 can be extended to solve the following problem.

PROBLEM 5. *Given a ground set U and a partition U_1, U_2, U_3 of U. Further given a family of subsets F of U. Decide if there is a subset P of F such that $P_{|U_1}$ forms a partition of U_1, $P_{|U_2}$ forms a covering of U_2 and $P_{|U_3}$ forms a packing of U_3.*

2.3 Dancing links

Let us consider the cells of the incidence matrix I that filled with ones. A fixed cell C containing one has four neighbors. Two in the same row. One C_L on the left of C. One C_R on the right of C. (If in the way of looking for C_L we would reach the beginning of the row, then we should continue from the end of the row.) Two in the same column. One C_T on the top of C. One C_B below C. (If in the way of looking for C_T we reach the top of the column, then we continue from the bottom of the column.) Then we direct an arrow from C to each of C_L, C_R, C_T, C_B. The result is a four fold linked list. This list represents the incidence matrix I. Geometrically the four fold linked list is a torus. The vertical lines are the longitudes on the torus. The horizontal lines are the latitudes on the torus. The first row and the first column play the role of a kind of coordinate system or frame. Figure 4 depicts the incidence matrix I in Example 1 as an illustration. A horizontal and a vertical portion of the four fold linked list are shown in Figure 2 and 3, respectively

As the computation unfolds one deletes rows and columns from I to get a smaller incidence matrix. When back tracking occurs one has to restore an earlier version of the incidence matrix. The most striking feature of this book keeping machinery is that the links retain enough information to make this restoration possible.

A row operation is the following. Given a row and a fixed entry in this row. We go through the entries of the row moving from left to right starting at the fixed entry. During this trip we link out each entry vertically. The entry at the starting point is treated exceptionally. We do not link out the starting entry.

One can reverse the result of a row operation completely and restore the original situation by going through the entries of the row moving from right to left stating at the fixed entry and link in the entries vertically. We do not link in the starting entry. Let us call this operation a reverse row operation.

A column operation is the following. Given a column and a fixed entry in the column. We go through the entries of the column from up to bottom starting at the fixed entry and carry out a row operation in the row of each entry. The row of the fixed starting entry in the column is treated exceptionally. We do not carry out a row operation in connection with the row of this entry. Also the entry from the horizontal frame in this column is treated exceptionally. We do not carry out a row operation in the row of the frame entry. Instead we link out the frame entry horizontally.

On can completely reverse the result of a column operation by going through the entries of the column from down to up and carrying out a reverse row operation in the row of each entry. Let us call this operation a reverse column operation.

A pivot operation is the following. Given a row. We will call this the pivot row. We go through the entries of the pivot row moving from left to right starting at the frame entry in the row. At each entry we carry out a column operation. The column of the frame entry is treated exceptionally. We do not carry out a column operation in connection with the column of the frame entry. Instead we link out this entry vertically.

One can reverse a pivot operation completely and restore the original conditions by going through the pivot row moving from right to left starting at the frame entry and carry out a reverse column operation in connection with the column of each entry. At the frame entry we do not carry out a reverse row operation instead we link in the entry vertically. But in fact we do not want to restore the original situation completely. After restoring the original conditions we would like to delete the pivot row from the table. This can be achieved by going through the pivot row moving from left to right starting at the frame and linking out each entry vertically. Let us call this operation a reverse pivot operation.

Example 2. Consider the ground set U and family F of

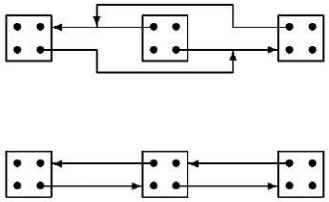


Figure 2: Bottom: A horizontal portion of the double linked list. Top: The middle entry is linked out horizontally.

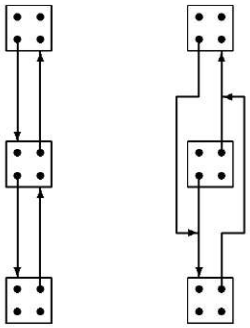


Figure 3: Left: A vertical portion of the double linked list. Right: The middle entry is linked out vertically.

subsets of U given in Example 1. Suppose A_6 is chosen to be a member of the partition P we intend to construct. The new incidence matrix is constructed by deleting the members of F that are not disjoint to A_6. With the row of A_6 we start a pivot operation. First we mark the horizontal section in the 6-th row from the frame to the 1-st element of A_6. Next link out the row of A_6 and mark a vertical section joining to the end point of the earlier marked section. With the column of the 1-st element of A_6 we start a column operation. At the frame link out the column of the 1-st element of A_6 and mark a vertical section joining to the end point of the earlier marked section. Then mark a horizontal section joining to the end point of the earlier marked section to start a row operation. Link out vertically the end point of the marked section. Mark a horizontal section joining the earlier marked section. Link out vertically the end point of the marked section. Continue in this way linking out the elements of A_1 until we reach again the 3-rd column. With this a row operation is completed. Then move down vertically to seek further members of F that are not disjoint to A_6. Repeat the whole procedure in connection with each element of A_6. At the end the path we followed reach back to the point we started. Figures 5 to 7 illustrate a few steps of the procedure.

Knuth [2] illustrates this algorithm by solving various puzzles. But make no mistake. This is a serious algorithm with important applications.

3. PREPROCESSING

Let us consider an exact cover problem with the initial

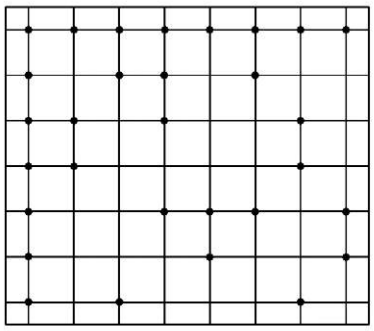


Figure 4: The incidence matrix I in Example 1 represented as a double linked list. The entries are marked as bullets. The single non-oriented edges represent double oriented links.

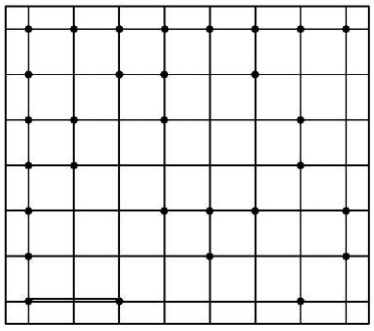


Figure 5: First step of canceling A_6 from the family F in Example 1. We marked a horizontal section in the 6-th row.

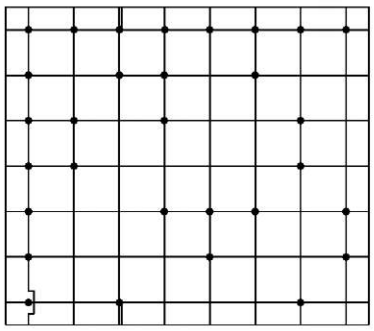


Figure 6: We linked out the row of A_6 and marked a vertical section.

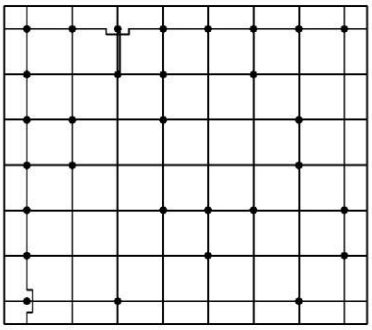


Figure 7: We linked out the column of the first element of A_6 and marked a vertical section.

data (U, F), where $U = \{1, \ldots, n\}$ and $F = \{A_1, \ldots, A_m\}$ is a family of subsets of U. Suppose that $A_i \cap A_j = \emptyset$. Set $U' = U \setminus (A_i \cup A_j)$ and let F' be the family of all the members of F that are disjoint to both A_i and A_j. Suppose that we know from some source that the ground set U' cannot be partitioned into the members of the family F'. This piece of information can be exploited to speed up the search. Namely, we form an m-dimensional column vector a. The i-th and j-th components of a are equal to one and the remaining components of a are equal to zero. We augment the incidence matrix of the original problem with the vector a as a secondary column. Using the terminology of the reformulation elucidated in Subsection 2.2 we may say that we have added the constraint $x_i + x_j \leq 1$ to the system of equations $Ax = b$.

In this section we present ways to spot such useful constraints. We describe a way to condense these inequalities into a smaller number of inequalities in case this looks desirable. In other words we will consider the problem of condensing the secondary columns into a fewer number of secondary columns.

3.1 Condensing the secondary columns

Let I_S the m by l submatrix consisting of all the secondary columns of the incidence matrix I of an exact cover problem. Each column of I_S has two entries equal to one and $m-2$ entries that are equal to zero. A typical column codes the inequality $x_i + x_j \leq 1$, the i-th and j-th component of the column are equal to one.

Using I_S we construct a graph Λ. At the beginning of the construction Λ is the complete graph on the vertex set $V = \{1, \ldots, l\}$. We pick the columns of I_S one by one. If a column corresponds to the inequality $x_i + x_j \leq 1$, then we delete the edge connecting the nodes i and j. Let L be the adjacency matrix of Λ. We illustrate the procedure working out an example in details.

Example 3. The matrix of the secondary columns I_S of an incidence matrix I is given in Table 2. The adjacency matrix L of the graph Λ is shown in Table 3

Next we color the nodes of Λ. A coloring of the vertices of Λ with t colors is termed a well coloring if each vertex receives exactly one of the t colors and if adjacent vertices are not receiving the same color. The set of all the vertices of Λ colored with the i-th color form a color class C_i. Clearly,

Table 2: The incidence matrix I_S in Example 3.

1	•	•	•	•											
2	•				•	•									
3							•	•	•	•					
4							•				•	•	•	•	
5		•			•										•	
6								•			•					•	•
7									•			•				•	
8			•										•				•
9				•		•				•				•	•		

Table 3: The adjacency matrix L of the graph Λ in Example 3.

	1	2	3	4	5	6	7	8	9
1			•	•		•	•		
2			•	•		•	•	•	
3	•	•			•			•	
4	•	•			•				
5			•	•		•	•	•	
6	•	•			•				•
7	•	•			•			•	•
8		•	•		•		•		•
9						•	•	•	

the color classes $C_1, \ldots, C_t$ form a partition of the vertex set V of Λ.

It is well known that determining the minimal number of colors a given graph can be well colored with belongs to the NP complete complexity class. We will use a greedy algorithm to color the nodes of Λ.

Suppose that a color class, say C_1, has at least three elements. Let $C_1 = \{i(1), \ldots, i(r)\}$, where $r \geq 3$. Since C_1 is a colors class of the nodes of Λ, it follows that the inequality $x_i + x_j \leq 1$ holds for each $i, j \in C_1$. In other words only at most one of the zero-one variables $x_{i(1)}, \ldots, x_{i(r)}$ can be equal to one. Therefore the inequality $x_{i(1)} + \cdots + x_{i(r)} \leq 1$ must hold. We form an m-dimensional column vector a whose entries on the $i(1), \ldots, i(r)$ components are equal to one and the other components are filled with zeros. We add a to the incidence matrix I_S as a secondary column. Then we delete each column of I_S that corresponds to a constraint $x_i + x_j \leq 1$ with $i, j \in C_1$. In this way we add one column and drop $r(r-1)/2$ columns. We carry out this reduction in connection with each color class that contains at least three elements. Of course in order to compensate the changes when we delete the column of I_S corresponding to the constraint $x_i + x_j \leq 1$ we have to connect the nodes i and j in the graph Λ. We can color the new graph Λ again and if there is a color class with at least three elements, then we can further reduce the number of columns of the I_S matrix.

The nodes of the graph Λ in Example 3 can be colored with three colors. For instance such that the colors classes are the following

$$C_1 = \{1, 2, 5, 9\}, \quad C_2 = \{3, 4, 6, 7\}, \quad C_3 = \{8\}.$$

Using these color classes the I_S matrix in Table 2 can be condensed into a smaller matrix given by Table 4.

Table 4: The condensed form of the I_S matrix in Example 3 and displayed in Table 2.

1	•		•			
2	•					
3		•		•		
4		•			•	•
5	•					
6		•				•
7		•				
8			•		•		•
9	•			•		•	

3.2 Dominance

Let us consider Problem 1 with the data (U, F).

Definition 1. Suppose $u \in U$, $A, B \in F$ such that $u \notin A \cup B$ and $A \cap B = \emptyset$. We say that u dominates the pair A, B if $u \in C$ implies $C \cap (A \cup B) \neq \emptyset$ for each $C \in F$.

PROPOSITION 1. *If u dominates A, B, then A, B cannot be together members of a partition $P \subseteq F$ of U.*

PROOF. Let $P \subseteq F$ be a partition of U and assume on the contrary that $A, B \in P$. Since P is a partition, there is a $C \in P$ for which $u \in C$. As $u \notin A$, $u \notin B$, it follows that $C \neq A$, $C \neq B$. Since P is a partition we get that $C \cap A = \emptyset$, $C \cap B = \emptyset$. On the other hand by the definition of the dominance $C \cap A \neq \emptyset$ or $C \cap B \neq \emptyset$. This contradiction completes the proof. □

For a fixed $u \in U$, $A, B \in F$ by inspecting all possible choices for $C \in F$ one can decide if u dominates A, B. Suppose that $F = \{A_1, \ldots, A_m\}$. If u dominates A_i, A_j, then we add a secondary column to the incidence matrix of the problem. The secondary column codes the inequality $x_i + x_j \leq 1$ preventing A_i, A_j being together members of a partition of U.

Before embarking on a large scale exact cover search it is advisable to carry out a complete inspection to locate all the possible dominances. We may not find any dominance but also we may and it may reduce the size of the search space considerably.

As an exercise the reader can verify that in the exact cover problem instance given in Example 1 the element $2 \in U$ dominates $A_2, A_5 \in F$. So we added a secondary column to the incidence matrix. This secondary column is labeled by a. Table 1 shows the original incidence matrix and the new incidence matrix extended with four secondary columns.

Let us consider Problem 4 with the data (U, F), where U is partitioned into U_1 and U_2. Let $I = [I_P, I_S]$ be the m by n incidence matrix associated with (U, F). The primary columns are in I_P and the secondary columns are in I_S.

Definition 2. Let a, b be column vectors of I_P. We say that a dominates b if $a_i = 1$ implies $b_i = 1$ for each i, $1 \leq i \leq m$.

PROPOSITION 2. *If a dominates b, then we may move b from I_P to I_S.*

PROOF. Let u, v be the elements of U that correspond to the columns a, b, respectively. Set $U_1' = U_1 \setminus \{v\}$, $U_2' = U_2 \cup \{v\}$.

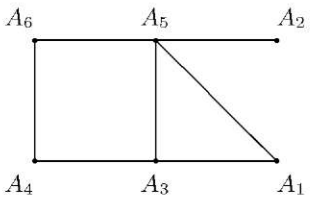


Figure 8: Graph Γ in Example 1.

Suppose $P \subseteq F$ such that $P_{|U_1}$ is a partition of U_1 and $P_{|U_2}$ is a packing of U_2. It is clear that $P_{|U_1'}$ is a partition of U_1' and $P_{|U_2'}$ is a packing of U_2'.

Next assume that $P \subseteq F$ such that $P_{|U_1'}$ is a partition of U_1' and $P_{|U_2'}$ is a packing of U_2'. As $P_{|U_1'}$ is a partition of U_1', there is an $A \in P$ for which $u \in A \cup U_1'$. Consequently $u \in A$. Since the column a dominates column b from $u \in A$, it follows that $v \in A$. Consequently $v \in A \cup U_1$. Therefore $P_{|U_1}$ is not only a packing of U_1 but $P_{|U_1}$ is a partition of U_1. This completes the proof. □

In the incidence matrix given in Table 1 the 1-st column dominates the 6-th column. So the 6-th column can be changed to a secondary column. The 5-th columns dominates the 3-rd column and so the 3-rd column can be turned into a secondary column. Finally, the 4-th column dominates the 7-th column and hence the 7-th column can be a secondary column.

Turning a primary column to a secondary column improves the efficiency of the algorithm for fewer columns are used to compute the branching factors. This improvement is not particularly significant. However, the cost of checking the dominances of columns is not high either. In the same time it may happen that a secondary column drops out entirely when we condense the secondary columns.

3.3 Cliques

To the data (U, F) of Problem 1 we may assign a graph Γ. The nodes of Γ are the elements of F. Two distinct nodes A_i, A_j are connected by an edge if $A_i \cap A_j = \emptyset$. Let M be the adjacency matrix of Γ. The rows and columns of M are labeled by the elements of F and so M is an $|F|$ by $|F|$ matrix, that is, an m by m matrix. The graph Γ in Example 1 is depicted in Figure 8 and Table 5 contains the adjacency matrix M of Γ.

Definition 3. The graph Γ is called the packing graph of the instance (U, F) of the exact cover problem.

The cliques in the graph Γ correspond to packings of the ground set U and so naming Γ the packing graph of the instance (U, F) of Problem 1 is justifiable.

In the particular case when each member of the family F has the same number of elements, that is, when $|A_1| = \cdots = |A_m|$, then each partition of U must be an exact k-cover of U, where $k = |U|/|A_1|$.

The additional information that each partition $P \subseteq F$ of U consists of k members of F can be exploited to speed up the search. For the sake of brevity let us call a partition consisting of k subsets a k-partition. A k-partition $P \subseteq F$ of U corresponds to a k-clique Δ in the graph Γ. The existing and well tested clique search algorithms (see [3], [4], [8])

Table 5: The adjacency matrix M of the graph Γ in Example 1. The rows and columns are labeled by $1,\ldots,6$ instead of $A_1,\ldots,A_6$.

	1	2	3	4	5	6
1			•		•	
2					•	
3	•			•	•	
4			•			•
5	•	•	•			•
6				•	•	

can be used to solve exact k-cover problems too. It seems that the exact cover algorithm outperforms the clique search algorithms. The author does not know about any systematic rigorous comparison project and so the evidence is only anecdotical. Such an undertaking would need algorithms implemented in a similar programming environment and a stock of bench mark tests. At this moment both are lacking.

The clique search algorithm makes amply evident that well coloring of the nodes of graph Γ helps in reducing the size of the search space. Suppose $C_1,\ldots,C_t$ are all the color classes in a well coloring of Γ with t colors. As a k-clique Δ in Γ cannot have two nodes in the same color class, it follows that $k \leq t$. We can use this idea in connection with exact k-cover search.

For the color class C_i let us form an m-dimensional column vector a_i such that the j-th component of a_i is equal to one whenever $j \in C_i$. We then add the column vectors $a_1,\ldots,a_t$ to the incidence matrix of the exact cover problem as secondary columns.

Suppose that at one point during the exact k-cover search we find that a partially built partition $P \subseteq F$ of U contains p members of F and q of the secondary columns $a_1,\ldots,a_t$ are still in the incidence matrix. (A partially built partition P of U is of course simply a packing of U.) If $p+q<k$, then plainly the partial partition P cannot be extended to a k-partition of U. Therefore a little extra book keeping provides us with an opportunity to prune the search tree.

The coloring idea also provides us with a preprocessing opportunity. Suppose that the nodes $i,j \in U$ of Γ are adjacent. Consider the set V of all the common neighbors of i and j. (Since Γ has no loops $i,j \notin V$.) Let $\Gamma_{i,j}$ be the subgraph of Γ spanned by V. Suppose that the nodes of $\Gamma_{i,j}$ are well colored with t colors. Note that if $t<k-2$, then $\Gamma_{i,j}$ cannot contain any $(k-2)$-clique and consequently Γ cannot contain any k-clique. In other words, if $t<k-2$ holds, then the members $A_i,A_j \in F$ cannot be together in a k-partition $P \subseteq F$ of U. At the outset of the computation one may test each edge of Γ and as a result one may get a number of secondary columns to add to the incidence matrix and speeding up the search in this way.

4. PARALLEL SEARCH

In this section we divide the given exact cover problem into a large number of smaller instances. These smaller size problems can be solved independently of each other on separate processors and using ordinary serial exact cover search algorithms.

We place the subproblems to a large stack. The master processor maintains the stack assigning the subproblem on the top of the stack to the slave processor that is ready to work. One usually is not in position to predict the run times of these subproblems. Therefore we try to construct problems many times more than processors are available. We hope that in this way the work load of the processors turn out to be reasonable well balanced.

4.1 Splitting partitions

Let (U,F) be an initial data of Problem 1 and suppose that F_1, F_2, F_3 form a partition of F.

Definition 4. We say that F_1, F_2, F_3 is a splitting partition of (U,F) if $F_1 \neq \emptyset$, $F_3 \neq \emptyset$ and if $A_i \cap A_j \neq \emptyset$ for each $A_i \in F_1$, $A_j \in F_3$.

For a splitting partition F_1, F_2, F_3 of (U,F) we consider Problem 1 in two copies with initial datum $(U, F_1 \cup F_2)$, $(U, F_2 \cup F_3)$, respectively.

PROPOSITION 3. *If $P \subseteq F$ is a solution of Problem 1 with the initial data (U,F), then either P is a solution of Problem 1 with the initial data $(U, F_1 \cup F_2)$ or P is a solution of Problem 1 with the initial data $(U, F_2 \cup F_3)$.*

PROOF. Assume on the contrary that P is a partition of U that solves the (U,F) instance of Problem 1 and P does not solve any of the $(U, F_1 \cup F_2)$, $(U, F_2 \cup F_3)$ instances of Problem 1. This means that there is an $A_i \in P$ such that $A_i \in F_1$ and there is an $A_j \in P$ such that $A_j \in F_3$. By the definition of the splitting partition F_1, F_2, F_3 of (U,F) it follows that $A_i \cap A_j \neq \emptyset$. This violates that P is a partition of U. □

We would like to point out that a partition P of U can solve the instances $(U, F_1 \cup F_2)$, $(U, F_2 \cup F_3)$ simultaneously. In other words the list of solutions to the first instance and the list of solutions to the second instance may have elements in common. Dividing a problem into two smaller problems in this way cannot be used to determine the number of solutions or enumerate all possible solutions. However, it is perfectly usable to inspect all solutions. One may inspect a solution several times but none of the solutions remains unchecked.

Clearly, the larger is the quantity $\rho = \min\{|F_1|, |F_3|\}$ the smaller are the sizes of the instances $(U, F_1 \cup F_2)$, $(U, F_2 \cup F_3)$ and so we try to find a splitting partition F_1, F_2, F_3 of (U,F) with a maximal ρ.

