Evolutionary Balancing of Healthy Meals

Barbara Koroušić Seljak
Computer Systems Department, Jožef Stefan Institute
Jamova 39, SI-1000 Ljubljana, Slovenia
Barbara.Korousic@ijs.si

Keywords: Optimization, multiconstrained fractional knapsack problem, genetic algorithms.

Received: July 7, 2004

In this paper we present an evolutionary algorithm for solving the nutrition problem of composing and balancing healthy meals. We treat this problem as a single-objective and multiconstrained fractional knapsack problem that is easy to formulate, yet, its decision problem is in the class of NP-complete problems. In other words, some heuristic algorithm is required to provide good problem solutions in reasonable (polynomial) computational time. We applied a genetic algorithm and modified its parameters to yield high-quality and reliable solutions (healthy and balanced meals) that respect multiple weakly-correlated dietary recommendations and guidelines and include as much seasonal functional foods as possible. Functional foods contain physiologically active compounds that provide health benefits beyond their nutrient contributions.

Povzetek: V članku predstavljamo evolucijski algoritem za reševanje problema optimalne sestave jedilnika.

1 Introduction

Nutrition is the process of nourishing, by which our body obtains nutrients and non-nutrients. These are chemical substances obtained from food and used in the body to provide energy, structural materials, and regulating agents to support growth, maintenance and repair of the body's tissues. Foods are composed of water and solids. Solid materials include carbohydrates, lipids, protein, vitamins, minerals and other compounds. Water, carbohydrates, lipids, protein, vitamins and some of the minerals found in foods are nutrients, while components of foods that contain alcohols, phytochemicals, pigments, aditives and others are non-nutrients. Some non-nutrients are beneficial (like flavonoids, isoflavones or lignans), some are neutral, and a few are harmful. Food choices influence our mental performance, emotional well-being and physical performance (health).

There exist many advices (dietary recommendations and guidelines) for proper nutrition based on objective scientific medical and nutrition research that may prevent and control nutritional deficiencies, infectious diseases and even chronic diseases [1]. They define the amounts of energy, nutrients and other dietary components that best support health. Although the intent of these advices may seem simple enough, they are the subject of much misunderstanding and controversy. Using a reliable nutrition software could help determine which advices should be applied and adjusted to meet our individual needs.

In this paper we introduce the mathematical aspect of such a nutrition software that is used as an optimization tool for composing and balancing healthy meals. A healthy meal provides sufficient energy and enough of all the nutrients and beneficial non-nutrients. Balancing a meal involves using enough, but not too much, of each type of food.

In Section 2 we provide a formulation of the problem of composing and balancing meals as a single-objective and multiconstrained fractional knapsack problem; in Section 3 we describe a genetic algorithm for the single-objective and multiconstrained knapsack problem; in Section 4 we evaluate the method; and in Section 5 we list conclusions and suggest possible future work.

2 Problem of balancing healthy meals

Finding an optimal composition of foods that could be served as a healthy and balanced meal is a complex problem for two reasons: there are many problem constraints (advices for proper nutrition) that are weakly-correlated, and the search space (the set of all possible combinations) is complex. The problem is even more difficult because the food quality may vary by season. Anyhow, solutions of the problem are trade-off solutions. For such solutions no improvement in any constraint is possible without violating at least one of other constraints.

We treat the problem of composing and balancing healthy meals as a single-objective and multiconstrained (multidimensional) fractional knapsack problem (MFKP) that is easy to formulate, yet it can be solved efficiently only by using some optimization techniques. It is also called the multi-knapsack problem. Many practical problems can be formulated as a multiconstrained knapsack problem, for example, the capital budgeting problem. Other applications of the problem include