PROBLEM 6. *Given (U,F) find a splitting partition F_1, F_2, F_3 of (U,F) preferably with large value of ρ.*

Finding a splitting partition for which ρ is maximal is itself a computationally hard problem. It can be reduced to finding a certain maximal bipartite complete subgraph in a bipartite graph. We overcome this difficulty by constructing a nearly optimal splitting partition using a computationally less demanding greedy algorithm instead of finding an optimal splitting partition.

Here is a straight forward greedy algorithm to construct a splitting partition F_1, F_2, F_3 of (U,F).

(1) Initially set $F_1 = \emptyset$, $F_2 = \emptyset$, $F_3 = U$.

(2) Pick an $A \in F_2 \cup F_3$. Let F_A be the family of members of F_3 that is disjoint to A. We choose an A for which $|F_A|$ is minimal.

(3) If $|F_3| - |F_A| - 1 \geq |F_1| + 1$, then set $F_1 = F_1 \cup \{A\}$, $F_2 = (F_2 \cup F_A) \setminus \{A\}$, $F_3 = F_3 \setminus [F_A \cup \{A\}]$ and continue at step 2.

(4) If $|F_3| - |F_A| - 1 < |F_1| + 1$, then terminate.

If $F_1 \neq \emptyset$, $F_3 \neq \emptyset$, then F_1, F_2, F_3 is a splitting partition of (U, F). A possible parallel algorithm to solve Problem 1 is the following. The greedy algorithm provides us with a splitting partition F_1, F_2, F_3 of (U, F). If $\rho \geq 2$ we consider the smaller problems with the initial datum $(U, F_1 \cup F_2)$, $(U, F_2 \cup F_3)$. If $\rho = 1$, then we do not construct smaller problems. We repeat this procedure in connection with the smaller problems. Continuing in this way finally we end up with the smaller instances $(U, F_1'), \ldots, (U, F_s')$ of Problem 1. The new instances can be processed independently of each other.

4.2 The chopping method

Let (U, F) be an initial data of Problem 2 and suppose that the sets $F_1, \ldots, F_t$ form a partition of F. Choose an i, $1 \leq i \leq t$ and let σ_i be the number of distinct members B, C of F_i for which $B \cap C = \emptyset$. Set $\sigma = \sigma_1 + \cdots + \sigma_t$.

Definition 5. The partition $F_1, \ldots, F_t$ is called a (σ, t) chopping of (U, F).

If $B, C \in F_i$ and $B \cap C = \emptyset$, then we construct a new instance of the exact $(k-2)$-cover problem whose initial data is (U', F'). Here F' consists of all members of F that are disjoint to B and C and $U' = U \setminus (B \cup C)$. Suppose we have a $(\sigma, k-1)$ chopping of (U, F). In this chopping there are σ pairs $B_1, C_1, \ldots, B_\sigma, C_\sigma$ of members of F for which the above construction can be carried out. Let $(U_1', F_1'), \ldots, (U_\sigma', F_\sigma')$ be these resulted initial datum.

PROPOSITION 4. *If the exact k-cover problem with the initial data (U, F) has a solution, then the exact $(k-2)$-cover problem has a solution with at least one of the initial datum $(U_1', F_1'), \ldots, (U_\sigma', F_\sigma')$.*

PROOF. Assume on the contrary that the exact $(k-2)$-cover problem does not have any solution with the initial data (U_i', F_i') for each i, $1 \leq i \leq \sigma$. By the assumption of the proposition there is a partition of U that solves the (U, F) instance of the k-cover problem. Let P be such a partition of U.

If $B_1, C_1 \in P$, then $P \setminus \{B, C\}$ is a solution of the (U_1', F_1') instance of the $(k-2)$-cover problem. This contradicts the indirect assumption. Thus either $B_1 \notin P$ or $C_1 \notin P$. We can add a secondary column to the instance (U, F) that prohibits B_1 and C_1 being members of an exact k-cover together. We may repeat this argument with B_2, C_2 and finally with B_σ, C_σ. In this way we can add σ secondary columns to the instance (U, F). Note that the σ secondary columns can be condensed into $k-1$ secondary columns. Namely, if $F = \{A_1, \ldots, A_m\}$ and $A_i \in F_j$, then the i-th component of the j-th condensed secondary column is equal to 1. Let (U^*, F^*) be the instance augmented by the $k-1$ secondary columns. The above argument shows that each partition P of U that solves the instance (U, F) can be extended to a partition P^* of U^* that solves the instance (U^*, F^*).

The secondary columns guarantee that each partition that solves the instance (U^*, F^*) can have at most $k-1$ members of U^*. But P has k members. This contradiction completes the proof. □

A parallel k-cover algorithm suggested by Proposition 4 is the following. Let us start with the (U, F) instance of the k-cover problem. Construct a $(\sigma, k-1)$ chopping of (U, F). Identify the sets $B_1, C_1, \ldots, B_\sigma, C_\sigma$ of F. Using these sets construct the $(k-2)$-cover problems with initial datum

$$(U_1', F_1'), \ldots, (U_\sigma', F_\sigma').$$

If one these instances has a solution then this can be extended to a solution of the original instance (U, F). By Proposition 4, each solution of the instance (U, F) arises in this way. Stating it equivalently we reduced the (U, F) instance of the k-cover problem to the

$$(U_1', F_1'), \ldots, (U_\sigma', F_\sigma')$$

instances of the $(k-2)$-cover problem. The σ subproblems can be solved independently of each other. One can use any serial $(k-2)$-cover algorithm.

We have to point out that this algorithm can find a k-cover of U several times. This it is not suitable to determine the number of solutions of the instance (U, F). However, a slight modification of the procedure can prevent finding a solution twice. Namely, when the $(k-2)$-cover algorithm terminates with the initial data (U_1', F_1') add a secondary column to each instances

$$(U_2', F_2'), \ldots, (U_\sigma', F_\sigma')$$

that prevents the sets B_1, C_1 to be members of a $(k-2)$-cover together. Then work with the modified instances

$$(U_2^{(1)}, F_2^{(1)}), \ldots, (U_\sigma^{(1)}, F_\sigma^{(1)}).$$

When the $(k-2)$-cover algorithm terminates with the initial data $(U_2^{(1)}, F_2^{(1)})$ add a secondary column to each of the instances

$$(U_3^{(1)}, F_3^{(1)}), \ldots, (U_\sigma^{(1)}, F_\sigma^{(1)})$$

that prevents B_2, C_2 to be members together of a $(k-2)$-cover. Then work with these new instances

$$(U_3^{(2)}, F_3^{(2)}), \ldots, (U_\sigma^{(2)}, F_\sigma^{(2)}).$$

Continuing in this way finally solve the $(U_\sigma^{(\sigma-1)}, F_\sigma^{(\sigma-1)})$ instance of the $(k-2)$-cover problem.

It seems that we have lost the parallel nature of the algorithm. But this is not the case. One can construct the

$$(U_1^{(0)}, F_1^{(0)}),\ (U_2^{(1)}, F_2^{(1)}), \ldots, (U_\sigma^{(\sigma-1)}, F_\sigma^{(\sigma-1)})$$

instances of the $(k-2)$-cover problem at the beginning and solve them independently of each other. Here $(U_1^{(0)}, F_1^{(0)}) = (U_1', F_1')$.

Any partition $F_1, \ldots, F_{k-1}$ of F can serve as a $(\sigma, k-1)$ chopping of (U, F). However, the number of the initial instances can be reduced by choosing a $(\sigma, k-1)$ chopping of (U, F) for which the value of σ is possibly small. This suggests the next problem

PROBLEM 7. *Given (U, F) and a positive integer t. Find a (σ, t) chopping of (U, F) preferably with small value of σ.*

Finding a (σ, t) chopping of (U, F) with an optimal value of σ is computationally expensive. Therefore we settle for a greedy algorithm to construct a not necessarily optimal (σ, t) chopping. The greedy procedure below provides a (σ, t) chopping of (U, F).

(1) Set $G = F$, $F_1 = \cdots = F_t = \emptyset$ as initial values.

(2) While $G \neq \emptyset$ repeat the following. Pick an $A \in G$. Let d_i be the number of members of F_i that is disjoint to A. Let d_j be a smallest of $d_1, \ldots, d_t$. Set $G = G \setminus \{A\}$, $F_j = F_j \cup \{A\}$.

5. A CLUMSY ALGORITHM

Let us consider Problem 1 with the data (U, F), where $U = \{1, \ldots, n\}$ is the given ground set and $F = \{A_1, \ldots, A_m\}$ is the given family of subsets of U. Let I be the incidence matrix associated with the data (U, F), that is, the cell at the intersection of the i-th row and j-th column contains one if $j \in A_i$. We try to construct a partition $P \subseteq F$ of U. Initially we set $P = \emptyset$. In a typical case we choose an $A_i \in F$ that is disjoint to each member of P. Setting

$$U = U \setminus A_i, \quad F = F \setminus \{A_i\}, \quad P = P \cup \{A_i\}$$

we end up with a new smaller instance of the problem.

In the course of this reduction we check many times if given sets $A_j, A_j \in F$ are disjoint or not. The adjacency matrix M of the graph Γ contains these pieces of information and they do not change as the computation unfolds. The dancing link algorithm we propose is not as sophisticated as Knuth's dancing link algorithm. We call it the clumsy version.

A row operation is the following. Given a row. We go through the entries of the row moving from left to right starting at the frame entry and link out the entries vertically. One can completely reverse the result of a row operation by going through the entries of the row moving from right to left starting at the frame entry and link in the entries vertically. Let us call this operation a reverse row operation.

A column operation is the following. Given a column. We go through the entries of the column moving from up to down starting at the frame entry and link out the entries horizontally. One can completely reverse the result of a column operation by going through the entries of the column moving from down to up starting at the frame entry an link in the entries horizontally. Let us call this operation a reverse column operation.

A row pivot operation is the following. Given a set $A_i \in F$. We go through the entries of the horizontal frame moving from left to right starting at the corner of the frame. The entries in the horizontal frame correspond to elements of the ground set U. If the entry in the horizontal row corresponds to the element $u \in U$ and $u \in A_i$, then carry out a column operation. Finally place the index i to a stack. One can completely redo the result of a row pivot operation restoring the original conditions. Pick up the index i from the stack. Go through the elements of A_i in reverse order. If $u \in A_i$, then link in horizontally the entry corresponding to the element u and carry out a reverse column operation. Let us call this operation a reverse row pivot operation.

A column pivot operation is the following. Given a subset $A_i \in F$. The row corresponding to the subset A_i is called the pivot row. We go through the entries of the vertical frame moving from up to down starting at the corner of the frame. The entries in the vertical frame correspond to members of the family F. If the entry in the vertical frame corresponds to $A_j \in F$ and $A_j \cap A_i \neq \emptyset$, then carry out a row operation. Then place the index j to a stack. (The condition $A_j \cap A_i \neq \emptyset$ can be checked by reading off an entry of the adjacency matrix M of the graph Γ what we store.) One can completely reverse the results of a column pivot operation to restore the original situation. In order to do this pick up the j indices from their stack one by one and carry out a reverse row operation in connection with the j-th row. (Remember that the j indices are stored in reverse order in the stack.) But in fact we do not wish to restore the original situation completely. We want to delete the pivot row from the table. We can achieve this simply by carrying out a row operation in connection with the pivot row. Let us call this operation a reverse column pivot operation.

Suppose that we are given an instance (U, F) of Problem 1. Based on the branching factors we locate a member A_i of F. Using A_i we carry out a row pivot operation then a column pivot operation on the incidence matrix and we end up with a smaller instance of the problem.

The clumsy algorithm is not faster than the more sophisticated version and for the adjacency matrix M and for the stacks it requires more storage place. The only reason we mention it that it handles the information relegated to the secondary columns in a more flexible manner. As a matter of fact the clumsy algorithm does not need secondary columns at all. For example the information that the sets $A_i, A_j \in F$ cannot be together members of a partition $P \subseteq F$ of U can be incorporated into the problem simply changing an entry in the adjacency matrix M from one to zero. This modification can be done at any phase of the computation. The parallel computations greatly benefit from this larger flexibility.

In order to get some idea of the sizes of the instances one can handle we would like to mention two computational results briefly. They are published in [6] and [7], respectively. In the first one the initial data can be characterized by the next numbers $|U| = 3^7 = 2187$, $|F| = 3^7 = 2187$, $k = 3^5 = 243$. This means that the search tree had 243 levels. All exact k-covers were required for inspection. There were 158 760 solutions altogether.

In the second instance the summary statistics of the initial data is the following $|U| = (10)(13^2) = 1690$, $|F| = 13^3 = 2197$, $k = 13^2 = 169$. Therefore the search tree has 169 levels. All exact k-covers were needed for inspection. There were about 5×10^7 solutions.

6. CONCLUSIONS

To each instance (U, F) of the exact cover problem a graph Γ, the packing graph of the instance, can be associated. Certain edges of Γ can be deleted without changing the number of partitions of the ground set U. This information can be conveniently incorporated into the secondary columns of the initial data of the exact cover instance. This speeds up the computations. We discuss how a dominance concept helps to detect deletable edges in Γ and then how to condense the resulted secondary columns by using greedy coloring. This idea works for each exact cover problem. For exact k-covers further speed up is available also based on greedy coloring. We may refer to these as preprocessing ideas.

The paper also deals with parallel exact cover algorithms. The first one is based on the concept of splitting partition of the instance (U, F). This can be applied for solving each exact cover problem. The second parallel algorithm can be used only for solving exact k-cover problems and it utilizes the concept of $(\sigma, k-1)$ chopping of the instance (U, F). In the last section of the paper we propose to keep the packing

graph Γ in the memory of the computer during the whole computation in order to retain a greater flexibility for this last type of parallel algorithm.

7. ACKNOWLEDGMENTS

The author would like to thank István Kovács for his help in presenting the paper and László Kóródi for his help in the computations.

8. REFERENCES

[1] P. Kaski, P. R. J. Östergård, *Classification Algorithms for Codes and Designs*, Springer-Verlag Berlin Heidelberg 2006 pp. 145–154.

[2] D. E. Knuth, Dancing links, in *Millennial Perspectives in Computer Science*, J. Davies, B. Roscoe, and J. Woodcock, Eds., Palgrave Macmillan, Basingstoke, 2000, pp. 187–214.

[3] R. Carraghan, P. M. Pardalos, An exact algorithm for the maximum clique problem, *Operation Research Letters* **9** (1990), 375–382.

[4] P. R. J. Östergård, A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics* **120** (2002), 197–207.

[5] P. R. J. Östergård, S. Szabó, Elementary p-groups with the Rédei property, *International Journal of Algebra and Computation* **17** (2007), 171–178.

[6] S. Szabó, Nonfull-rank factorizations of finite elementary 3-groups, *International Electronic Journal of Algebra* **3** (2008), 96–102.

[7] S. Szabó, Verifying Rédei's conjecture for $p = 13$, *Mathematics of Computation* **80** (2011), 1155–1162.

[8] E. Tomita, T. Seki, An efficient branch-and-bound algorithm for finding a maximum clique, *Lecture Notes in Computer Science* 2631 (2003), 278–289.

Community detection and its use in Real Graphs

András Bóta
Department of Computer Science
University of Szeged, Hungary
bandras@inf.u-szeged.hu

László Csizmadia
Department of Computer Science
University of Szeged, Hungary
cheeseme@freemail.hu

András Pluhár
Department of Computer Science
University of Szeged, Hungary
pluhar@inf.u-szeged.hu

ABSTRACT

We survey and unify the methods developed for finding overlapping communities in Small World graphs in the recent years. The results have impact on graph mining; we give some demonstration of this.

General Terms

Theory

Keywords

data mining, graphs, communities

1. INTRODUCTION

The discovery of Small World graphs has changed the direction of interest in graph theory profoundly. These graphs are different from those that were studied before, and also the questions that were asked about those. It is not easy to collect the information to build such a graph, or give models to generate it. The sheer size of the real problems prohibits most of the time consuming algorithms, so the researcher has to fall back on simpler heuristics, sometimes derived from physical intuition [3, 5, 20]. Following the usual notation, a graph G has vertex set $V(G)$, edge set $E(G)$. If the later one consists of ordered pairs, then G is directed, and an edge might be also weighted.

An intriguing question is the classification of vertices of a graph. One can consider the usual clusters and also overlapping sets, that we call communities. Here we concentrate on the possible definitions, search and use of communities. While for clustering both the top down and bottom up algorithms are used for defining and finding the classes, all known algorithms for communities are bottom up.

2. SOME ALGORITHMS

Here we consider only three algorithms. The selection is arbitrary, although has some justification. Maybe the first algorithm that was used for finding communities is the N^{++}.

However, since we could get no permission to use the data set it was designed for, it has not been published yet in English. After that several similar algorithms were proposed; unfortunately the qualities of implementations differ so it is not easy to compare them. The k-Clique percolation method was the first widely known algorithm, which was also applied to real world problems. Edge clustering is the third algorithm we mention; it has mainly theoretical interest.

2.1 The N^{++} algorithm.

[23, 11] It is a generic algorithm, with arbitrary functions

$$f : 2^{V(G)} \times V(G) \to \mathbb{R}$$

and $c : \mathbb{N} \to \mathbb{R}$. Here $f(A, x)$ describes the strength of a community A with a vertex x. Then the algorithm joins x to A if $f(A, x) \geq c(|A|)$. The **Build** routine gets the first approximation of communities $\mathcal{K}$ in a bottom up way.

The pseudo-code of Build

begin(Build)

Input G, k, c (`max k-size c-communities`)

Let $\mathcal{K} := V(G)$ (`nodes are communities.`)

For $i = 1$ to k

$\forall A \in \mathcal{K}$, $x \in V(G)$ if $f(A, x) \geq c(|A|)$ then put $A \cup \{x\}$ into $\mathcal{K}$.

Remove all $A \in \mathcal{K}$, for which $A \subset B \in \mathcal{K}$, and $A \neq B$.

Print $\mathcal{K}$, "c-communities of G up to size k."

end(Build)

After running Build, we use **Merge** to glue communities that are almost identical. Let C be a graph, where $V(C) = \mathcal{K}$, and $(A, B) \in E(C)$ if $A \cap B$ is "big" then changes $\mathcal{K}$ to $(\mathcal{K} \setminus \{A, B\}) \cup \{A \cup B\}$. Then the components of C are declared to be the communities. The practice suggested the following set-ups. The big means the 60% of the smaller set. The function $f(A, x)$ depends on the number of paths with length one and two from x to A. That is to get the communities containing x, it is enough to search $N^{++}(x) := N(N(x))$.

Some similar methods are listed in [13].

2.2 k-Clique percolation.

[21] Here a $k \in \mathbb{N}$ is fixed. After finding all k-size clique in G, the graph Q_k is considered such that the vertices of Q_k are these cliques, and $(A, B) \in E(Q_k)$ iff $|A \cap B| = k - 1$. Finally a k-community is the unions of cliques of a connected components.

2.3 Edge clustering.

[22, 25] One chooses an arbitrary clustering on the set of edges. Then the communities are defined as the set of endpoints of the clusters.

These methods differ in output, i. e. in the type of communities, and in the computing costs. Although the edge clustering is easy to compute, it has serious drawbacks in use. (First of all is that the overlap among communities is maximum one vertex.) The N^{++} and Clique percolation are more promising; here the implementation issues are crucial. For small world graphs both can perform almost in linear time, which is a natural requirement if one wants to deal with real problems.[1]

2.4 A unified view

These algorithms, and those that were mentioned but not listed, has a common core. Their execution consists of two steps. In the first a hypergraph $\mathcal{F} = (V, \mathcal{H})$ is defined (and computed), where $V = V(G)$, the original point set of the graph G, and $\mathcal{H} \subset 2^V$. The elements of $\mathcal{H}$ can be considered as the building blocks of the communities. In the second step one endows the set $\mathcal{H}$ with an appropriate d distance function and thereby establishes a metric space $\mathcal{M} = (\mathcal{H}, d)$. Then a chosen clustering algorithm is executed on $\mathcal{M}$, yielding a set of clusters $\mathcal{C}$. Finally, the arising clusters are associated to the subsets of V such that $K_i = \cup_{H \in C_i \in \mathcal{C}} H$, where K_i, the ith community corresponds to C_i, the ith cluster and K_i is just the union of those hyperedges that belong to C_i.

In the case of the mentioned algorithms $\mathcal{H}$ consist of vertex sets of the small dense subgraph, k-cliques and the edges, respectively. The distance functions are represented by an appropriate graph $\mathcal{D}$, take the value one if there is an edge, infinity otherwise. In the first case $(K_i, K_j) \in \mathcal{D}$ if $|K_i \cap K_j|$ is big enough, in the second if $|K_i \cap K_j| = k - 1$, while we left this as a parameter in the third case.

3. EVALUATION

Since more or less all community (or cluster) definitions are arbitrary [18], there are several ideas to measure their goodness. This is a crucial point and naturally the viewpoint of researches differ. There are direct and indirect methods to assess the usefulness of communities, the following list is far from being complete.

3.1 Appearance, parametrization

First of all, one has to run the algorithms, get the outputs and possible make mathematical predictions for certain graph classes. That is an important factor is the speed of these algorithms. However, it is not easy to compare the real speed of these algorithms since it depends strongly on the implementations and test graphs (being real or theoretical). Definitely all three algorithms, and perhaps most algorithms in that family we described in subsection 2.4 are fast, and designed to solve huge problems. In subsection 3.4 we recur to this problem, and report some date on time and a goodness measure (modularity) of the solutions.

The clique percolation method is appealing from both theoretical and practical view. For Erdős-Rényi random graphs the clique percolation process is thoroughly studied and well understood, [6]. It was reported to be useful also in practice, [1]. However, it sometimes gives too large communities and the parametrization is elusive, since one has to decide for which value k to be chosen?

The N^{++} algorithm looks arbitrary, and do not yield for theoretical investigations. Its main advantages are the speed, the small diameter of the communities and its robustness. The edge clustering methods are not well studied or tested in practice. Their inherent problem is that communities derived this way may have only one common element, what is too restrictive in real graphs.

We tested on these algorithms on some benchmark graphs, let us illustrate our findings on the famous Zachary graph, see [26]. This is a friendship graph of a karate club that split into two parts, A and B. Part A is centered around their Japanese master, while part B is led by the club administrator. The the clique percolation method gives three communities for $k = 3$ with sizes 3, 6 and 24. For $k = 4$ there are also three communities, the sizes are 4, 4 and 7, while for $k = 5$ there is one community of size six. Here a blend of $k = 3$ and $k = 4$ seems to be appropriate, and the communities are on the two sides of border where the split occurred. The N^{++} algorithm results in twelve communities, four of size three, five of size four, one of size six, and two of size seven. All but one communities are entirely either in A or in B. One might argue that the club was always one the verge of demise that happened at the end.

3.2 Graphical.

Another way is to compare the communities with some visualized form of the graph; this was the most common approach in the early publications. Indeed, the clustering methods provide classes that conform the eye. Assessing communities (permitting overlapping) are harder, since visualization is not an obvious task anymore. Some ideas, like showing the intersection graph of communities can help. However, this approach has certain limits; it works only for small graphs and it is always subjective.[2]

Another possibility is to draw some derived graphs. Among these the intersection graph H of the communities performed best. Here the vertices are the communities of G, and an edge is drawn if the communities associated to the vertices

[1] This means millions of vertices. The N^{++} available in the Sixtep software, while the Clique percolation in the CFinder.

[2] For graph visualization the so-called *force directed* algorithms performed best. However, these usually take $O(n^2)$ time that prohibit the use when n is several thousand or million.

has a non-empty overlap. That is $I(G) = (V(H), E(H))$, where $V(H) = \mathcal{K}$ and $(C_i, C_j) \in E(H)$ if $|C_i \cap C_j| > 0$.

Again, for the Zachary graph the clique percolation method gives an unconnected graph H. The intersection graph H based on the N^{++} algorithm is more delicate. It consist of two dense subgraph with one common vertex x. A four element community corresponds to vertex x, and this community contains the master (1), the administrator (33) and the vertices labeled by 3 and 9. The community was almost a clique, except that the master and the administrator were not friends. When the split occurred, 3 and 9 ended up in different parts destroying completely the only community that connected the two parts. One might speculate that the friendship of 3 and 9 was responsible for the cohesion of the club, and when it could not take more pressure they took parts which meant the end of the club, too.

3.3 Random Small World Graphs

There are several ways to generate random graphs having similar properties that of real Small World graphs, [2, 8]. From those we tried out the Preferential Attachment (PA) and the Vertex Copy (VC) models. In both of these models the graph is build step by step, while the neighborhood of the newly arrived vertex x is chosen differently. In the PA model the new vertex x brings k new edges, and the other end of these edges are at an old vertex y with probability proportional to $d(y)$ and taken independently from each other. In the VC model an old vertex s is selected uniformly, and the new vertex x takes vertices independently from $N(s)$ with a prescribed probability p.

The results are far from being conclusive, and indeed tell more about the models (PA and VC) than the community algorithms (CPC, N^{++}). Note, that a different approach, using random intersection graphs, is investigated in [24].[3] Here we illustrate it on two sets of graphs that approximately belong to the same category. For all these the number of vertices is 100, G_1 and H_1 were generated by the PA model, $|E(G_1)| = 192$, $|E(H_1)| = 358$ while G_2 and H_2 come from the VC model with $|E(G_2)| = 151$ and $|E(H_2)| = 378$. The #C and #CO mean the number of clusters and communities, while the column with head k contains the number of communities of size k. At the case of CPM the column k refers to the parameter of the algorithm instead, that is the algorithm was run for $k = 3, 4, \ldots$ The number of clusters were determined by a modularity maximization algorithm (a version of Newman), see the next subsection.

Graph and Method	#C	#CO	3	4	5	6	7	> 7
G_1 / CPM	10	7	7					
G_1 / N^{++}	10	9	5	0	0	2	1	1
G_2 / CPM	9	17	13	4				
G_2 / N^{++}	9	22	8	7	2	4	1	0
H_1 / CPM	6	10	7	3				
H_1 / N^{++}	6	37	5	2	3	9	7	12
H_2 / CPM	6	24	4	8	6	6		
H_2 / N^{++}	6	26	8	3	2	5	1	7

[3]For intersection graphs the CPM gives too large communities sometimes. A possible remedy is to fix the diameter, like in N^{++}.

3.4 Modularity.

The Newman modularity [20] is the following function of a graph G and its partition:

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j),$$

where $m = |E(G)|$, A_{ij} is the adjacency matrix of G, k_i is the degree, c_i is the cluster of the ith vertex, and $\delta(c_i, c_j)$ is the Kronecker symbol. The clustering algorithms may be based on some mathematical/physical heuristics like edge-betweenness (EB), eigenvectors (EV), label propagation (LP), spin glass (SG), walk trap (WT), or try to maximize the modularity function itself on the set of all partitions with a greedy algorithm (Gr). The formula can be generalized to communities [19]. One write s_{ij} instead of $\delta(c_i, c_j)$, where s_{ij} is an arbitrary similarity measure between vertices i and j. (In [19] the u_i is a probability distribution of i over the communities, and $s_{ij} = \langle u_i, u_j \rangle$, but it could be $\|u_i - u_j\|$ form any norm.) On the other hand, it is possible to get communities by maximizing the modularity function. The findings of [16] show the cluster and community structure cannot be measured on the same scale, some additional weighting must be introduced to solve this. The algorithms were tested for some graphs, we illustrate the results on the already mentioned Zachary graph. The clusterings are followed the Clique Percolation (CPM) with clique sizes $k = 3$ and $k = 4$, and N^{++} with its default parameters. The running time is in seconds, $\#C$ stands for the number of clusters or communities, whatever it applies.

Method	Modularity	Running time	#C
EB	0.4013	0.0100	5
EV	0.3727	0.0000	3
Gr	0.3807	0.0000	3
LP	0.4020	0.0000	3
SP	0.4063	1.1500	6
WT	0.4198	0.0000	4
CPM 3	0.2438	0.012	3
CPM 4	0.2557		3
N^{++}	0.1947	0.6690	12

One can evaluate cluster/community algorithms in indirect ways. That is by taking a problem in which the communities might have predictive value, and check the usefulness of these. We have observed dependencies among functions in some social graphs (telecommunication, friendship, Erasmus contracts etc.), and practically all methods provided useful hints. However, here the use of communities greatly outperforms the methods which use only clusters.

3.5 Refinements, time and orders.

One can conduct similar studies like the graphical method if have some functions that are defined on the vertices or the edges. Again, we have seen some highly subjective but still robust phenomena that might deserve to be mentioned.

First of all, the clusters are usually much bigger than the communities, and their number is less.

The number of communities might follow power law, although even to test this is impossible.

The communities are usually within the clusters, and give a fine structure of those larger classes. However, the reverse direction is also detected, the clusters might give information on communities. To be more precise, the most interesting communities are those ones in which elements belong to several clusters.

In social graphs we confirmed the role of the weak links described in [14], and also tested the different algorithms. The communities given by N^{++} are containing strong edges almost exclusively, while most of the weak edges are among communities. On the other type of small world graphs, the so-called technical graphs[4] there are no such effects. We used data from [17]. (The CPM does not give good results with any k, perhaps its performance is too sensitive to the measurement errors, missing data.)

The social graphs might have natural vertex attribute, the time when a vertex has been joined to the net. This order may not be manifested in the clusters if one considers the whole graph, but shows remarkable coincidence when restricting the graph to the neighborhood of a fixed vertex. In that case the clusters usually can be interpreted with some interval of time or spatial restrain. Note, that communities may cross the borders of clusters.

3.6 Weights.

Dealing with weighted graphs is difficult. It turns out that for the indirect methods the numerical results are more reliable. While all these methods can be extended to weighted graphs, the performance of them is little known [7].

In the rest we outline a model which is an example for indirect evaluation. The infection models are central in applications of real graphs [4], but to build appropriate ones is far from being trivial. The main points are (i) which model to choose, (ii) what are the significant variables and (iii) how to decide the values of the parameters. Our investigations concentrated on two problems in corporate banking, default (failing in paying debt) [9] and delay (in paying debt) [10]. We have to stress, although the two problems look similar, there are subtle differences. The main similarity is that these processes can be considered as some kind of "infectious disease." However, one has to be careful since financial difficulties may come from intrinsic reasons. (The rise or fall of the economic might be accounted by taking a fictitious node.) So the task is to devise a methodology that, given the *a priori* probabilities of some problem (say the default), estimates the *a posteriori* probabilities. The difference of these probabilities is recognized as *network effect* in the certain problem. The characteristics of the problem (e. g. no recovery, the probability of transmission is not constant) exclude the SIR or SIS models that play central role in Epidemiology. The best suited model is the Independent Cascade.

[4]In social graphs the presence of edges (x, y) and (x, z) increases the conditional probability of the the edge (y, z), while in the technical graphs this probability is decreased in that case.

3.7 Independent Cascade Model (IC)

This model is due to Domingos and Richardson [12], but an equivalent is in [15]. Here an edge weighted graph G is given, where to the edge (v, w) a probability $p_{v,w}$ is associated. The process of infection goes as follows. In the 1st step the set of infected vertices F_1 considered active, that is $F_1 = A_1$. In general for a vertex $w \in V(G) \subset F_{i-1}$ gets infected with probability $p = \prod_{v \in A_{i-1}} p_{v,w}$, and in that case $w \in F_i$. Note that the infected vertices may transmit the disease only in the very next step, that is $A_i = F_i \setminus F_{i-1}$. If for an i $F_i = F_{i-1}$, then the process halts.

3.8 Weighting and optimization.

First of all, one has to modify the IC model for effective use. Since the probabilities are assesses by simulation, it is natural to subject the a priori infection to this, too [9]. While the modified IC model provides extreme flexibility for modeling complex system, it is also very difficult to find appropriate transmission probabilities, or even an measure that tells from the better from the worse. The weights are assigned by a standard AI method, making a training set and a test set on the past data. A possible solution for the measurement is the use of the *gain curve*.

The vertices of the graph are ordered monotone decreasing way by their infection computed on the training set. Let $w_1, \ldots, w_n$ be the values of the same vertices given in the test set. Then the function $\text{gain}(x) = \sum_{i \le x} w_i / \sum_{i=1}^{n} w_i$; and $\int_{x=1}^{n} \text{gain}(x)dx$ should be maximized.

An estimation for an edge probability $p_{v,w}$ is based on the vertex and edge attributes that are available in the data. To maximize the performance measured by the gain function, a systematic search was done to try out the possible combinations of the reasonable functions of the considered variables. This included linear, quadratic, logarithm, exponential and sigmoid functions. The final aggregation of these transformed values was also treated this way. To find the best parameters of this function, a grid search was used.

3.9 Results.

Here we single out only one experiment out of several ones. A thorough study was executed on the data of one of the largest Hungarian bank (OTP), and the findings published in [10]. Here the estimation of default probabilities of certain clients (small and middle enterprise sector) was the goal. The OTP Bank Corporate transaction database was used, where the graph building period was from August 2008 to April 2009 (6 months) and the infection period was from February 2009 to April 2009 (the last 3 months from it). For default event observation two periods were chosen: a longer one from May 2009 to April 2010 (12 months), and a shorter one from May 2009 to April 2010 (3 months).

I. It turned out that shorter periods (3 month) gave better models than those were based on longer periods.

II. The direction of the edges counts, it should be taken as buyer - provider, i. e. if x sends money and y receives it than (x, y).[5]

[5]There is some effect even when the edges are taken indi-

III. The variables and findings worth considering follow.

(i) Community information. (If the edge belongs to a community?)

(ii) The edge (x, y) inherits the variables of x (but not y).

(iii) Relative traffic, that is the transfer of the edge divided by the sum of all incoming transfer.

(iv) The age of the client. (How old is the company?)

(v) Behavioral types. (queuing on the account, overdraft etc.)

Even though, the most significant variables are the ones listed in (i) and (iii).

Based on this, we found an expected 3-4 to even 10-12 times lift in the different segments [10]. The fact that a vertex x is in a same community with an infected increases the chance of x's infection by a factor three. Note that there were similar findings in [9] on different data. However, in [9] the parameter values for the IC based model were set by using trial and error, while in [10] a more sophisticated search was done. The computations were carried out by the use of Sixtep software.

4. ACKNOWLEDGMENTS

The first author was partially supported by the the joint grant of Hungarian Government and the European Union in the framework of the Social Renewal Operational Programme, project no. TÁMOP 4.2.2-08/1-2008-006, while the third author was partially supported by the Hungarian National Science Foundation Grants OTKA K76099 and also by the grants TÁMOP-4.2.2/08/1/2008-0008 and TÁMOP-4.2.1/B-09/1/KONV-2010-0005.

5. REFERENCES

[1] B. Adamcsek, G. Palla, I. J. Farkas, I. Derényi, and T. Vicsek. Cfinder: Locating cliques and overlapping modules in biological networks. *Bioinformatics*, 22(8):1021–1023, February 2006.

[2] R. Albert and A. L. Barabási. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[3] R. Albert and A. L. Barabási. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74(1):47–97, January 2002.

[4] M. Boguná, R. Pastor-Satorras, and A. Vespignani. Absence of epidemic threshold in scale-free networks with connectivity correlations. *arXiv:cond-mat/0208163v1 [cond-mat.stat-mech]*, 2002.

[5] B. Bollobás. *Modern Graph Theory*. Springer, New York, New York, 1998.

[6] B. Bollobás and O. Riordan. Clique percolation. *Random Structures and Algorithms*, 35(3):294–322, October 2009.

[7] A. Bóta. Applications of overlapping community detection. In $(CS)^2$ - *Conference of PhD Students in Computer Science*, page . , June 2010.

[8] A. Cami and N. Deo. Techniques for analyzing dynamic random graph models of web-like networks: An overview. *Networks*, 51(4):211–255, July 2008.

[9] A. Csernenszky, G. Kovács, M. Krész, A. Pluhár, and T. Tóth. The use of infection models in accounting and crediting. In *Challenges for Analysis of the Economy, the Businesses, and Social Progress*, pages 617–623. , November 2009.

[10] A. Csernenszky, G. Kovács, M. Krész, A. Pluhár, and T. Tóth. Parameter optimization of infection models. In $(CS)^2$ - *Conference of PhD Students in Computer Science*, June 2010.

[11] L. Csizmadia. *Recognizing communities in social graphs*. MSc thesis, University of Szeged, Hungary, 2003.

[12] P. Domingos and M. Richardson. Mining the network value of costumers. In 7th Intl. Conf. on Knowledge Discovery and Data Mining, pages 57–66. ACM, August 2010.

[13] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75–174, February 2010.

[14] M. Granovetter. The strength of weak ties. *American Journal of Sociology*, 78(6):1360–1380, May 1973.

[15] M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, May 1978.

[16] E. Griechisch. Comparison of clustering and community detection algorithms, the extension of the modularity. In $(CS)^2$ - *Conference of PhD Students in Computer Science*, June 2010.

[17] C. A. Hidalgo, B. Klinger, A. L. Barabási, and R. Hausmann. The product space conditions the development of nations. *Science*, 317(5837):482–487, July 2007.

[18] J. Kleinberg. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems (NIPS)* **15**, 2002.

[19] T. Népusz, A. Petróczi, L. Négyessy, and F. Bazsó. Fuzzy communities and the concept of bridgeness in complex networks. *Physical Review E*, 77(1):016107, January 2008.

[20] M. Newman. The structure and function of complex networks. *arXiv:cond-mat/0303516 v1*, 2003.

[21] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(9):814–818, June 2005.

[22] A. Pluhár. On the clusters of social graphs based on telephone log-files. *Research Report*, 2001.

[23] A. Pluhár. Determination of communities in social graphs by fast algorithms. *Research Report*, 2002.

[24] D. Stark. The vertex degree distribution of random intersection graphs. *Random Structures and Algorithms*, 24(3):249–258, May 2004.

[25] T.S.Evans and R.Lambiotte. Line graphs of weighted networks for overlapping communities. *arXiv:0912.4389v2 [physics.data-an]*.

[26] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

rected. This is due to another network effect, since the edges are describing an economic interdependence/same sector.

GREEDY HEURISTICS FOR DRIVER SCHEDULING AND ROSTERING

Viktor Árgilán, Csaba Kemény, Gábor Pongrácz, Attila Tóth
Institute of Applied Sciences
Juhász Gyula Faculty of Education
University of Szeged
Boldogasszony sgt. 6, 6725 Szeged, Hungary
{gilan, kemeny, pongracz, attila}@jgypk.u-szeged.hu

Balázs Dávid,
Institue of Informatics
Faculty of Science and Informatics
University of Szeged
Árpád tér 2., 6720 Szeged, Hungary
davidb@inf.u-szeged.hu

ABSTRACT

In this paper, we introduce greedy methods for both the driver scheduling and the driver rostering problems. The methods were created to be efficient parts of a decision support system in public transportation meeting the demands of an industrial application by being flexible with adjustable settings. The developed algorithms provide good results for real life instances in acceptable time.

Keywords

crew scheduling, crew rostering

1 INTRODUCTION

Driver scheduling and rostering are important problems arising in public transportation. Based on a feasible vehicle schedule, driver scheduling determines the shifts for each day of the planning period. This is followed by driver rostering, where the shifts are assigned to drivers. All these steps are limited by different rules and constraints, usually defined by several authorities (government, EU, etc.).

Both the driver scheduling and the driver rostering problems are NP-hard, but several different approaches have been introduced in literature for the problems above [4]. Solving the problems by formulizing exact mathematical models [2], using constraint programming [3,5,9], or evolutionary algorithms [6,7,8] give good theoretical solutions. However, not all of the above methods can be applied efficiently in practice due to the size of the problem. The planning period we have to consider is usually 1-2 months, which makes the size of the problem even larger, than using the methods for only several days.

The above problems are usually considered as parts of decision-support systems [1], which allows users to create several different solutions for the problem. For this, they need to set different parameters, compare them, or even re-calculate some results of the system, meaning that the algorithms will be executed several times. Since real life applications of these problems in a decision support system require a running time of a couple of minutes only, the applied algorithms need to have a fast running time. In addition to this, the quality of their given solution is also an important aspect.

The above requirements lead to the introduction of heuristic approaches, which can result in an efficient solution in adequate time. Our goal was to introduce fast heuristic algorithms for both problems, which work effectively on real life data, giving a "quasi-optimal" solution. Greedy heuristics are classical methods of algorithm design, and are known to have an extremely fast running time. Though they usually fail to achieve global optimum, we will show through our test results that they can be applied efficiently on the driver scheduling and rostering problems, and give a good solution for real-life instances.

The structure of this paper is the following: first, we deal with the driver scheduling problem, followed by the driver rostering problem. In both cases, we first present the traditional IP formulation, and show why this type of approach is unusable on a real life instance. Choosing other models would have been possible, but the problems arising from the size of the IP model also exist at different representations as well. After this, we propose greedy methods for the problems. Finally, we present the results of the methods on a real life instance, and draw the conclusions.

The results of our algorithms are presented on instances provided by the Szeged City Bus Company (Tisza Volán ZRt, Urban Transport Division).

2 DRIVER SCHEDULING

The input of our driver scheduling method is a feasible vehicle schedule. This contains the properties of the assigned buses and the basic properties (starting time, ending time, starting place, ending place) of the vehicle tasks. Driver scheduling uses the tasks of these vehicle schedules to define shifts with minimizing the cost of assigning a task to a certain shift. The scheduling is always produced for a single day. Only those rules are considered which concern only the daily duty so the optimization works on the planning period sequently day-by-day.

Two tasks can be scheduled to the same shift, if they satisfy certain basic rules. The tasks cannot overlap in time, and two tasks scheduled after each other must be geographically sequential

(the ending location of the first one has to be the same as the starting location of the second one). If these locations are different, a new task has to be introduced between them to satisfy this geographical condition. However, there are other conditions for the problem (given by the EU, government or company regulations), which are more complicated.

The most important rules for a daily shift are the constraints of maximal working time, the length and frequency of the breaks, and the number and type of possible resting locations (longer breaks can only be spent at specific resting places). Other special rules may apply apart from the above mentioned ones, but these vary from company to company.

The objective function minimizes the number and cost of the shifts which give the total cost.

Literature usually discusses two different solution methods. The first is Generate and Selection Approach (GaS), which initially generates sets of the candidate shifts. The size of these sets may vary from a couple of 100.000 to even more then several 1.000.000. The shifts, giving the best solution, are selected from these sets. This problem can be reduced to the set covering or set partitioning problems, and formulized as the following:

Minimze

$$w_1 \sum_{j=1}^{n} c_j x_j + w_2 \sum_{j=1}^{n} x_j$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, i = 1, 2, \ldots, m$$

$$x_j \in \{0,1\}, j = 1, 2, \ldots, n$$

where

n = the number of candidate shifts

m = the number of tasks

x_j = shift variable, x_j = 1 if shift j is selected and 0 otherwise

c_j = cost of shift j

a_{ij} = 1 if tasks i is covered by shift j, 0 otherwise

w_1 and w_2 are weights

There are several efficient methods to solve the problem above. However, the use of this approach for a real-life instance (like in our case: Szeged, Hungary: 170.000 inhabitants, ~2700 trips on a usual workday) results in several million different shifts for a day. In addition to this, the applied rules also increase the complexity further. Because of the above observations, finding a feasible solution is really hard. The number of candidate shifts is large, and generating the problem is a slow process because of the high number of rules. Solving the IP problem above with such a size also requires significant running time.

The other solution method that can be used for the problem is the Constructive Approach. Instead of generating all the possible shifts, this method constructs a single solution for the problem, while it tries to minimize the cost. Our algorithm uses this approach.

The main requirement we considered when we created our method was that the solution algorithm should be adjustable according to the needs of the user, with modifyable methods of solutions using different settings and parameters. The basis of our method based on these observations is a controlled greedy approach, which can give an adequate, feasible solution in a couple of seconds, according to the needs of the user.

The main idea is to divide the driver scheduling into sequential substeps, where different optimization methods can be used and combined. By this, good flexibility can be reached, since it is quite easy to adjust the process to the expectations. It is even possible to improve the output of a step before the next one by hand, as sometimes a professional human mind can see what an optimization algorithm could not find.

The process goes through the following steps:

1. Creating work-pieces. This makes work-pieces by pulling the tasks together by the possiblity of the driver reliefing with different rules.
2. Creating pre-shifts. This generates initial solutions, called pre-shifts, ommiting the difficult rules.
3. Cutting. This step splits the pre-shifts into smaller feasible parts considering all the rules.
4. Joining. In this step the shift parts are joined together producing longer feasible shifts.
5. Refining. It improves the cost of the solution by changing work-pieces between the shifts.
6. Finishing. Finally this step inserts all required duties into the shift and define an idle activity for all idle time measuring the final cost of the shifts.

The aim of step 1 is to reduce the size of the problem. There are consecutive vehicle tasks, that have to be carried out by the same driver (because there is no time between the tasks to change drivers, or the tasks simply belong together, etc.), meaning that these can be replaced by one driver task with starting parameters of the first task and ending parameters of the last task. Using this, we can define a work-piece as one or more consecutive tasks of a vehicle carried out by the same driver. Determining work-pieces can be done with a simple method in linear time.

Step 2 generates the initial shifts. The main idea is to schedule all the work-pieces into shifts, while we consider only some basic constraints (eg. the resulting shifts cover all the events without overlapping), but the shifts do not necessarily satisfy all the given rules. These shifts are well-optimised, and can be generated in reasonable time. The following steps of our process transforms them to get a feasible solution regarding all given rules.

The task of step 3 is to create feasible pieces considering the rules. The infeasibility of a pre-shift can be derived mostly from two reasons. The first is the problem with its length, meaning that the pre-shift is too long and conflicts the constraints of maximum working time or minimum resting time. The other reason is that there is not enough time inside the pre-shift for breaks as defined by the rules. Both problems can be solved by cutting the pre-shifts into smaller parts. At this point, several strategies can be applied,

which are based on the length, structure, cost or on the traffic load of the different periods of the day. The main point of the previous two steps is to generate effectively shift parts which are useful producing feasible shifts.

Step 4 joins the resulting shift parts together, applying also different strategies. Since the resulting shifts of the previous phase are all feasible with respect to every rule, they must also remain feasible after a join. For measuring the fitness of the joining of two shifts a new joining-fitness function is introduced. The fitness of a joining depends on two main factors:

- how close the two parts are to each other in time (the difference of the starting time of the second shift and the ending time of the first shift)
- wheter the starting location of the second shift and the ending location of the first shift are the same, or different (if they are same, then a connecting activity is not needed, and the driver does not change vehicle, so the vehicle used on the last task of the first part and the vehicle on the first task of the second part are the same)

The joining of two parts can be executed with two different greedy methods, or also according to the load of the different periods of the day.

For step 5 a complete shift is given as an input, so the number of shifts is not changed further at this step. However, the cost of the scheduling can be decreased by exchanging a pair of proper work-pieces from two different shifts. This type of change is made only if the sum of the costs of the two shifts can be reduced this way. Rules also have to be considered, these changes cannot violate them in any shifts. Refining can also be executed by using several different methods, like a local search or globally examining all shifts.

Step 6 is a deterministic method, which inserts other events defined by the rules into the shifts. The resulting shifts of the first 5 steps only contain vehicle tasks, and tasks where the driver travels between the geographical locations of the vehicle tasks. So until this point the optimization is quite general and more or less independent of the local circumstances. However, a final shift must contain all the obligatory duty defined by the company, and also all idle time of the driver must be assigned to an acticity. We first assign those tasks, which have to be carried out at a fixed time in the schedule of a driver (eg. passengers getting on and off exactly before and after a run). After the assignment, the remaining activities are inserted which have no specified time interval in which they have to be carried out (break, administration, maintenance, etc). Finally, special idle-tasks are assigned to all remaining unused inner time of the shift depending on its length.

During the optimization process described above, the total cost of a shift cannot be computed before step 6. A shift has to contain many different driver activites in addition to vehicle tasks (administration, getting on/off the vehicle, etc.). These can only be determined once the shifts are complete, and an actual driver is assigned all his daily shifts. Because of this, only the vehicle tasks, travelling time of the driver and idle time of the driver are taken into consideration during the driver scheduling phase. A cost is assigned to every gap between every two consecutive tasks by statistical analysis. The cost of these periods is calculated using the present practice of driver shifts used by the company.

Let T be the set of vehicle tasks, A be the set of other activities ($T \cap A = \emptyset$), S' be the scheduling of the company and $s' \in S'$ a shift. An idle time $pn \subset s'$ is a time period of the shift defined by two consecutive vehicle tasks t_i, t_{i+k} containing other driver activities such that $s' = (\ldots, t_i, a_{i+1}, a_{i+2}, \ldots, a_{i+k-1}, t_{i+k}, \ldots)$ where $t \in T$, $a \in A$ and $pn = (a_{i+1}, a_{i+2}, \ldots, a_{i+k-1})$. Then the category of this period defined by its length:

$$p^{rank} = \left\lfloor \frac{t^{st}_{i+k} - t^{et}_{i}}{5} \right\rfloor$$

where t^{st} is the starting time and t^{et} is the ending time of vehicle task t. This means a rank in a 5 minutes time scale R. The cost of this period is calculated by summing the cost of the inner activities:

$$p^{cost} = \sum_{j=i+1}^{i+k-1} a_j^{cost}$$

where a^{cost} is the cost of activity $a \in A$.
The cost of a time period R_m in the scale calculated by the average of the cost of the idle periods p_k categorized to this rank.

$$R_m^{cost} = \frac{\sum_{k=1}^{|R_m|} p_k^{cost}}{|R_m|}$$

where each $p_k^{rank} = m$.

Using this scale the cost of a shift calculated as a weighted sum of the vehicle tasks, the traveling and the idle periods.

$$c(s') = \alpha \cdot \sum_{t \in T} \left(t^{et} - t^{st} \right) + \beta \cdot \sum_{a \in A'} \left(a^{et} - a^{st} \right) + \gamma \cdot \sum_{i=1}^{|s'|-1} R_{p_i}$$

where $A' = \{a: atype = \text{TRAVELLING}\}$ is the set of the travelling activities, p_i-s are the idle periods in shift $s' \in S'$ and α, β, γ are weights.

The final cost can be calculated for the shifts after step 6 of the process. Since every shift activity and idle time has a cost category defined by the company, the cost of a shift is the sum of the costs of the events it contains.

$$c(s) = \sum_{i=1}^{|s|} e_i^{cost}$$

where $e \in T \cup A$ is an event in shift s.

Notice that at this point we do not take into consideration the overtime and undertime. To measure the over- or undertime of a driver we need the driver rostering. The average daily working time of a driver is calculated according to his contract and the time interval over which we are scheduling (two months in our

case). That is why the quantity of the over- and undertime depends on the rostering and not the scheduling of a single day.

Besides the adjustable parameters (e.g. the required length of a break) the flexibility of the method is mainly provided by steps 2, 3 and 4, where the user can combine different methods, and manually alter the result of the different methods after every step. Since the algorithm solves the driver scheduling problem for one day in a few seconds, solving the problem for the whole planning period takes only a several minutes.

3 DRIVER ROSTERING

The input of the driver rostering problem are shifts which are given as the output of the driver scheduling. The output is a driver roster. The algorithm also needs certain information about the drivers and the driver rules.

Driver constraints are given rules and regulations (mostly by the EU, and the local government). The most important of these rules are:

1. The number of maximum weekly working hours of a driver must be respected.
2. The required minimum weeky resting time of a driver must be respected.
3. The number of minimum resting hours of a driver between two consecutive shifts must be respected.
4. Drivers cannot work consecutively for more than k days within a given timeframe.
5. The number of minimum resting days within a mont must be respected.
6. Each driver has to be assigned at least r number of Sundays as resting days in each month.

The shifts are given by a driver scheduling method for the total planning period, and each shift has to be assigned to exactly one driver, considering the rules introduced above. A planning period can be of arbitrary length, either a couple of weeks or months, or even an entire year.

We introduce two main approaches to driver rostering: first, we examine a traditional IP model of the problem, and analyze its constraints. Then a greedy rostering method is introduced for the driver rostering problem, which solves the problem with a fast running time, and gives a good quality solution.

3.1 IP approach

The problem can be represented as an IP model, where all shifts have to be executed by a driver exactly once, and the constraints regarding drivers can not be violated. The objective function of the problem also tries to minimize the number of drivers, the under- and overtime. The driver rules give the constraints of our model, and the objective function minimizes the sum of the cost of the rostering.

Let

$$S = \{s_j\}_{j=1}^{n}$$

be the set of shifts for the whole planning period, S_m the set of shifts of month m, S_w the set of shifts of week w, and S_d the set of shifts of day d, $S_d \subset S$, $S_w \subset S$, $S_m \subset S$, where n is the number of shifts of the planning period. Let

$$S = \{s_i\}_{i=1}^{t}$$

be the set of all drivers, where t is the number of drivers.

Let $\phi: S \to R^v$ be a function which describes the attributes of the shifts using a real vector of lenght v, and let D be the set of drivers and function $\varphi: D \to R$ determine the contract type of each driver. The problem is to find $h: S \to D$ assignment where we minimize the cost:

$$c\left(h\left(s\right)\right) \to \min .$$

For an assignment h let

$$x_{ij} = \begin{cases} 1, \text{if h}(s_j) = d_i \\ 0, \text{otherwise} \end{cases}$$

where i denotes the driver index, and j denotes the shift index.

Using the rules described above, the IP model of the driver rostering is:

$$\forall i \forall w \sum_{s_j \in S_w} wt(s_j) \cdot x_{ij} \le maxweekwt \cdot 60 \quad (1)$$
$$(1 \le i \le t, 1 \le w \le n_w)$$

where $wt(f_j)$ is the working time of shift f_j given in minutes, n_w is the number of shifts in week w, and $maxweekwt$ is the maximum allowed weekly working hours for a driver.

$$\forall i \forall w \; 7 \cdot 24 \cdot 60 - \sum_{s_j \in S_w} wt(s_j) \cdot x_{ij} \ge minweekwrt \cdot 60 \quad (2)$$
$$(1 \le i \le t, 1 \le w \le n_w)$$

where $minweekrt$ is the minimal weekly resting time of a driver given in hours. The product 7·24·60 gives the length of a week in minutes.

$$\forall i \sum_{\substack{\forall s_j, s_k \in S \\ j \ne k \\ st(s_k) - et(s_j) < minshiftrt \cdot 60}} x_{ik} x_{ij} = 0 \quad (3)$$

where $minshiftrt$ is the number of minimum resting hours of a driver between two consecutive shifts. The starting time of the shift is denoted by $st(s_k)$ while the ending time of the shift is denoted by $et(s_j)$.

$$\forall i \forall d \sum_{\forall s_j \in \bigcup_{k=d}^{d+contwd} S_k} x_{ij} \leq contwd \quad (1 \leq i \leq t,\ 1 \leq d \leq n - contwd) \tag{4}$$

where *contwd* is the maximum number of consecutive working days allowed in a given term.

$$\forall i \forall m \sum_{\forall s_j \in S_m} x_{ij} \leq \text{days}(m) - minmonthrd \quad (1 \leq i \leq t,\ 1 \leq m \leq numofmonths) \tag{5}$$

where days(m) gives the number of days in month m, *minmonthrd* gives the number of minimum resting days in a month, and *numofmonths* gives the number of month of the planning period.

$$\forall i \forall m \sum_{\substack{\forall s_j \in S_m \\ s_j \in \bigcup S_q \\ q\ \text{Sunday}}} x_{ij} \leq \text{sundays}(m) - minnumofsunday \quad (1 \leq i \leq t,\ 1 \leq m \leq numofmonths) \tag{6}$$

where q denotes the Sundays in the given month, sundays(m) gives the number of Sundays of month m, and *minnumofsunday* is the minimum number of Sundays that are needed to be assigned as resting days each month.

Given the constraints above, our objective function is:

$$\sum_{i=1}^{t} \sum_{j=1}^{n} c_{ij} \cdot x_{ij} \rightarrow \min$$

where c_{ij} is the cost of driver i executing shift j.

Solving the above model by using an IP solver would give the optimal solution for the problem. However, a huge disadvantage of this method is that the running time of the solver grows exponentially to the size of the problem. Because of this, a data instance taken from a real-life situation can have a running time up to several months (a problem of such magnitude has several million variables). On the other hand, a decision support system can not apply a solution method with such a slow running time. This is not acceptable, as companies would like to execute the methods several times using different parameter settings, and compare the results to choose the driver roster that fits their situation the best.

To satisfy the above requirements, a different method is needed, which gives a near-optimal solution, but has an extremely fast running time. As mentioned before, greedy-based heuristics are known for these propterties.

3.2 Greedy algorithm

Greedy algorithms are usually known to provide solution faster for our problem than other optimization methods. Though it comes with the cost that the greedy choice property guarantees only local optimum, but a good pre-processing of the input, especially a proper selection method for the initial set of drivers can improve the global solution too.

Initially the driver set is defined. Each driver has a contract type which defines the working hours for a day in average of a given time interval (in our case it is two months). Our method uses an initial set of drivers calculated by a seperate algorithm. This gives an estimate driver number, considering the existing different types of driver contracts and uses the number of employed drivers for the given day as an upper bound As a first step, depending on their total working time, the shifts can be divided into sets, which helps in giving this estimation. Usually the resulting number helps us to achieve a good quality solution, but this estimation can be greater then the actual number of drivers available for the company. The second step uses the ratio between these different contract types. The estimate values are modified, using pre-set upper bounds: if the value is greater than its upper bound, then it is decreased to the upper bound, and their differenc is converted to a different contract type (either with greater, or with smaller working hours) which can be increased. The ratios of conversion are in case of 4-, 6- and 8-hours contracts:

- 2 contracts of 4 hours can be converted into 1 contract of 8 hours;
- 3 contracts of 4 hours can be converted into 2 contracts of 6 hours;
- 4 contracts of 6 hours can be converted into 3 contracts of 8 hours.

These conversions can be applied both ways, eg. if the maximum number of drivers with 8-hour contracts is exceeded, they can be converted either to drivers with 4-hour or 6-hour contracts. Considering the total daily working time, the above mentioned rules can be applied, as they are all equivalent conversions. This means that for example an 8-hour long shift can either be executed by a driver with a contract of 8-hours, or 2 drivers with a contract of 4-hours. The resulting number now can be applied as an initial input for the number of drivers used by the greedy heuristic.

Then the rostering algorithm iterates through every day of the planning period. Each day, the daily shifts are assigned to feasible drivers, who are chosen according to a certain strategy (fitness, random).

Let S be the set of shift, and D the set of drivers. Our greedy algorithm for the driver scheduling problem is:

```
GREEDY_DR(S,D)
    for days i=1..n
        S'=unassigned shifts of day(i)
        while S'≠∅
            s_j=Select_shift(S')
            D'=drivers of D, who can execute s_j
            d_k=Select_driver(D',s_j)
            assing s_i to d_k
            S'=S'\{s_j}
        end while
    end for
```

The method enables the user to choose between several selection strategies of the shift (Select_shift) and the driver (Select_driver)

executing the chosen shift. For example the shifts can be selected randomly or sequently by the calendar, and the drivers can be selected randomly, with a first fit approach, or by using a fitness function minimizing the over and undertime to determine the best driver for the shift.

Our experience shows that a cost value of driver number combinations in the search space more or less strictly decreasing, while the minimal area is well determinable and the direction of the decrease is easily recognizable in every point. The estimate values given by our special method in the initial step is close to this minimal area. Executing the greedy algorithm (GREEDY_DR) above sequentially, using a neighbour step in the driver combination (increasing the number of drivers in a contract type in case of overtime and decreasing in case of undertime) with an expected lower cost can lead us to a solution closer to the optimum.

4 TEST RESULTS

The presented methods have been implemented and tested in Matlab using a general pc, implementations under different programming environments might give slightly different results.

The running time of the vehicle scheduling method is a few seconds for a daily schedule, which adds up to 2-3 minutes to a planning period of 2 months. It can be improved by using parallel execution. The driver rostering method has a running time of around 2 minutes, which gives us a total running time around of 4-5 minutes (depending on the problem size).

We have tested our methods on instances provided by Tisza Volán Zrt. The present practice of the company uses 264 drivers for this planning period. Our results are summarized in Table 1 (the number of drivers, and the differences between the drivers average daily working hours and their required working hours by their contract type).

Table 1. Driver numbers and average working hour differences from the contract types. Test cases show different contract type settings.

	Number of drivers	min. difference (h)	max. difference (h)	average difference (h)
Test1	225	-0,05003	-0,00223	-0,01164
Test2	228	-0,1052	0,07445	0,000724
Test3	223	0	0,0626	0,013213
Test4	231	0	0,10063	0,026401
Test5	226	-0,09693	-0,00023	-0,02735
Test6	228	0,0032	0,054267	0,017737

It can be clearly seen, that the methods resulted in a significant decrease in the number of drivers, while the average daily working hours of a driver always stay close to the contract type of the driver. The table only shows the total number of drivers for each instance, however, this is always a sum of the drivers with different type of contracts.

Each instance stands for a driver schedule with different cut and join methods, for which we determined the inital set of drivers, and executed some iterations of the driver rostering algorithm until it improves the cost. The results we get after the last iteration are the solutions of the problem. All values in the columns are averages, which are calculated using the following method: the monthly working time of each driver is divided by the number of workdays in that month, resulting in a daily average working time. Table 1 contains the minimum, maximum and average differences from the corresponding contract types. Negative numbers represent undertime, while positive numbers represent overtime.

All test results (except for Test2) contain drivers of all 3 contract types (8, 6 and 4 hours). The number of drivers with 6-hour contracts is 4 in all cases, while the number of drivers with 4-hour contracts greatly vary: 7 in Test4, 4 in Test3, 2 in Test5, 1 in Test1 and Test6, and 0 in Test2.

The company always has higher number of drivers in their roster, than they actually need. This way they always have enough drivers, when needed (e.g sending people on vacation, or unforeseen events).

5 CONCLUSION AND FUTURE WORK

This paper shows our results for solving the problems of both driver scheduling and driver rostering in public transportation. The algorithms we presented for the problems are both based on the greedy strategy which guarantees a fast running time, while the algorithms themselves give promising results on real-life instances. The algorithm can execute different optimization strategies through the process, and then compare the solutions accordingly. This flexibility makes it more useful in industrial usage.

Further improvement of the results is possible, using well chosen heuristic techniques on the output of the driver rostering algorithm. Currently a local-search based heuristic is being perfected and tested which is able to correct a shift if it conflicts a rule. This heuristic introduces a cost-function to every crucial rule of the driver rostering, with certain weights depending on the importance of the rule, and our goal is to minimize these costs. The heuristic examines each day of the planning period, and is able to swap the shifts of any two drivers on the given day. The shifts are swapped, if it decreases the overall cost of the rostering, and the algorithm has several termination criteria.

The introduced greedy methods are part of a research and development project, currently in the research phase. The Matlab implementation is mainly for research purposes, the methods will be implemented in the final project using a C++ enviroment.

6 ACKNOWLEDGMENTS

This paper was partially supported by Szeged City Bus Company (Tisza Volán, Urban Transport Division).

7 REFERENCES

[1] J. Békési, A. Brodnik, M. Krész, D. Pas: An Integrated Framework for Bus Logistics Management: Case Studies. *Logistik Management* 5., pp. 389-411, 2009.

[2] P. Cappanera, G. Gallo: A Multicommodity Flow Approach to the Crew Rostering Problem, *Operations Research* 52. pp. 583-596. 2004.

[3] A. Caprara, F. Focacci, E. Lamma, P. Mello, M. Milano,P. Toth, D. Vigo, Integrating constraint logic programming and operations research techniques for the crew rostering problem, *Software Practice and Experience* 28. pp. 49–76. 1998.

[4] A.T. Ernst, H. Jiang, M. Krishnamoorthy and D. Sier, Staff scheduling and rostering: A review of applications, methods and models, *European Journal of Operational Research* 153. pp. 3–27. 2004.

[5] N., Guerinik, M.V.m Caneghem: Solving Crew Scheduling Problems by Constraint Programming. *In Proceedings of the 1st Conference of Principles and Practice of Constraint Programming* pp. 481–498. 1995.

[6] M. Moz, M. Vaz Pato: A genetic algorithm approach to a nurse rerostering problem, *Computers & Operations Research* 34. pp. 667–691. 2007.

[7] Moz, M., Respcio, A., and Vaz Pato, M: Bi-objective Evolutionary Heuristics for Bus Drivers Rostering, *Working paper 1-2007, Centro de Investigaao Operacional,* Universidade de Lisboa, 2007.

[8] K. Nurmi, J. Kyngäs, G. Post: Driver Rostering for Bus Transit Companies. *Engineering Letters* 19:2, 2010

[9] R.S.K. Kwan, A.S.K. Kwan, and A. Wren: Evolutionary driver scheduling with relief chains. *Evolutionary Computation* 9. pp. 445–460. 2001.

A note on context-free grammars with rewriting restrictions*

Zsolt Gazdag
Department of Algorithms and their Applications
Faculty of Informatics
Eötvös Loránd University
Pázmány Péter sétány 1/C
H-1117 Budapest, Hungary
gazdagzs@inf.elte.hu

ABSTRACT

In this work two special types of random context grammars are investigated. In one of these grammars one can apply a context-free rule to a nonterminal A only if the current sentential form does not contain a particular nonterminal associated with A. The other grammar works in the opposite direction, it requires the presence of the associated nonterminal to apply the rule. Here we show that these systems can generate non-context-free languages.

1. INTRODUCTION

Context-free grammars are among the most investigated topics in formal language theory. They were used first in the study of human languages, but later it turned out that the specification of programming languages can also be modelled by these grammars. On the other hand, it is well known that context-free grammars cannot cover all aspects of natural and programming languages. Therefore several types of grammars that are based on context-free rules but have larger generative power were introduced. This larger generative power is often achieved by adding a regulated rewriting mechanism to context-free grammars. Representatives of these grammar systems are, for example, random context grammars (see e.g. [3]). In these systems two sets of nonterminals, a permitting and a forbidding one, are associated with every context-free rule. A rule of a random context grammar is applicable, if it is applicable in the context-free sense and

(1) nonterminals from the associated forbidding set do not occur, while

(2) every nonterminal from the permitting set does occur in the current sentential form.

This simple regulation makes these systems computationally complete if ε-rules are allowed. In [5] an even simpler regulation on the application of context-free rules was made and the resulting grammars were investigated. In this paper we call these grammars *context-free grammars with rewriting restrictions*.

These systems are in fact special random context grammars, but here the permitting and forbidding sets are associated with the nonterminals rather than the rules of the system. Moreover, one of these sets is always a singleton set and the other one is empty. In more detail, a context-free grammar with rewriting restrictions is essentially a context-free grammar G enriched with a function f which associates with every nonterminal of G a nonterminal and a sign symbol ($+$ or $-$). We say that a rule rewriting the nonterminal A is applicable if it is applicable in the context-free sense and, additionally, the following holds. If f associates with A the nonterminal B and the sign symbol $-$ (resp. the sign symbol $+$), then B does not occur (resp. does occur) in the current sentential form.

It is shown in [5] that the above described systems are still Turing-equivalent if the underlying context-free grammar is not ε-free. On the other hand, it is noted in [5], that these systems do not generate all recursively enumerable languages if the system is such that f associates the sign $-$ with every nonterminal. The same applies when f associates the sign $+$ with every nonterminal. Nevertheless, these special types were not further investigated in [5]. In this work we show that these special types can also generate non-context-free languages.[1]

2. PRELIMINARIES

We assume that the reader is familiar with the basic concepts of the formal language theory, such as alphabets, words,

*This research was supported by the project TÁMOP-4.2.1/B-09/1/KMR-2010-003 of Eötvös Loránd University.

[1]Here we note that, independently from us, an even stronger result was proved in [2], but only in the permitting case, i.e., when f associates the sign $+$ with every nonterminal. It was shown that in this case context-free grammars with rewriting restrictions are equivalent to permitting random context grammars, a subclass of random context grammars, where the forbidding sets are all empty. A poster with this result was presented at DLT 2010 conference [1] after the submission of our work.

languages, etc. For comprehensive guides to these topics the reader is referred to [4] and [6]. For an alphabet Σ, Σ^* denotes the *set of all words over* Σ. The *empty word* is denoted by ε.

A *context-free grammar* (CFG in short) is a 4-tuple $G = (V, \Sigma, R, S)$ where V and Σ are alphabets of *nonterminal* and *terminal symbols*, respectively (it is assumed that $V \cap \Sigma = \emptyset$), $S \in V$ is the *start symbol*, and R is a finite set of *production rules* of the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. The *derivation relation* of G is defined as follows. For all $u_1, u_2, v \in (V \cup \Sigma)^*$ and $A \in V$, we say that $u_1 A u_2 \Rightarrow u_1 v u_2$ if and only if there is a rule $A \to v$ in R. The *reflexive, transitive closure of* $\Rightarrow$ is denoted by $\Rightarrow^*$. The *language generated by* G is the language $L(G) = \{u \in \Sigma^* \mid S \Rightarrow^* u\}$. We say that a language is *context-free* (CF) if it is generated by a CFG. The family of all CF languages is denoted by $\mathcal{L}(\mathrm{CF})$.

A *context-free grammar with rewriting restrictions* (CFGrr in short) is a tuple $G = (V, \Sigma, R, S, f)$, where V, Σ, R, S are the same as in a CFG and $f : V \to \{+, -\} \times V$ is a function. If, for every $A \in V$, $f(A) \in \{-\} \times V$, then G is a *context-free grammar with forbidding rewriting restrictions* (CFGfrr, for short). Likewise, if, for every $A \in V$, $f(A) \in \{+\} \times V$, then G is a *context-free grammar with permitting rewriting restrictions* (CFGprr). Since in a CFGfrr (resp. CFGprr), for every nonterminal A, the first component of $f(A)$ is the symbol $-$ (resp. $+$), we will define in these grammars f as a function $f : V \to V$.

The *derivation relation* of G is defined similarly to the case of CFGs with the following additional restriction. For every $u_1, u_2 \in (V \cup \Sigma)^*$ and rule $A \to v \in R$, $u_1 A u_2 \Rightarrow u_1 v u_2$ provided that either

(1) $f(A) = (+, B)$ and B appears in $u_1 A u_2$, or

(2) $f(A) = (-, B)$ and B does not appear in $u_1 A u_2$.

The *language generated by* G is defined in the same way as in the case of CFGs.

Finally, we denote by $\mathcal{L}(\mathrm{CFGrr})$, $\mathcal{L}(\mathrm{CFGprr})$, and $\mathcal{L}(\mathrm{CFGfrr})$ the families of languages generated by the corresponding context-free grammars with rewriting restrictions.

3. THE MAIN RESULT

In this section we show that both CFGs with permitting rewriting restrictions and CFGs with forbidding rewriting restrictions are strictly more powerful than CFGs. We consider first the forbidding case.

THEOREM 1. $\mathcal{L}(\mathrm{CF}) \subset \mathcal{L}(\mathrm{CFGfrr})$.

PROOF. It can be easily seen that $\mathcal{L}(\mathrm{CF}) \subseteq \mathcal{L}(\mathrm{CFGfrr})$. Indeed, let $G = (V, \Sigma, R, S)$ be a CFG. We can construct an equivalent CFGfrr $G' = (V', \Sigma, R, S, f)$ as follows. Let $V' = V \cup \{*\}$ where $*$ is a new nonterminal symbol not occurring in V and, for every $A \in V$, let $f(A) = *$. Clearly $*$ is a nonterminal that does not occur in the sentential forms of the derivations of G'. Thus $*$ never blocks a derivation of G', which means that $L(G) \subseteq L(G')$. On the other hand, $L(G') \subseteq L(G)$ also holds, thus we have $L(G) = L(G')$.

Next we give a CFGfrr G_f such that $L(G_f) \notin \mathcal{L}(\mathrm{CF})$. The terminal alphabet is $\{a, b, c, d, \#, \$\}$ and G_f will generate, among others, all words in $\{a^i \# c^j \$ b^k d^l \mid i > k \geq 0, j > l \geq 0\}$ but no words in $\{a^i \# c^j \$ b^k d^l \mid k \geq i \geq 0, l \geq j \geq 0\}$. To achieve this, we do the following. The only rule that can introduce the symbol $\#$ is such that after its application the number of $b's$ cannot be increased after the position of $\#$ in the sentential form. We construct a similar rule with the symbol \$ on the right-hand side.

Let $G_f = (V, \Sigma, R, S, f)$ be a CFGfrr where

- $V = \{S, A, A_1, B, B_1, C, C_1, D, D_1\}$,
- $\Sigma = \{a, b, c, d, \#, \$\}$,
- $f(S) = A_1$, $f(A) = B_1$, $f(B) = A$, $f(A_1) = B$, $f(B_1) = A_1$, $f(C) = D_1$, $f(D) = C$, $f(C_1) = D$, $f(D_1) = C_1$.

Moreover, let R be the following set of rules. R is divided into three parts: $R = R_1 \cup R_2 \cup R_3$ where

- $R_1 = \{S \to ACBD\}$,
- $R_2 = \{A \to aA_1 \mid \varepsilon, \quad B \to bB_1 \mid \varepsilon, \quad A_1 \to A \mid \#A_1AB, \quad B_1 \to B\}$,
- $R_3 = \{C \to cC_1 \mid \varepsilon, \quad D \to dD_1 \mid \varepsilon, \quad C_1 \to C \mid \$C_1CD, \quad D_1 \to D\}$.

We observe the following things. First of all, in every derivation of G_f, the rule $S \to ACBD$ is applied exactly once, namely at the first derivation step. After that step the nonterminal S does not occur again in the derivations. Thus the forbidding nonterminal $f(S) = A_1$ never blocks a derivation step that involves S (this also means that we could have chosen any nonterminal in V as $f(S)$).

Secondly, the rules in R_2 and R_3 work independently from each other in the following sense. The nonterminals $f(A)$, $f(A_1)$, $f(B)$, $f(B_1)$ do not occur in the right-hand sides of the rules of R_3, and similarly, $f(C), f(C_1), f(D), f(D_1)$ do not occur in the rules of R_2. Thus applying a rule from R_2 cannot introduce a nonterminal that would block the application of rules in R_3 and vice versa.

Finally, we note that the structure of the rules in R_2 is the same as that of the rules in R_3.

By the above observations, to see what words can be derived from S, it is enough to examine those sub-derivations of G_f that use rules from R_2 only. Those sub-derivations that use rules from R_3 are similar. We will not describe the language $L(G_f)$ precisely as we are interested only in its intersection with the regular language $\{a^i \# c^j \$ b^k d^l \mid i, j, k, l \geq 0\}$.

Clearly the first derivation step in every derivation of G_f is $S \Rightarrow ACBD$. In the word $ACBD$ the B cannot be rewritten

because of the presence of A. Thus we can apply only $A \to aA_1$. Now A_1 cannot be rewritten because of B, so we apply $B \to bB_1$. Thus we have the sub-derivation

$$ACBD \Rightarrow^* aA_1CbB_1D.$$

By rules in R_2 we can rewrite only A_1. If we apply the rule $A_1 \to \#A_1AB$, then we get the sentential form $a\#A_1ABCbB_1D$. In this case no rule from R_2 can be applied (because of the common presence of nonterminals A, B, A_1, B_1). Therefore no word from Σ^* can be derived in this way. Thus we chose the rule $A_1 \to A$, then the rule $B_1 \to B$ and get the following sub-derivation:

$$aA_1CbB_1D \Rightarrow^* aACbBD.$$

Repeating the above applied rules, G_f can increase the number of $a's$ and $b's$ simultaneously.

Continuing our derivation, we apply the rule $A \to aA_1$ (note that $A \to \varepsilon$ could be applied also) and then the rule $B \to \varepsilon$. In this way B_1 did not appear in the sentential form, thus we can apply now the rule $A_1 \to \#A_1AB$ getting the following derivation:

$$S \Rightarrow^* aACbBD \Rightarrow^* aaA_1CbD \Rightarrow aa\#A_1ABCbD.$$

Now we have the situation when the number of $a's$ before the symbol $\#$ still can be increased, but the number of $b's$ in between the nonterminals C and D is fixed.

Next we continue by applying the rules $A \to \varepsilon$, $B \to \varepsilon$, $A_1 \to A$, $A \to \varepsilon$ that yields the following derivation:

$$S \Rightarrow^* aACbBD \Rightarrow^* aaA_1CbD \Rightarrow aa\#A_1ABCbD \Rightarrow^* aa\#CbD.$$

Finally, applying rules to $aa\#CbD$ from R_3 similarly, we can get, for example, the following derivation:

$$S \Rightarrow^* aa\#CbD \Rightarrow^* aa\#ccc\$bdd.$$

Next we show that $L(G_f) \notin \mathcal{L}(\mathrm{CF})$. Assume on the contrary that $L(G_f) \in \mathcal{L}(\mathrm{CF})$. Note that there are words in $L(G_f)$ which do not contain the symbol $\#$ or \$, and words which contain more than one occurrence of $\#$ or \$. To eliminate these words from $L(G_f)$ we define the regular language $L = \{a^i\#c^j\$b^kd^l \mid i,j,k,l \geq 0\}$. Clearly,

$$L' := L(G_f) \cap L = \{a^i\#c^j\$b^kd^l \mid i > k, j > l\}.$$

It is known that context-free languages are closed under the intersection with regular languages (cf. e.g. Theorem 7.27 of [4]), thus $L' \in \mathcal{L}(\mathrm{CF})$.

Next we define a homomorphism that erases the symbols $\#$ and \$ from the words of L'. Let $h : \Sigma^* \to \{a,b,c,d\}^*$ be the homomorphism such that $h(x) = x$ for every $x \in \{a,b,c,d\}$, and $h(\#) = h(\$) = \varepsilon$. It is easy to see that

$$L'' := h(L') = \{a^ic^jb^kd^l \mid i > k, j > l\}.$$

As context-free languages are also closed under homomorphisms (cf. e.g. Theorem 7.24 of [4]), we get that L'' is in $\mathcal{L}(\mathrm{CF})$. On the other hand, it is well known that L'' is not a context-free language. This is a contradiction that finishes the proof of the theorem. □

Next we show a similar result concerning the language class $\mathcal{L}(\mathrm{CFGprr})$.

THEOREM 2. $\mathcal{L}(\mathrm{CF}) \subset \mathcal{L}(\mathrm{CFGprr})$.

PROOF. Clearly $\mathcal{L}(\mathrm{CF}) \subseteq \mathcal{L}(\mathrm{CFGprr})$, thus it is enough to construct a CFGprr G_p such that $L(G_p) \notin \mathcal{L}(\mathrm{CF})$.

Let $G_p = (V, \Sigma, R, S, f)$ be a CFGprr where

- $V = \{S, A, A_1, B, B_1, C, C_1, D, D_1\}$,
- $\Sigma = \{a, b, c, d\}$,
- $f(S) = S$, $f(A) = A$, $f(B) = A_1$, $f(A_1) = B_1$, $f(B_1) = A$, $f(C) = C$, $f(D) = C_1$, $f(C_1) = D_1$, $f(D_1) = C$.

Again, R is divided into three parts: $R = R_1 \cup R_2 \cup R_3$ where

- $R_1 = \{S \to ACBD\}$,
- $R_2 = \{A \to aA_1 \mid \varepsilon,\ B \to bB_1,\ A_1 \to A,\ B_1 \to B \mid \varepsilon\}$,
- $R_3 = \{C \to cC_1 \mid \varepsilon,\ D \to dD_1,\ C_1 \to C,\ D_1 \to D \mid \varepsilon\}$.

First we note that here, similarly to the case of G_f, the application of the rules in R_2 is independent from the application of the rules in R_3. The idea behind the construction of G_p is the following. The rules with A as the left-hand side can be applied always when A is in the sentential form. However, if both A and B are in the sentential form, then applying $A \to \varepsilon$ stops the derivation as this way A_1 cannot be introduced and so B cannot be rewritten anymore. The same applies to the nonterminals C and D. Thus G_p produces a's and b's simultaneously, and eventually applies the rule $B_1 \to \varepsilon$. After this several a's still can be produced before the application of the rule $A \to \varepsilon$. The derivation goes in the same way concerning the nonterminals C and D.

Now, it can be seen that the language defined by G_p is the language

$$\{a^ic^jb^kd^l \mid i \geq k \geq 1, j \geq l \geq 1\}.$$

Clearly this language is not in $\mathcal{L}(\mathrm{CF})$. □

4. CONCLUSION

We have considered two simple types of context-free grammars with rewriting restrictions. These systems involve a very simple regulation above the application of the rules. Despite of this, they can still generate languages outside of the class $\mathcal{L}(\mathrm{CF})$. This also means that the sets of derivation trees of these systems are not recognizable by finite tree automata. It follows that even those powerful tree transducers

(macro and pebble tree transducers, for example) that induce tree transformations with recognizable domains cannot check whether the input tree is a valid derivation tree of a CFGfrr or a CFGprr.

Concerning the future work, we think that the languages in $\mathcal{L}$(CFGfrr) are closed under the intersection with regular languages. As they are also closed under homomorphisms, proving the above result would yield that even the language $\{a^i c^j b^k d^l \mid i > k, j > l\}$ is in $\mathcal{L}$(CFGfrr).

It is known that permitting random context grammars are equivalent to CFGs with permitting rewriting restrictions (see [1]). It seems to be a challenging task to prove or disprove the equivalence of forbidding random context grammars (where the permitting sets are all empty) and CFGs with forbidding rewriting restrictions.

Finally, as already mentioned, CFGs with permitting/forbidding rewriting restrictions are not Turing complete. It is also interesting whether we can narrow down the class of languages that they generate. We think we can show that every language in $\mathcal{L}$(CFGprr) is semilinear, but proving that languages in $\mathcal{L}$(CFGfrr) are also semilinear seems to be difficult. We tried to simulate CFGs with forbidding rewriting restrictions by several devices that can generate only semilinear sets, but we did not succeed mainly because of the following reason. CFGs with forbidding rewriting restrictions always test whether the number of occurrences of a nonterminal B in the current sentential form is zero or not before applying a rule $A \to \alpha$ with $f(A) = B$. It seems that these "zero tests" cannot be simulated easily by those devices that can generate only semilinear sets.

5. ACKNOWLEDGEMENTS

The author is grateful for the useful comments and suggestions of the referees, which led to the improvement of this paper.

6. REFERENCES

[1] J. Dassow, T. Masopust. On Restricted Context-Free Grammars. In Y. Gao, H. Lu, S. Seki, S. Yu (Eds.), *Proceedings of DLT 2010*, pages 434–435, LNCS 6224, 2010.

[2] J. Dassow and T. Masopust. On restricted context-free grammars. *Unpublished manuscript.*

[3] J. Dassow and Gh. Paun. *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, New York, 1990.

[4] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to automata theory, languages, and computation*, 2nd edition. Addison Wesley, 2000.

[5] T. Masopust. Simple restriction in context-free rewriting. *Journal of Computer and System Sciences* 76:837–846, 2010.

[6] M. Sipser. *Introduction to the Theory of Computation*, 2nd edition. Thomson, Boston, 2006.

Using Multigraphs in the Shallow Transfer Machine Translation

Jernej Vičič
University of Primorska
Glagoljaška 8, SI-6000 Koper
jernej.vicic@upr.si

ABSTRACT

The paper presents a new architecture for a shallow-transfer rule based machine translation system. The newly proposed architecture keeps track of all translation candidates generated by the ambiguities of the analysis phase. The architecture is based on multigraphs. The empirical evaluation shows that the new architecture produces better translation quality results with constant delay in time.

Keywords

Multigraph, Shallow transfer RBMT, Machine translation

1. INTRODUCTION

The paper presents a new architecture for a shallow-transfer rule based machine translation system. The newly proposed architecture does not cope with morphological and syntactical ambiguities but keeps track of all possible translation candidates, thus retaining all the information to the last phases of the translation process. The architecture is based on multigraphs. A discussion about time and space complexity of the proposed architecture, the algorithms and the data structures is presented. An experimental system as a proof of concept has been constructed on the basis of Apertium [2] and language data for the language pair Slovenian-Serbian. The rest of the paper is organised as follows: The domain description is presented in the Section 2, the motivation for the research is presented in Section 3. The methodology is presented in Section 4, space and time complexity of the presented data structures and algorithms is presented in Section 5, empirical evaluation and results are presented in Section 6 and conclusions in Section 7.

2. DOMAIN DESCRIPTION

2.1 Shallow Transfer Rule-Based MT

One of the methods, which guarantees relatively good results for the translation of closely related languages is the method of a rule-based shallow-transfer approach. It has a long tradition and it had been successfully used in a number of MT systems, the most notable of which is Apertium [2].

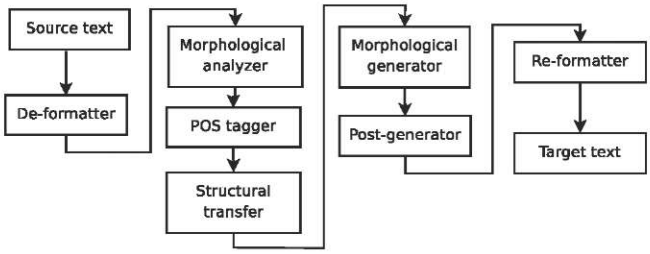


Figure 1: The modules of a typical shallow transfer translation system. The systems [2, 8, 11, 12] follow this design.

Shallow-transfer systems usually use a relatively linear and straightforward architecture where the analysis of a source language is usually limited to the morphemic level.

The Figure 1 shows the architecture of the most known translation systems for related languages Apertium [2] and Česílko [8].

The monolingual dictionaries are used in the morphological parsing of the source text by the morphological analyzer module and in the generation of the translation text in the target language by the morphological generator module. The Part Of Speech (POS) tagger module is used to disambiguate the ambiguous output of the morphological analyzer module. The bilingual dictionary is used for word-by-word translation, in our case the translation is based on lemmata. The shallow transfer rules are used to address local syntactic and morphological rules such as local word agreement and local word reordering. The module using the bilingual dictionary and the shallow transfer rules is the structural transfer module. The remaining modules deal with text formatting which is not domain of this paper. All methods and materials discussed in this paper were tested on a fully functional machine translation system based on *Guat* [13], a translation system for related languages based on *Apertium* [2].

The majority of the translation systems for related languages use the shallow parsing machine translation architecture as shown in [14]. Apertium is an open-source machine translation platform. The platform provides a language-independent machine translation engine, tools to manage the linguistic data necessary to build a machine translation system for a given language pair and linguistic data.

2.1.1 *Apertium*

Apertium is an open-source machine translation platform, initially aimed at related-language pairs but recently expanded to deal with more divergent language pairs (such as English-Catalan). The platform provides a language-independent machine translation engine, tools to manage the linguistic data necessary to build a machine translation system for a given language pair and linguistic data for a growing number of language pairs. All these properties make Apertium a perfect choice in a cost effective machine translation system development.

2.1.2 *GUAT*

All methods and materials discussed in this paper were tested on a fully functional machine translation system based on *GUAT* [15] and [13], a translation system for related languages based on *Apertium* [2]. The system *GUAT* was used as the sandbox for the implementation of proposed methods. Guat is automatically constructed so there is still room for improvement mainly through data correction tasks. The basic architecture of the system follows the architecture of apertium [2] and is presented in the Figure 1.

2.2 Multigraph

In mathematics, a multigraph or pseudograph is a graph which is permitted to have multiple parallel edges between nodes, that is, edges that have the same start and end nodes. Thus two vertices may be connected by more than one edge. Formally, a multigraph G is an ordered pair $G := (V, E)$ with:

- V a set of vertices or nodes,
- E a multiset of unordered pairs of vertices, called edges or lines.

Such multigraphs allow compact description of all available translation hypotheses.

2.3 Java JUNG

The Java Universal Network/Graph Framework, JUNG, is an open-source software library that provides a common and extendible graph/network analysis and visualization framework. It is written in Java. The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, multi-modal graphs, graphs with parallel edges, and hypergraphs. It provides a mechanism for annotating graphs, entities, and relations with metadata. This facilitates the creation of analytic tools for complex data sets that can examine the relations between entities as well as the metadata attached to each entity and relation. There are a number of already implemented algorithms from graph theory, data mining, and social network analysis already available in the library.
JUNG also provides a visualization framework that makes it easy to construct tools for the interactive exploration of network data.

3. MOTIVATION

The shallow-transfer RBMT architecture usually exploits a morphological disambiguator (tagger), which precedes any kind of more or less deterministic transfer phase. This is obviously a huge limitation, especially for lexical transfer, since in most language pairs there are many words whose translation depends upon the syntactic and/or semantic context. If the system contains some (shallow) syntactic parser and/or structural transfer, they also tend to produce ambiguous output relatively often.
The most important motivative reasons for this research are:

- The production of a new POS tagger, especially a good quality tagger, is not a simple task. One of the easiest methods is training of a stochastic tagger based on HMM algorithm [17]. Some parts of this task can be automatized using unsupervised learning methods or supervised learning methods like [1], but it still involves the selection of a new tag set, the production of a tagged training corpus, testing of the corpus and at the end the basic learning process.
- The quality level of the tagging process of today's state-of-the-art POS taggers for highly inflectional languages like [7] and [5] is relatively low, comparing to the quality of POS taggers for the analytical languages like the English language, and also comparing to the overall quality of the translation systems for related languages.
- According to the today's most used designs for translation systems for related languages, the shallow transfer translation systems, the disambiguation module follows the source language morphological analysis at the beginning of the translation process. This design is shown on figure 1. Such design is adopted by Apertium [2] and Česílko [8]. Errors produced at the early stages of the translation process usually cause bigger problems than errors introduced at latest phases as later phases of the translation rely on the output of the preceding phases.
- Multiple translation candidates allow selection of the best candidates in the final phase when all available data for the translation has been accumulated. The most common translation errors are fluency errors of the target language and not adequacy errors. These errors commonly do not interfere with the meaning of the translation but rather on the grammatical correctness of the translation. They are mostly caused by the errors in morphological analysis or morphological syntheses.

The omission of the tagger and introduction of a ranking scheme based on target language statistical model as suggested in [9] yields better translation results as suggested in the same paper. The introduction of multiple translation candidates generated from all possible morphological ambiguities as suggested in [9] leads to an exponential growth of the number of possible translation candidates. The paper [14] proposed a rule-based method for eliminating the impossible translation candidates thus lowering the number of possible translation candidates. A statistical ranking method was used to select an arbitrary number of best candidates. The rules were automatically constructed.
The method proposed in this paper does not use any kind of statistical ranking or any kind heuristics to avoid the exponential explosion of possible translation candidates, all the candidates are considered and the best candidate selected in the last phase, the ranking phase.

4. METHODOLOGY

4.1 Proposed architecture

The unified data structure would result in the rewrite of almost all modules of the original Apertium system. One of the most appealing features of the Apertium system is the transparency of the

translation process. All data is shared in text form through simple UNIX pipes resulting in an easy error discovery and easy debugging. The proposed data structure would have to be serialized in a human-readable form.

No change was made to the Apertium toolset for the means of the presented experiment. A new module has been added to the architecture, the Multigraph supervisor, which constructs the multigraph data structure and communicates with the Apertium modules through UNIX named pipes. The new architecture is presented on the Figure 2. The Multigraph supervisor module constructs the translation candidates by sending parts of the sentences, connecting edges in the data structure, to the appropriate module and saves the result in the same data structure gradually constructing all translation candidates. The data structure is further presented in the Section 4.1.1.

The POS tagger module that handles the morphological disambiguation has been omitted from the architecture as all the ambiguities are stored and dealt with in the later modules.

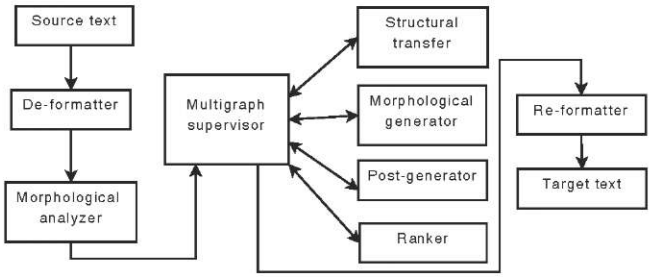


Figure 2: The proposed new architecture of a shallow transfer RBMT system using multigraphs. The Supervisor module uses the original modules in the translation process.

4.1.1 *The data structures*

Two data structures based on multigraphs were used in the Multigraph supervisor module, the construction process and the most distinct properties are presented in the Subsections following this section.

4.1.2 *Morphological analysis*

The morphological analysis produces ambiguous results, there are multiple Morpho-Syntactical descriptors - MSD [4] that can be attributed to one word form, an example of an ambiguously tagged sentence is presented in Figure 3.

The introduction of multiple translation candidates generated from all possible morphological ambiguities as suggested in [9] leads to an exponential growth of the number of possible translation candidates.

The output of the morphological analysis is a set of all possible morphological tags describing each word. Every word with more than only one tag can be observed as a set of possible ambiguities. In the case of highly inflectional languages like the pair presented in this paper the number of ambiguous possibilities increases. The set of all possible translation candidates is constructed as the vector product of all ambiguous sets. The number of possible translation candidates grows exponentially with the length of the sentence, the equation 1 shows the upper limit of the number of possible translation candidates.

```
Danes je lepo vreme.
Danes
Danes<adv>
je
biti<vbser><pres><p3><sg>
jesti<vblex><pres><p3><sg>
prpers<prn><subj><p3><f><sg><gen>
lepo
lep<adv>
lep<adj><f><sg><acc><pos>
lep<adj><f><sg><ins><pos>
lep<adj><nt><sg><acc><pos>
lep<adj><nt><sg><nom><pos>
vreme
vreme<n><nt><sg><acc>
vreme<n><nt><sg><nom>
```

Figure 3: The ambiguously tagged sentence *Danes je lepo vreme.*

$$|TC| = \prod_{i=0}^{|S_{max}|} x_{max} \tag{1}$$

where TC is the set of possible translation candidates for the longest sentence S_{max} and x_{max} is the biggest number of ambiguities for a word. Although the equation 2, which shows the average number of possible translation candidates, presents much lower numbers, the complexity of the problem still remains exponential.

$$|TC| = \prod_{i=0}^{|\bar{S}|} \bar{x} \tag{2}$$

Equations 4.1.2 and 4 show empirical values for an example source sentence and typical numbers collected from a corpus test-set.

$$\begin{gathered} |\bar{S}| = 40 \\ \bar{x} = 15 \\ |TC| = \textstyle\prod_{i=0}^{40} 15 = 110,573323209e+45 \end{gathered} \tag{3}$$

$$\begin{gathered} |S| = 15 \\ x = 3 \\ |TC| = \textstyle\prod_{i=0}^{15} 3 = 14,348,907 \end{gathered} \tag{4}$$

The data structure that can contain all the information produced by the morphological analysis in a compact form and also enable easy access to all translation candidates would be a multigraph, where nodes represent word boundaries, edges represent all possible ambiguous word forms. An example multigraph of the same example sentence from Figure 3 is shown on Figure 4.

4.1.3 *Structural transfer*

The structural transfer rules are usually made in two parts; the search pattern and action. We will concentrate on Apertium style rules although the abstraction would apply to most systems. A

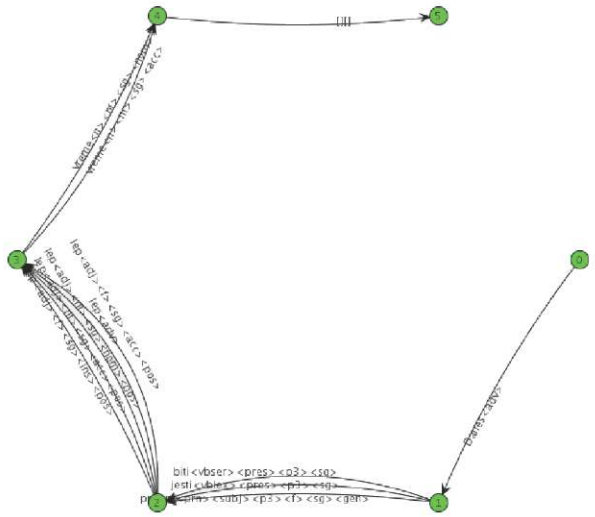


Figure 4: The multiple possibilities of the morphological analysis are stored in the edges of the multigraph. The example multigraph is constructed from the data on the Figure 3.

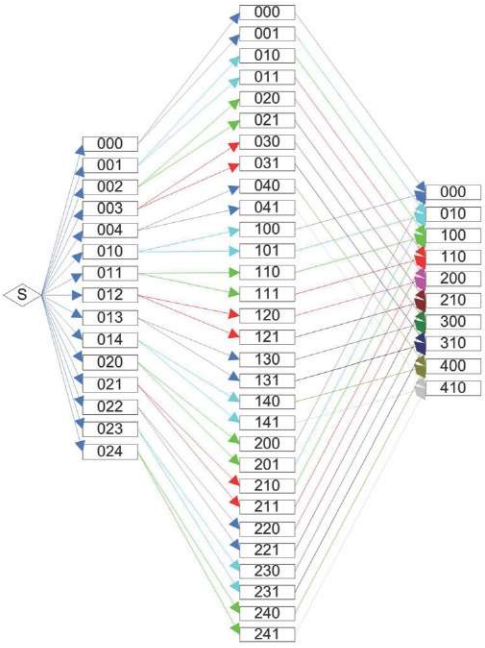


Figure 5: The data structure storing all the data from the Structural transfer module.

```
Danes je lepo vreme.
Danes
Danes<adv> biti<vbser><pres><p3><sg> lep<adv>
000
```

Figure 6: The trigram is represented by the string *"000"*

search through the Apertium systems [1] showed that the length of 3 suffices for more than 98% of rules. The linguistic explanation is that the rules act in a very limited context.
Although we can safely use the length of the longest context of 3 in the majority of cases, we will abstract the length to an arbitrary length l_R.
In order to be able to apply the rules in the Left to Right Longest Match (LRLM) which has been proven to be effective by [16], all possible candidates of the length l_R are constructed starting at the beginning of the multigraph, as the one presented on the Figure 4, and gradually moving to the last node of the multigraph. It can be easily proven that this algorithm constructs a LRLM coverage of all translation candidates. The candidates are sent to the structural transfer module and the result is stored in a new data structure shown in the Figure 5. The value for l_R has been set to 3 to simplify the visualization of the data structure.

The Figure 5 shows a representation of the complex data structure produced from the morphological output, presented in Figure 3, and stored in the multigraph presented in Figure 4 All the morphological descriptions have been numbered. The paths connecting the morphological descriptors have been constructed using these numbers. Let us observe the example on Figure 6 where the presented trigram is represented by the string *"000"*.

All strings finishing at a certain length are stored in the same column. The trigrams containing the morphological descriptors of three adjacent word forms are stored in nodes, the edges will be used to store the probabilities of the trigrams.

4.1.4 *Morphological generation*

[1]Apertium project at Sourceforge: http: //sourceforge.net/projects/apertium/

The lexical units that were stored as n-grams in the complex data structure are feed to the Morphological generator which generates the linearized text. The data is stored in the same data structure.

4.1.5 *Ranking*

The ranking process simulates the trigram language model. The process computes the probabilities for all trigrams and stores them in the multigraph data-structure. The probability of the observed trigram is stored to all the edges finishing in the observed node.
Any algorithm that produces the minimal path of the graph can be used to find the best candidate, in this experiment the Dijkstra algorithm [3] was used. The values on the edges of the graph have been altered to reflect the technical constraints of the Dijkstra algorithm (summation of the weights and only positive values). The values were logarithmically and absolute values were taken into account.

5. SPACE AND TIME COMPLEXITY

A few definitions that will alleviate the discussion of the complexity of the presented algorithms and data structures.
l_R - longest rule pattern length
l_S - longest sentence length
a - biggest number of ambiguities
The data structure presented in Section 4.1.2 is a multigraph with l_S number of nodes and $|V| = a$ number of edges between two

nodes. The total number of edges:

$$|E| = l_S * a \tag{5}$$

In numbers: setting the longest sentence to 30 words and the biggest number of ambiguities for a word to 26 (from the corpus Multext-east [6]). The total number of edges:

$$|E| = l_S * a = 30 * 14 = 780 \tag{6}$$

The total number of nodes:

$$|V| = 30 \tag{7}$$

The data structure presented in Section 4.1.3 is a multigraph with $l_S - l_R + 1$ number of sets of nodes, where each set of nodes can have up to $|V| = a^{l_R}$ nodes. Each node is connected with up to a edges (valence) to nodes in adjacent set of nodes. In numbers: the Equation 8 and 9 presents the total number of nodes and the Equation 9 presents the total number of edges using the same values as presented in the previous example. The total number of nodes for the multigraph data structure:

$$|V| = a^{l_R} = 28 * 14^3 = 76832 \tag{8}$$

The total number of edges for the multigraph data structure:

$$|E| = 76832 * a = 1075648 \tag{9}$$

The worst-case scenario cannot be reached as most of the word positions and POS variations are dependent.

6. EMPIRICAL EVALUATION AND AND RESULTS

Two main goals were evaluated in this experiment:

- the change in the quality of the final results, the translations.
- the time complexity

The newly proposed system was compared to two already available translation system for the same language pair:

- the original off-the shelf Apertium system with the Slovenian-Serbian translation data, described in [13]
- to the system presented in the [14]

The later system uses a method to restrict the number of translation candidates.

6.1 Translation quality comparison

The Word Recognition Rate - WRR metric, which is derived from the edit-distance [10], was used to evaluate the translation quality. The metric counts the number of deletions, insertions and substitutions that need to be performed among the observed sequences, i.e. count the number of edits needed to produce a correct target sentence from automatically translated sentence. This procedure shows how much work has to be done to produce a good translation. The metric roughly reflects the complexity of the post-editing task. The evaluation task in both comparisons comprised of translating the test sentences using all translation systems and manually correcting the output of the systems to a suitable translation. The definition of a suitable translation understood in this experiment is a translation that is syntactically correct and expresses the same meaning as the source sentence.

The comparison between the system presented in this paper and the [14] was done on the same test-set as the evaluation presented in [14]. The test-set was relatively small due to the constraints of the systems evaluated in the [14]. The test data for this part of the experiment comprised of the 57 sentences. The sentences were chosen by length (shorter sentences than 15 words). This limitation enabled a fair comparison of the translation quality of all the systems. The complexity of each sentence was arbitrary, there was no special selection of the sentences using this criteria although shorter sentences are usually simpler in structure. The results of this part of the evaluation are presented in Figure 7.

GUAT original - the reference system, based on Apertium architecture

GUAT all candidates - a system that kept all translation candidates to the last phase (best translation performance, exponential growth of possible translation candidates).

GUAT rules+ranker - the system with a method that restricted the number of possible translation candidates in the starting phases of the translation.

Multigraph - the system with the newly proposed architecture.

The comparison between the system presented in this paper and the original GUAT system, which is based on the original Apertium architecture, was done on a new test-set which comprised of 200 sentences randomly selected from [6] corpus. Both systems used the same translation data, the construction process of the translation data was described in [13] and the data is available at the Sourceforge[2], The results of this part of the comparison are presented in Figure 8. The system with the newly proposed architecture shows an improvement over the reference system.

6.2 Time complexity

The empirical evaluation of the time complexity was done simultaneously with the evaluation of the translation quality. The test-data is described in Section 6.1, which comprised of 200 sentences randomly selected from [6] corpus. The tests were performed on a personal computer[3]. Table 1 shows the time complexity comparison between the original GUAT system and the newly proposed system.

Table 1: Empirical evaluation of the time complexity. The newly proposed system is roughly 6 times slower than the original on the selected test sentences.

System:	GUAT	Multigraph
Nr. of sentences	200	200
Total time (seconds)	377.78	2239.56
Per translation	1.89	11.20
Ratio	1.00	5.93

Description of the Table 1:

[2]http://sourceforge.net/projects/apertium/

[3]Laptop computer with Intel Core 2 Duo 1.6GHz Processor and 2GB of memory

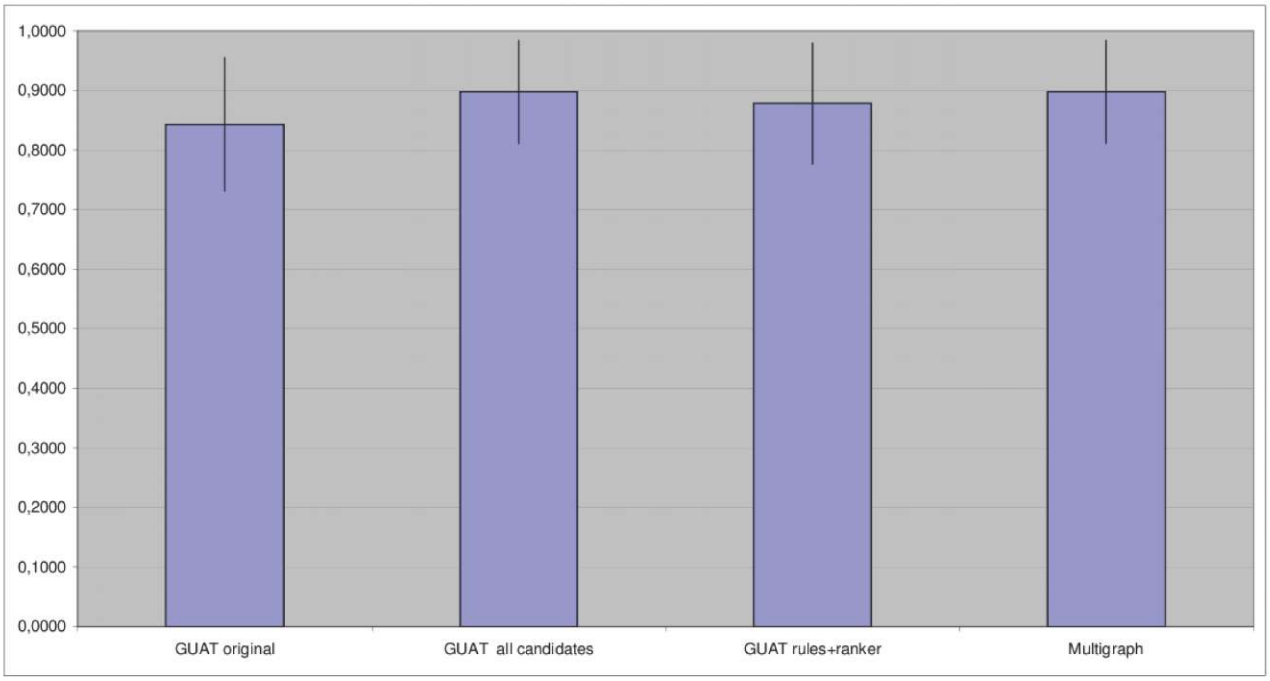


Figure 7: The translation quality evaluation, using WRR metric, of the systems presented in [14]. The newly proposed system outperforms all systems.

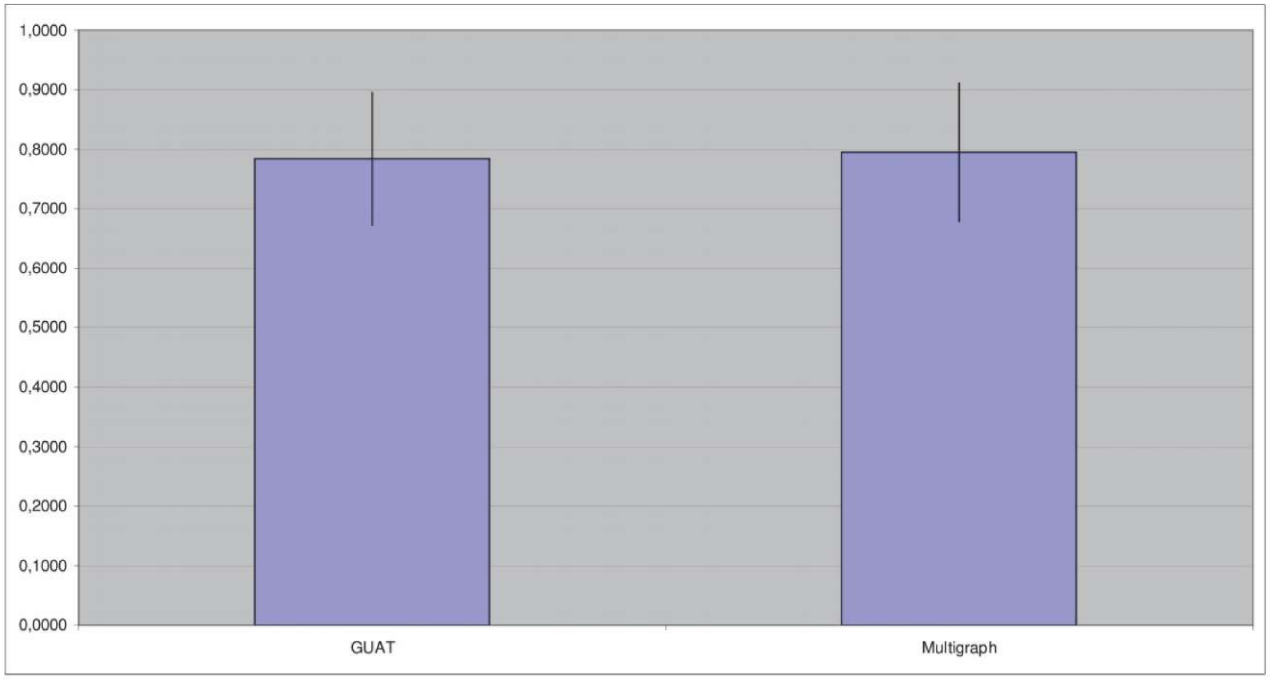


Figure 8: The translation quality evaluation, using WRR metric, of the GUAT system based on original Apertium architecture. The newly proposed system outperforms the original system.

- *GUAT* - The reference system, based on [13]
- *Multigraph* - the system with the newly proposed architecture.
- *Total time (seconds)* - the total time the system spent to translate all 200 test sentences.
- *Per translation* - the average time spent per translation.
- *Ratio* - The ratio between the time spent by the reference system - GUAT and the described system.

7. CONCLUSION

The presented architecture represents a viable solution to the problem of exponential growth of the number of possible translation candidates in a non-disambiguated shallow transfer translation system. The empirical evaluation showed an improvement in the translation quality compared to the original system and also compared to the values presented in [14], which was an attempt to limit the number of possible translation candidates. The empirical evaluation also showed that the new system performed as well as a system that included all possible translation candidates which shows that it always selected the best translation candidate.
The empirical evaluation of the time consumption showed that the new system performed roughly 6 times slower as the reference system and the constant factor was present in all test examples showing that the time complexity differed to a constant factor.
The proof-of-the-concept system has been implemented and it proved to be working as expected. A true implementation of the newly proposed architecture with the new module is already in the progress. It will be incorporated into the Apertium framework.

8. REFERENCES

[1] T. Brants. TnT—a statistical part-of-speech tagger. In *Proceedings of the 6th Applied NLP Conference*. Seattle, WA, 2000.

[2] A. M. Corbi-Bellot, M. L. Forcada, and S. Ortiz-Rojas. An open-source shallow-transfer machine translation engine for the Romance languages of Spain. In *Proceedings EAMT conference*, pages 79–86, May 2005.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[4] T. Erjavec. MULTEXT-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora. In *Proc. of the Fourth Intl. Conf. on Language Resources and Evaluation, LREC'04*, 2004.

[5] T. Erjavec. Multilingual tokenisation, tagging, and lemmatisation with totale. In *Proceedings of the 9th INTEX/NOOJ Conference*, 2006.

[6] T. Erjavec. Multext-east version 4: Multilingual morphosyntactic specifications, lexicons and corpora. In *LREC*, 2010.

[7] J. Hajič. Morphological tagging: data vs. dictionaries. In *Proceedings of the North American chapter of the Association for Computational Linguistics conference*, 2000.

[8] J. Hajič, P. Homola, and V. Kuboň. A simple multilingual machine translation system. In *Proceedings of the MT Summit IX*, New Orleans, 2003.

[9] P. Homola and V. Kuboň. Improving machine translation between closely related romance languages. In *Proceedings of EAMT*, pages 72 – 77, 2008.

[10] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk*, pages 845–848, 1965.

[11] K. P. Scannell. Machine translation for closely related language pairs. In *Proceedings of the Workshop Strategies for developing machine translation for minority languages*, 2006.

[12] F. M. Tyers, L. Wiechetek, and T. Trosterud. Developing prototypes for machine translation between two sámi languages. In *Proceedings of EAMT*, 2009.

[13] J. Vičič and P. Homola. Speeding up the implementation process of a shallow transfer machine translation system. In *Proceedings of the 14th EAMT Conference*, pages 261–268, Saint Raphael, France, 2010. European Association for Machine Translation.

[14] J. Vičič, P. Homola, and V. Kuboň. A method to restrict the blow-up of hypotheses of a non-disambiguated shallow machine translation system. In *RANLP*, pages 1–8, Borovec, Bulgaria, 2009.

[15] J. Vičič. Rapid development of data for shallow transfer rbmt translation systems for highly inflective languages. In *Language technologies : proceedings of the conference*, pages 98–103, 2008.

[16] J. Vičič and M. L. Forcada. Comparing greedy and optimal coverage strategies for shallow-transfer machine translation. In *Intelligent information systems XVI : proceedings of the International IIS '08 conference*, pages 307–316, 2008.

[17] L. R. Welch. Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, 53(4):1–14, 2003.

Model Checking of the Slotted CSMA/CA MAC Protocol of the IEEE 802.15.4 Standard

Zoltán L. Németh
Department of Computer Science
University of Szeged, Hungary
zlnemeth@inf.u-szeged.hu

ABSTRACT

We develop a verification model of the slotted CSMA/CA MAC protocol of the IEEE 802.15.4 standard in the Spin model checker. Then, we conduct experiments with varying parameters and models. Guarded by our experiences we argue that model checking can be an adequate tool for analysis of wireless sensor network protocols.

Keywords

Formal Verification, Model Checking, Wireless Sensor Networks, IEEE 802.15.4 Standard

1. INTRODUCTION

Wireless sensor networks, WSN for short, are rapidly developing new technologies with several military, industrial and civilian applications, e.g., they are applied in battlefield surveillance, habitat and environment monitoring, industrial and home automation [10]. WSN consist of small, autonomous devices called motes. Motes are restricted in several ways. They have limited energy resources (typically batteries), limited computational power (typically microcontrollers with small amount of memory) and restricted communication capabilities (typically short range radios of a few tens of meters). Contrary to these restrictions of individual motes, if they are organized into a (self organizing) network, as a whole they are capable to accomplish tasks that otherwise would be hard to achieve.

Since both the motes themselves and the tasks to be solved can be very different, designing a WSN is usually a challenging task without predefined solutions. As in traditional networking the design is usually divided into development of protocol layers. But the analysis and the verification (of correctness) of the protocol design are not yet well established [13].

The usual methods in practice are simulation, emulation, testing on real implementation and sometimes mathematical analysis [8], but all these methods have serious backwards and limitations [13]. Therefore, considering new or complementary ways in WSN protocol verification is an interesting research perspective.

The other subject of the paper is model checking [1,6]. This is a verification technique that has been successfully applied to prove correctness of hardware and software design in the last few decades [5]. Thus, using this technique for verification of WSN is naturally appealing. The present paper investigates the possibility of this. This idea is not new, there are some related papers, like [3,4,9,11,13]. But application of model checking, or more generally that of formal methods, in the field of WSN is far from being general. This is mainly due to some serious obstacles discussed below.

The first problem is the hardness of faithful modeling. It is obvious that model checking, like every model-based verification technique, is at most as good as the underlying verification model is. But in WSN we must confront with some very strange attributes that are extremely hard to model, like nondeterminism, inherent broadcast nature and unpredictable behavior of the radio communication, the lack of precise timing, continuous motion and other highly dynamic behaviors. Note that this is also an issue in simulations, where it is known that imprecise simplifications of the characteristics of WSN can lead to misleading results.

The second obstacle is the well known state space explosions, that means that the number of states rapidly exceeds computational limits due to the combinatorial blow up of the state space when one increases the parameters, e.g., the number of motes in the networks. There are several techniques to overcome this problem, but it seems to remain the main limitation of model checking in practice.

Therefore, we follow the approach presented in [2], and regard our investigations as experiments. We would like to argue that model checking can be an appropriate tool for design and analysis WSN protocols and for tuning their parameters.

Our study concerns some aspects of the medium access control, MAC for short, protocol of the IEEE 802.15.4 standard [7]. The standard defines the physical and the MAC layer of WSN. While the physical layer is generally accepted and applied, the MAC layer is more discussed and its analysis is an active research area [4,8].

In the sequel we briefly introduce the slotted CSMA/CA MAC protocol of the standard. Then, we develop a general model of it in the Spin model checker [12]. After that, we conduct experiments with varying parameters and models, present our results, and finally derive some conclusions regarding both the behavior of the protocol and our modeling experiences.

2. THE SLOTTED CSMA/CA ALGORITHM OF THE IEEE 802.15.4. STANDARD

The IEEE 802.15.4 standard [7] specifies the physical layer and the medium access control (MAC) sublayer for low-rate, low-power personal area networks. Here we only describe that part of it which is absolutely necessary to understand our study. Moreover, we apply some simplifications as well. The missing details can be found e.g., in [8].

We solely interested in the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) protocol of the standard. Motes use this algorithm to compete for medium access. This is necessary, since if two or more motes transmit at the same time, a collision occurs, which means that we lose all the messages, as the receiver cannot decode any of them. Moreover, unlike in wired networks, in wireless communication collisions cannot be detected, since it is not possible to listen while sending. Therefore, the protocol applies a collision avoidance method to reduce the number of collisions.

This protocol is used when the network operates in a so called beacon-enabled mode. This means that the coordinator of the network periodically sends special control messages, called beacons to maintain synchronization and to control the network. The synchronization implies that we can align all transmissions into specific time segments called slots. Hence, this algorithm is also called the slotted CSMA/CA protocol distinguishing it from the unslotted operation mode used in other parts of the standard.

For our investigations we use a fixed time unit which equals to the constant `a unit backoff period` defined in the standard. We will use this unit time length to measure every time period, and assume that all message lengths and waiting times are integer multiple of it.

The main goal of the protocol is to reduce the number of collisions. For this every mote performs at least two Clear Channel Assessment (CCA) operations before transmitting. If the channel reported busy, a random waiting period, called *backoff* is applied.

The (somewhat simplified) flowchart of the algorithm can be seen in Fig 1. The main idea is the use of binary exponential backoffs, i.e., in the case of a busy channel the algorithm doubles the potential length of the next backoff.

Note that the constants `MIN`, `MAX` and `RETRIES` are referred in the standard as `MacMinBE`, `aMaxBE` and `macMaxCSMABackoffs`, in turn. The algorithm has three variables: NB denotes the number of backoffs, CW is the length of the contention window, and BE is the backoff exponent. If a mote has a message to send, it first initializes the values of the variables, then it waits for a random integer number of between 0 and $2^{\mathrm{BE}} - 1$ time slots. Then, it performs CCA in the next time slot. If the channel is busy, then the mote increases the value of both NB and BE (but BE cannot exceed the value of `MAX`). Now, if the NB is less than or equal to the constant `RETRIES`, the mote performs a new random backoff and checks the channel again. Otherwise, the algorithm reports failure. If the channel is free, the mote only sends the message after two successful CCA-s, since the length of the contention window is 2. The reason for using a contention window is that acknowledgement of successful transmissions may be required. In that case there is always some pause between the end of the message and the following acknowledgment. Thus, a single CCA signaling clear channel cannot ensure that both the message and its acknowledgement are ended.

3. MODELING

In our experiments we consider `N` motes in a star topology. One mote, say M_0, is the base station, the coordinator, that controls the network and collects data from it. While the other motes, say M_i for $1 \leq i \leq \mathtt{N}-1$, sense data and send them directly to the base station. For $i > 0$, at each mote M_i the transmissions occur regularly with a given period of $\mathtt{P}_i$. This means that at the beginning of each $\mathtt{P}_i$ slots a new package is generated that M_i intends to send. After transmitting (or dropping) the package, if time remains until the generation of the next package, M_i waits in an idle state or may go to a sleeping state to save energy. Of course, if no time remains, the mote immediately starts to send its next package.

An essential component of every modeling is finding the right abstraction. Aiming at this we show and briefly discuss our model of the protocol used in the verification. Unfortunately, limited space prevents us from covering all modeling decisions in detail, but the main ideas can be seen in the following outline:

1: **for all** i **from** 1 **to** $\mathtt{N}-1$ **do**
2: $NB[i] \leftarrow 0$
3: $T[i] \leftarrow \mathrm{random}(0..2^{\mathtt{MIN}}-1)$
4: **end for**
5: **while not all** $T[i] = \mathtt{TIMELIMIT}$ **do**
6: $m \leftarrow \min_{1 \leq i \leq N} T[i]$
7: $COL \leftarrow \begin{cases} \text{true} & \text{if } \exists\, i \neq j : T[i] = T[j] = m; \\ \text{false} & \text{otherwise.} \end{cases}$
8: $start \leftarrow m + \mathtt{CW}$
9: $end \leftarrow start + \mathtt{LENGTH}$
10: **for** i **from** 1 **to** $\mathtt{N}-1$ **do**
11: **if** $T[i] = m$ **then**
12: $T[i] \leftarrow end$
13: $NB[i] \leftarrow 0$
14: **if not** COL **then** SUCCESSFUL SEND
15: $T[i] \leftarrow$ TIME FOR THE NEXT CCA I.
16: **else** A COLLISION OCCURS
17: $T[i] \leftarrow$ TIME FOR THE NEXT CCA II.
18: **end if**
19: **else**
20: **while** $T[i] < end$ **do**
21: **if** $T[i] < start$ **then**
22: $T[i] \leftarrow start + 1$
▷ More than one CCA needed.
23: **else** $T[i] \leftarrow T[i] + 1$
▷ The channel is busy at the first CCA.

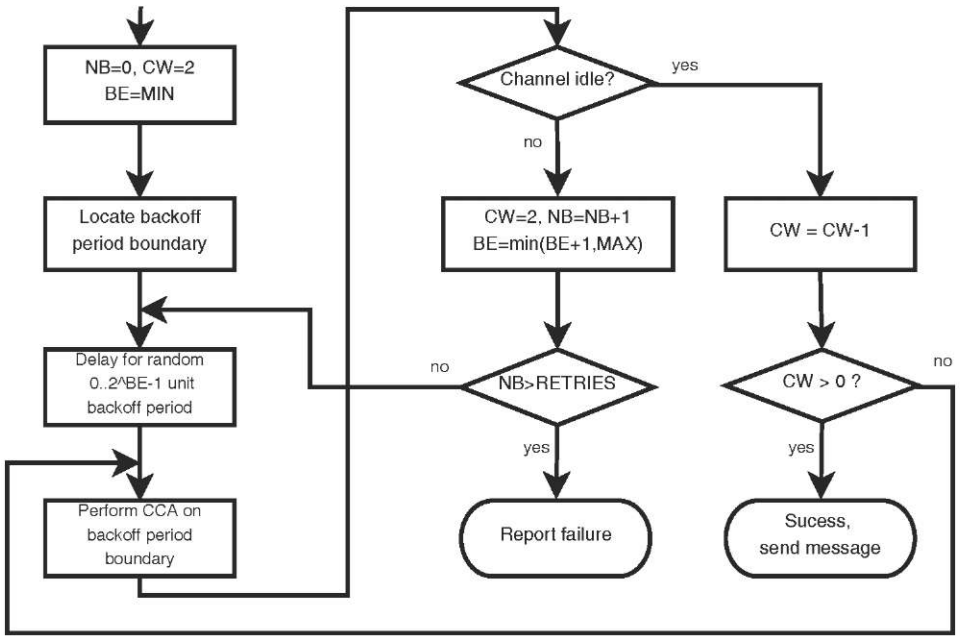


Figure 1: The CSMA/CA algorithm of the IEEE 802.15.4 standard

```
24:                end if
25:                NB[i] ← NB[i] + 1
26:                RETRY OR DROP THE PACKAGE AND
27:                SET T[i] and NB[i] ACCORDINGLY.
28:             end while
29:          end if
30:       end for
31: end while
```

The main idea is that we only store the time of the next CCA for every mote. For this we use the variable $T[i]$ for M_i. In the initialization phase (lines 1-4) for each mote we set a random backoff value before the first CCA. Then, until all motes reach the time given by `TIMELIMIT`, the following cycle is repeated (lines 5-31).

First, we determine the minimum of the $T[i]$-s and store this value in m (line 6). Clearly, if there is a single mote with $T[i] = m$, then, being the first, M_i will succeed in sending its message. Otherwise, i.e., when there are more the one motes with $T[i] = m$, a collision will occur (line 7). In any case the channel will be busy between times $start = m +$ `CW` and $end = start +$ `LENGTH` (lines 8-9). In the next cycle (lines 10-30) we set the time of the next CCA for each mote M_i as follows.

If M_i transmits, we first increase $T[i]$ to end, then we compute its time for the next CCA (line 11-18). This computation may depend on whether the network uses acknowledgements or not.

If $T[i]$ is greater than m, then M_i will not transmit. Instead, it will find the channel busy, since it starts its CCA-s later then m. If $T[i] < start$, this will happen at time $start$ resulting in more than one CCA. Otherwise, when $T[i] \geq start$, only a single CCA will occur. We increase the value of $T[i]$ accordingly (lines 21-24). As the last CCA reports the channel busy, we increase the backoff counter $NB[i]$ (line 25). If $NB[i]$ does not exceed the constant `RETRIES`, we add a possibly longer random backoff to $T[i]$, otherwise we retry to send or drop the package depending on what variation of the model we use (line 26-27). It is important to note that in the case of not sending motes we may need to increase $T[i]$ several times, until it reaches end, since more the one backoff can occur during a transmission of a (possibly long) package (lines 20-28). But, as soon as all motes reach the value of end, a new transmission(s) will happen, performed by the mote(s) having a minimal actual value of $T[i]$. As it was mentioned, we repeat the cycle (lines 5-31) until the end of the predefined value of `TIMELIMIT`.

In reality the verification model is slightly more complicated. E.g., approaching the `TIMELIMIT` we also need to check whether there is enough time to send a message. If not, the mote does not try to send, but defers the transmission instead. This behavior conforms to the standard. The modeling of the missing details as well as that of the variations of the model is more or less straightforward.

For the actual verification we used the open-source model checker Spin [6, 12], since its high level modeling language ensures a flexible framework to implement our model. Moreover, it contains several built in optimization techniques to support efficient verification.

One of the main question of this study is how to deal with the nondeterministic nature of the protocol. First of all, the best and the worst case behaviors of the protocol are uninteresting, since they can be determined easily, and consist of rather particular choice of the random backoff periods, as we shall see.

On the other hand the main benefit of model checking against testing, simulation or mathematical analysis can be the fact that in model checking we have control over the deterministic choices. Next we show an example for this.

In the simplest case we assume that the goal of the protocol is to send at least $X \geq 0$ messages successfully in a given time frame. It is obvious, that in this situation the worst case behavior happens, when all messages collide, hence the network is unable to achieve a single success. On the other hand, the best case behavior consists of only successful transmissions one right after the other without any collision. Thus, in the best case there are no collisions, while in the worst case there are many. Therefore, the following generalizations make sense (of course, for all natural numbers X and Y):

- Generalized worst case: W(X,Y): Do at most Y collisions guarantee at least X successes?
- Generalized best case: B(X,Y): Is it possible to achieve X successes with at least Y collisions?

For the purpose of model checking these properties can be easily formalized as formulas of linear temporal logic:

$$W(X,Y) = \Box(\text{COLL} \leq Y) \rightarrow \Diamond(\text{SUCC} \geq X), \quad (1)$$
$$B(X,Y) = \Diamond(\text{SUCC} \geq X \wedge \text{COLL} \geq Y). \quad (2)$$

In the formulas above, of course, SUCC denotes the number of successfully sent packages, and COLL stands for the number of collisions. Moreover, $\Box$ is the *always* and $\Diamond$ is the *eventually* LTL operator, see e.g. [1].

We are interested in the maximal value of Y for any given X defined by the $W(X)$ and $B(X)$ functions:

$$W(X) = \max\{Y \mid W(X,Y) \text{ is true for all possible runs.}\},$$
$$B(X) = \max\{Y \mid B(X,Y) \text{ is true for at least one run.}\}$$

Thus, $W(X)$ (resp. $B(X)$) gives the maximal number of collisions that guarantees (resp. allows) at least X successful transmissions. We will call these functions the *worst case* (resp. *best case*) *behavior function* of the protocol. These values can be easily determined by the model checker using repeated search.

For $W(X)$ we need to find the maximal value of Y for which property (1) holds. Technically, instead of starting from 0 and gradually increasing the value of Y, it is faster to start form an upper bound for Y and to decrease it, until the property is satisfied, since finding a counterexample is generally an easier computational task than validating a formula, which always requires the exploration of the whole state space. If (1) does not hold even for $Y = 0$, then $W(X)$ is undefined, since then X success cannot be guaranteed even without any collision.

For the computation of $B(X)$ we start from 0 and compute the maximal value of Y for which the negation of property (2) does not hold. The reason for using the negation is that the generalized best case requires the existence of just a single run, but the model checker always verifies all possible execution sequences.

4. THE EXPERIMENTS

During the course of modeling one of our aims was generality. Therefore, we introduced the following values as parameters: `N`, the number of motes; $\texttt{P}_1$, $\texttt{P}_2$, ... the periods of message generation; `LENGTH`, the length of the messages measured in slots; finally $\texttt{CW}_{init}$ (the initial value of CW), `RETRIES`, `MIN`, and `MAX` the parameters used in the standard.

Keeping the original protocol intact we considered two questions that affect significantly the behavior of the protocol.

1. Does the network employ an acknowledgement mechanism (i.e., does the sender make sure of successful arrival of a packet after transmission)? If this is applied, the motes are aware of the collisions, and they retry to send the collided packages.
2. Is there packet drop (i.e., after 1 + `RETRIES` numbers of backoffs and unsuccessful CCA operations does the sender drop the packet disregarding it permanently, or doses it retry to transmit the packet performing all the necessary steps of the protocol again from the beginning)?

According to the answers to these questions we defined four models, namely:

- **M** is without acknowledgement and packet drop,
- $\mathbf{M_{AD}}$ is with both acknowledgement and packet drop,
- $\mathbf{M_A}$ is with acknowledgement, but without packet drop, and
- $\mathbf{M_D}$ is without acknowledgement, but with packet drop.

In our first simple experiment we investigated the contention resolution, i.e., a situation in which all motes have a message to transmit at the same time. In this experiment the fixed parameters were $\texttt{N} = 4$, $\texttt{TIMELIMIT} = 54$, $\texttt{P}_1 = \texttt{P}_2 = \texttt{P}_3 = 54$, $\texttt{CW}_{init} = 2$, $\texttt{MIN} = 2$, and $\texttt{MAX} = 5$. We varied the other parameters as follows: $\texttt{LENGTH} = 2, 4, 6, \ldots, 16$ and $\texttt{RETRIES} = 0, 1, \ldots, 5$. We investigated both the generalized best case and worst case behaviors.

We computed the worst case behavior function $W(X)$ and the best case behavior function $B(X)$ for $X = 0, \ldots, 3$ and for all possible values of the parameters `LENGTH` and `RETRIES`. Some of the numerical results are presented in Table 1 and Table 2 for `RETRIES`=1.

For example, the values 3 and 7 in the intersection of row 'X=2' and column 'L = 4' in Table 1 refer to the fact that if `RETRIES`=1 and `LENGTH`=4, then during a `TIMELIMIT`=54 slots time period if the number of collisions is at most 3, then at least 'X=2' successful transitions are achieved (even in the worst case), but 'X=2' successful transitions can also happen in the presence of at most 7 collisions (in the best case).

$X \backslash L$	2	4	6	8	10	12	14	16
0	− 13	− 9	− 6	− 5	− 4	− 3	− 3	− 3
1	6 12	5 8	3 5	3 4	2 3	2 2	1 2	1 2
2	5 11	3 7	2 4	1 3	1 2	1 1	0 1	0 1
3	4 10	2 6	1 3	0 2	0 1	− 0	− 0	− 0

Table 1: The generalized worst case (on the left) and best case (on the right) behavior function of model $\mathbf{M_A}$ in the case of RETRIES = 1.

$X \backslash L$	2	4	6	8	10	12	14	16
0	− 13	− 9	− 6	− 5	− 4	− 3	− 3	− 3
1	6 12	5 8	3 5	3 4	2 3	2 2	1 2	1 2
2	− 11	− 7	− 4	− 3	− −	− −	− −	− −
3	− 10	− 6	− 3	− 2	− −	− −	− −	− −

Table 2: The generalized worst case (on the left) and best case (on the right) behavior function of model $\mathbf{M_{AD}}$ in the case of RETRIES = 1.

The other two models and the choice of the RETRIES parameter give similar numerical results, but instead of presenting them, we give a text description of our observations.

First, the results clearly show the differences in the behaviors of the four models.

The generalized best case behavior function of model $\mathbf{M}$ is independent from both the value of the RETRIES, and the LENGTH parameters. With one collision only one success, without collisions three successes can be achieved. The worst case behavior shows that, while collision freeness always ensures one success, two or three successes can only be guaranteed for small values of the RETRIES parameters especially for long messages. More precisely, two successes are not possible, if LENGTH ≥ 14 and RETRIES ≥ 3. Moreover, three successes are not possible, if $2 \leq$ LENGTH ≤ 4 and RETRIES ≥ 3, or if $6 \leq$ LENGTH ≤ 10 and RETRIES ≥ 2, or if LENGTH $= 12$ and RETRIES ≥ 1, or if $L \geq 14$. Thus, a large value of the LENGTH and the RETRIES parameter even without collisions can keep the model $\mathbf{M}$ from reaching a single successful transmission.

Model $\mathbf{M_A}$ applies acknowledgements, hence its behavior is far more favorable. Its best case behavior is independent from the value of RETRIES, on the contrary the worst case behavior is dependent on it. The values of the worst case ($W(X)$) and best case ($B(X)$) behavior functions are presented in Table 1 for different values of X (the number of successes) and LENGTH. In the table the worst case values refer to the case of RETRIES $= 1$. Again, increasing RETRIES and LENGTH parameters causes the decrease of $B(X)$.

Model $\mathbf{M_{AD}}$ shows similar behavior, with the notable exception of that the presence of packet drops makes also the best case dependent on the RETRIES parameter. But, now the changes are caused by small values of RETRIES. E.g., the model is unable to achieve more then one success, if RETRIES $= 0$; or RETRIES $= 1$ and LENGTH ≥ 10. Moreover, more than 2 successes are not possible, if RETRIES ≤ 2 and LENGTH ≥ 12. Similar differences occur in the worst case behavior.

Finally, its not surprising that the behavior of $\mathbf{M_D}$ differs from that of $\mathbf{M}$ similarly as the behavior of $\mathbf{M_{AD}}$ differs from $\mathbf{M_A}$.

We also performed experiments with sending multiple messages both under balanced and unbalanced load conditions. We obtained similar numerical data as in the previous tables.

In all, we performed more than 5000 model checking runs lasting about 9 hours. Unfortunately, we were severely restricted by memory limitations since the state space explosion problem occurred even with relatively small values of the parameters. For the verification we used an average PC with 3GB RAM.

While finding a counterexample was generally easy and fast, proving that no violating counterexample for a given formula exists was sometimes hard. 69 of the total runs were terminated by an "out of memory" error without fully completing the search for counterexamples.

But the main problem is not the actual amount of memory, which surely can be somewhat extended, but the exponential growth of the state space when one increases the number of motes (N). Our most complicated models (after several simplification) still have approximately 30 million states, and it seems impossible to exhaustively check models with more the 100 million states. But, changing the value from $N = 4$ to $N = 5$ is definitely beyond this threshold. The approximation suggests that, if LENGTH $= 16$ and RETRIES $= 5$, then the state space contains more than 1 billion states, that would require more than 50GB RAM to store. In our opinion, this explosion is due to the highly deterministic nature of the protocol, that seems very hard to overcome.

5. CONCLUSIONS AND FUTURE WORK

In this paper our main goal was to point out the potential of application of model checking for the analysis of WSN protocols and for tuning their parameters. We developed a verification model of the unslotted CSMA/CA MAC protocol of the IEEE 802.15.4 standard in the Spin model checker and studied the best and worst case behaviour functions of four basic models of the protocol with various parameter settings.

The modeling gives us a general framework for the analysis that can and should be extended in several directions. We plan to consider deadlines for message transmissions, breaking down collisions and successes into individual motes, and investigation of other performance metrics like latency and energy consumptions. It would be very interesting if one could also study problematic situations, like congestions, and their causes in this framework.

Also, our modeling attempts were far from being exhaustive. Further investigation is needed to explore the possibilities and limitations of this kind of experiments. Finally, validation of our findings with an other method of analysis, like

simulation or testing on real implementations is necessary. Nevertheless, we believe that model checking experiments even with such restricted instances can help us to better understand and explore the nonobvious behavior of WSN protocols.

Acknowledgment

This research was partially supported by the TÁMOP-4.2.2 /08/1/2008-0008 program of the Hungarian National Development Agency. The authors would like to thank Balázs Lévai for writing script files for the verification.

6. REFERENCES

[1] C. Baier, J.-P. Katoen, *Principles of Model Checking*, MIT Press, 2008.

[2] E. Brinksma, Verification is experimentation!, *Int. J. on Software Tools for Technology Transfer*, 3(2001), 107–111.

[3] A. Fehnker and L. van Hoesel and A. Mader, Modelling and Verification of the LMAC Protocol for Wireless Sensor Networks, in: *Integrated Formal Methods, IFM 2007*, LNCS, Vol. 4591, 253-272, Springer, 2007.

[4] M. Fruth, Probabilistic Model Checking of Contention Resolution in the IEEE 802.15.4 Low-Rate Wireless Personal Area Network Protocol, in proc: *ISoLA 2006*, 290-297.

[5] O. Grumberg, H. Veith (Eds.), *25 Years of Model Checking: History, Achievements, Perspectives*, Springer-Verlag, Berlin, Heidelberg, 2008

[6] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison- Wesley, 2003.

[7] IEEE 802.15 TG4, *802.15.4 Standard Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks*, 2006.

[8] J. Mišić, V. B. Mišić, *Wireless Personal Area Networks: Performance, Interconnection, and Security with IEEE 802.15. 4*, Wiley Publ., 2008.

[9] P. Ölveczky, S. Thorvaldsen, Formal modeling, performance estimation, and model checking of wireless sensor network algorithms in Real-Time Maude, *Theor. Comput. Sci.*, 410(2-3), 254-280, 2009.

[10] V. Potdar, A. Sharif, E. Chang, Wireless Sensor Networks: A Survey, *AINA Workshops 2009*, 636-641.

[11] O. Sharma, J. Lewis, A. Miller, A. Dearle, D. Balasubramaniam, R. Morrison and J. Sventek, Towards Verifying Correctness of Wireless Sensor Network Applications Using Insense and Spin, in proc.: *Model Checking Software, SPIN Workshop 2009*, LNCS Vol. 5578, 223-240.

[12] The SPIN Site, http://spinroot.com

[13] A. C. Viana, S. Maag, F. Zaidi, One step forward: Linking Wireless Self-Organising Networks Validation Techniques with Formal Testing Approaches, *ACM Computing Surveys*, 43(2), 2011, Article 7.

Experiment-based definitions for electronic exam systems

[Extended Abstract]

Andrea Huszti
Faculty of Informatics
University of Debrecen , Hungary
P.O. Box 12. H-4010 Debrecen
Hungarian Academy of Sciences and University of Debrecen , Hungary
huszti.andrea@inf.unideb.hu

ABSTRACT

We investigate security definitions for e-exam systems. Our aim concerning security properties of e-exam schemes is besides achieving all requirements that the traditional paper-based exam system provides to accomplish anonymity of examinees and exam correctors. We construct formal experiment-based definitions for *authenticity*, *correctness*, *secrecy* and *anonymity*.

Keywords

electronic exam systems, anonymity, experiment-based definitions

1. INTRODUCTION

E-exam management is one of the most important building blocks of an e-learning environment, it raises more security issues then other parts of an e-learning software. There are several papers [3, 8, 10] that detail design and non-formal security issues of the proposed e-exam schemes, and few commercial solutions [4, 7, 11] that do not describe security measures.

We investigate security definitions for anonymous e-exam systems, where exam questions might be multiple choice tests and write in questions, as well. Hence besides examinees, exam correctors are also participants. Providing these definitions of the basic requirements, we give a formal security framework for anonymous e-exam schemes, that can be applied for proving security issues of any proposed scheme. Our aim concerning security properties of e-exam schemes is besides achieving all requirements that the traditional paper-based exam system provides to accomplish anonymity of examinees and exam correctors. We construct formal definitions for *secrecy*, providing confidentiality of exam answers, *correctness*, ensuring that data transmitted is not modified, *anonymity*, guaranteeing examinees and exam correctors stay anonymous during the process and *authenticity*, stating that only eligible examinees and exam correctors are allowed to participate.

We have chosen experiment-based technique, or sometimes called game-based technique for constructing our definitions. Applying these formal definitions security proofs of complex cryptographic schemes become not that complicated, since these definitions give a good direction that one has to follow during the proof. We do not have to deal with too much formalism either, like in logic based constructions ([1], [2]), still we achieve reasonable level of mathematical correctness. For proving security requirements of a complex cryptographic scheme we will use reduction method, meaning if underlying cryptographic primitives are secure against certain attacks, then the proposed scheme possesses a certain security requirement. According to the reduction technique we need to prove that if a scheme does not feature a security requirement, then we can construct a machinery that breaks the underlying assumption. Security is defined as a game between an attacker and the system, that are probabilistic processes communicating to each other. Security means that the probability that a certain event occurs is negligible.

2. EXAM SCHEME

An exam scheme consists of three main stages: *registration, examination and grading stages*. During registration stage examinees (EX) and exam correctors (EC) prove their identity to Registry ($\mathcal{R}$). For eligible participants pseudonyms are generated that are authorized by $\mathcal{R}$. In the examination stage examinees receive exam questions and generates answers with time stamps that are sent to Examination Board (EB) with their pseudonyms. After successful submission a receipt is constructed for each examinee. After successful submission of answers EB sends exam answers to exam correctors, who give a mark. During grading stage real identity of examinees are revealed in order to insert the marks into the exam administration database.

An exam scheme formally consists of the following algorithms that output 0 if an error occurs:

$register(i, SK_{\mathcal{R}}) \longrightarrow \{pseudonym_i, 0\}$, where i is an identity number of an examinee or exam corrector, $SK_{\mathcal{R}}$ is the secret key of $\mathcal{R}$.

$takeexam(questions, pseudonym_i, SK_{TS}) \longrightarrow \{receipt_i, 0\}$, where SK_{TS} is the secret key of Time Stamp Service

Provider.

$correct(exam_i, SK_{TS}) \longrightarrow \{(mark_i, checksum), 0\}$, where *checksum* is verification information of grading

$getidentity(pseudonym_i) \longrightarrow \{i, 0\}$

We define an exam scheme as
ExS=$\{register, takeexam, correct, getidentity\}$.

3. EXPERIMENT-BASED DEFINITIONS

The basic requirements an electronic exam system should possess are as follows: *secrecy, correctness, anonymity* and *authenticity*. We consider passive and active adversaries. Passive adversaries listen to channels and all information on $\mathcal{BB}$. We denote all this information by I. Active adversaries collude with non-reliable participants, $\mathcal{A}$("*control participant*") describes the act of an active adversary. $\bar{I}$ denotes data generated by $\mathcal{A}$ controlling some other participant. At the end of our experiments guesses are made with knowledge of $\hat{I} = I \bigcup \bar{I}$. We give maximal power to the adversary in our definitions. In case of a concrete e-exam protocol assumptions on participant's reliability should be fixed, that might cancel an attack. Let us describe these requirements with details.

Our experiment-based definitions we use the following notation:

$\mathcal{BB}$: Bulletin Board, a physical apparatus that is publicly readable and only authorized participants are allowed to write on it
I: all available data (e.g. public channels, $\mathcal{BB}$) sent by honest participants
$\bar{I}$: all available data (e.g. public channels, $\mathcal{BB}$) sent by $\mathcal{A}$
$\hat{I}$: all data available
n/m: number of eligible examinees/exam correctors identified
$\bar{n}/\bar{m}$: number of ineligible examinees/exam correctors
$\hat{n}/\hat{m}$: number of examinees received a mark/exam correctors gave a mark

Exam answers are kept secret. During examination stage the generated answers are not revealed for an attacker. This property protects against cheating.

In our experiment adversary $\mathcal{A}$ has access to *register*(), *takeexam*(), *correct*() oracles, I contains all the output information of oracle queries. An oracle machine can be visualized as a Turing machine with a black box, that is able to calculate problems from any complexity classes in a single operation. After successful registration of examinee j, the adversary chooses $exam_{j_0}$ and $exam_{j_1}$, runs functions *takeexam*(.) and *correct*(.) with $exam_{j_b}$, where $b \in \{0, 1\}$ is chosen randomly by the system. During running *takeexam*(.) and *correct*(.) the adversary might collude with EB and get more information. With knowledge of all available information from communication channels and non-reliable participants $\mathcal{A}$ give guesses about b. If $\mathcal{A}$ gives good guesses with non-negligible probability then the scheme does not possess secrecy.

Experiment 1. **Secrecy of exam answers**
$\mathbf{Exp}^{secr-b}_{ExS,\mathcal{A}}(j)$,
$\{exam_i \leftarrow \mathcal{A}(\text{"}choose\ exams\text{"})\}_{i=1}^{q}$
$\{\bar{I} \leftarrow \mathcal{A}^{register(i,.),takeexam(pseudonym_i,.),correct(exam_i,.)}\}_{i=1}^{q}$
if $register(j,.) \neq 0$ **then**
$I \leftarrow register(j,.)$
$(exam_{j_0}, exam_{j_1}) \notin \{exam_1, \ldots, exam_q\} \leftarrow \mathcal{A}(\text{"}choose\ exams\text{"})$
if $takeexam(pseudonym_j,.) \neq 0$ **then**
$I \leftarrow takeexam(pseudonym_j,.)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$
if $correct(exam_{j_b},.) \neq 0$ **then**
$I \leftarrow correct(exam_{j_b},.)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$
else
return 0
end if
else
return 0
end if
else
return 0
end if
$d \leftarrow \mathcal{A}(knowledge\ of\ \hat{I}, \text{"}guess\ b\text{"})$
return d

The advantage of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{secr}_{ExS,\mathcal{A}}(.) = |\Pr[\mathbf{Exp}^{secr-0}_{ExS,\mathcal{A}}(.) = 1] - \Pr[\mathbf{Exp}^{secr-1}_{ExS,\mathcal{A}}(.) = 1]|.$$

Definition 1. A scheme for electronic exam ExS possesses property of **secrecy** if for any $\mathcal{A} \in PT^*$ the advantage $\mathbf{Adv}^{secr}_{ExS,\mathcal{A}}(.)$ is negligible.

Examinees are not allowed to deny an already submitted exam. Exam questions, answers and marks can not be altered and after submission no one is allowed to modify them.

In our experiment $\mathcal{A}$ has access to oracles *register*(.), *takeexam*(.), *correct*(.), *getidentity*(.) providing q samples of outputs. Adversary runs *register*(.), *takeexam*(.) and *correct*(.) with the system. During *takeexam*(.) and *correct*(.) the adversary colludes with EB, his aim is to modify data transmitted or generate favorable data. If an adversary succeeds in modification of honest participants' data or generating valid one, then experiment 2 outputs 1.

Experiment 2. **Correctness**

$\mathbf{Exp}^{corr}_{ExS,\mathcal{A}}(.)$
$\{\bar{I} \leftarrow \mathcal{A}^{register(.),takeexam(.),correct(.),getidentity(.)}\}_{i=1}^{q}$
for all $i \in EC$ **do**
if $register(i,.) \neq 0$ **then**
$I \leftarrow register(i,.)$
end if
end for
for all $j \in EX$ **do**
if $register(j,.) \neq 0$ **then**
$I \leftarrow register(j,.)$
if $takeexam(questions, pseudonym_j,.) \neq 0$ **then**
$I \leftarrow takeexam(questions, pseudonym_j,.)$

$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$
if $correct(exam_j, .) \neq 0$ **then**
$I \leftarrow correct(exam_j, .)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$
if $getidentity(pseudonym_j) \neq 0$ **then**
$I \leftarrow getidentity(pseudonym_j)$
else
return 0
end if
else
return 0
end if
else
return 0
end if
else
return 0
end if
end for
if $I \not\subseteq \hat{I}$ **then**
return 1
else
return 0
end if

The advantage of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{corr}_{ExS,\mathcal{A}}(.) = 2 \cdot \Pr[\mathbf{Exp}^{corr}_{ExS,\mathcal{A}}(.) = 1] - 1.$$

Definition 2. A scheme for electronic exam ExS possesses property of **correctness** if for any $\mathcal{A} \in PT^*$ the advantage $\mathbf{Adv}^{corr}_{ExS,\mathcal{A}}(.)$ is negligible.

Anonymity for examinees protects against partiality, meaning if an exam corrector knows whose exam he is correcting, he might be subjective in his evaluation. Anonymity of exam correctors prevents bribing and threatening attacks, such that if an examinee knows who will correct her exam, then she might pay the corrector some amount of money in order to get a better grade. Anonymity means exam correctors do not know which examinee's paper they are correcting and examinees do not know who corrects their papers, hence examinees and exam correctors are possible adversaries.

In our experiment $\mathcal{A}$ has access to oracles $register(.)$, $takeexam(.)$, $correct(.)$, $getidentity(.)$, gets all information from oracle queries. $\mathcal{A}$ chooses two examinees or exam correctors, runs algorithm $register(j_b, .)$ with $b \in \{0, 1\}$ chosen by the system. Since we give definitions for pseudonym-based e-exam schemes we assume that during registration $\mathcal{R}$ is honest. However after registration we assume $\mathcal{R}$ and EB might collude with $\mathcal{A}$ during $takeexam(.)$ and $correct(.)$. Knowing all information available $\mathcal{A}$ gives a guess for the chosen participant. Experiment 3 outputs 1, if $\mathcal{A}$ gives a good guess.

Experiment 3. **Anonymity**

$\mathbf{Exp}^{anon-b}_{ExS,\mathcal{A}}(.)$
$\{i \leftarrow \mathcal{A}(\text{"}choose\ examinees/exam\ correctors\text{"})\}_{i=1}^{q}$
$\{I \leftarrow \mathcal{A}^{register(i,.),takeexam(pseudonym_i,.),correct(.),getidentiy(.)}\}_{i=1}^{q}$
$(j_0, j_1) \leftarrow \mathcal{A}(\text{"}choose\ examinees/exam\ correctors\text{"})$
if $register(j_b, .) \neq 0$ **then**
$I \leftarrow register(j_b, .)$
if $takeexam(pseudonym_{j_b}, .) \neq 0$ **then**
$I \leftarrow takeexam(pseudonym_{j_b}, .)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$
if $correct(exam_{j_b}, .) \neq 0$ **then**
$I \leftarrow correct(exam_{j_b}, .;)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$
else
return 0
end if
else
return 0
end if
else
return 0
end if
$d \leftarrow \mathcal{A}(\text{"}guess\ b\text{"})$
return d

The advantage of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{anon}_{ExS,\mathcal{A}}(.) = |\Pr[\mathbf{Exp}^{anon-0}_{ExS,\mathcal{A}}(.) = 1] - \Pr[\mathbf{Exp}^{anon-1}_{ExS,\mathcal{A}}(.) = 1]|.$$

Definition 3. A scheme for electronic exam ExS possesses property of **anonymity** if for any $\mathcal{A} \in PT^*$ the advantage $\mathbf{Adv}^{anon}_{ExS,\mathcal{A}}(.)$ is negligible.

Let us detail the definition of *authenticity*. During the examination stage only eligible examinee's answers and eligible exam corrector's marks are considered. Registry verifies whether the sender is allowed to take or correct the exam. Eligible participants possess pseudonyms authorized by Registry.

In our definition we assume that adversaries collude with $\mathcal{R}$ after registration and grading stage, and with EB during examination. We state that an exam scheme provides authenticity if by the end of the evaluation stage the generated and processed valid pseudonyms are more with non-negligible probability than $\mathcal{R}$ has verified.

Experiment 4. **Authenticity** $\mathbf{Exp}^{auth}_{ExS,\mathcal{A}}(.)$
$\{\bar{I} \leftarrow \mathcal{A}^{register(.),takeexam(.),correct(.),getidentity(.)}\}_{i=1}^{q}$
for all $i \in EC$ **do**
if $register(i, .) \neq 0$ **then**
$I \leftarrow register(i, .)$
end if
end for
for all $(j \in EX)$ **do**
if $register(j, .) \neq 0$ **then**
$I \leftarrow register(j, .)$
if $takeexam(pseudonym_j, .) \neq 0$ **then**
$I \leftarrow takeexam(pseudonym_j, .)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$
if $correct(exam_j, .) \neq 0$ **then**
$I \leftarrow correct(exam_j, .)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB\text{"})$

if $getidentity(pseudonym_j) \neq 0$ **then**
$I \leftarrow getidentity(pseudonym_j)$
$\bar{I} \leftarrow \mathcal{A}(\text{"}control\ EB, R\text{"})$
else
return 0
end if
else
return 0
end if
else
return 0
end if
else
return 0
end if
end for
if $n + m < \hat{n} + \hat{m}$ **then**
return 1
else
return 0
end if

The advantage of an adversary $\mathcal{A}$ is

$$\mathbf{Adv}^{auth}_{ExS,\mathcal{A}}(.) = 2 \cdot \Pr[\mathbf{Exp}^{auth}_{ExS,\mathcal{A}}(.) = 1] - 1.$$

Definition 4. A scheme for electronic exam ExS has **authenticity** if for any $\mathcal{A} \in PT^*$ the advantage $\mathbf{Adv}^{auth}_{ExS,\mathcal{A}}(.)$ is negligible.

4. PROPOSED SCHEME

After giving experiment-based security definitions for electronic exam systems, we also propose an exam scheme that is secure under our definitions. Due to the restricted number of pages, we do not detail it here. You can find it in [9]. Our scheme is not strictly designed with concrete cryptographic primitives, we require only that the applied symmetric encryption scheme is IND-CPA secure [5], asymmetric encryption scheme is IND-CCA secure [5] and digital signature and message authentication scheme is secure against UF-CMA attacks [6]. The scheme is going to be implemented in a frame of project GOP-1.1.2-07/1-2008-0001.

4.1 Aknowledgement

The author is supported by TÁMOP 4.2.1-08/1-2008-003 project. The project is implemented through the New Hungary Development Plan co-financed by the European Social Fund, and the European Regional Development Fund. The author is partially supported by the project GOP-1.1.2-07/1-2008-0001 and also by the Hungarian National Foundation for Scientific Research Grant No. K75566.

5. REFERENCES

[1] László Aszalós, Philippe Balbiani *Logical aspects of user authentication protocols*, Proceeding of 7th seminar RelMiCS, 2nd Workshop Kleene Algebra (2003), pp. 277 – 287.

[2] László Aszalós, Philippe Balbiani *Some decidability result for logic constructed for checking user authentication protocols*, Journal of Computer Science and Control Systems (2008).

[3] J. Castella-Roca, J. Herrera-Joancomarti and A. Dorca-Josa, *A secure e-exam management system*, Proceeding of the First International Conference on Availability, Reliability and Security (ARES'06) (2006), pp. 864 – 871.

[4] ExamSoft Worldwide, http://www.examsoft.com.

[5] S. Goldwasser, S. Micali, *Probabilistic encryption*, Journal of Computer and System Sciences, (1984), **28**, pp. 270 – 299.

[6] S. Goldwasser, S. Micali and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal on Computing, (1988), **17(2)**, pp. 281 – 308.

[7] GurukulOnline Learning Solutions, http://www.gurukulonline.co.in/index.htm.

[8] J. Herrera-Joancomarti, Josep Prieto-Blazquez, J. Castella-Roca, *A secure electronic examination protocol using wireless networks*, International Conference on Information Technology: Coding and Computing (ITCC'04) **2** (2004), pp. 263 – 267.

[9] A. Huszti, *Experiment-based definitions for electronic exam systems*, to appear.

[10] A. Huszti, A. Pethő, *A Secure Electronic Exam System*, Publicationes Mathematicae Debrecen, **77(3-4)** (2010), pp. 299 – 312.

[11] Software Secure, *Securexam*, http://www.softwaresecure.com.

matcos-10

University of Primorska Press
www.hippocampus.si

ISBN 978-961-6832-10-6
Not for resale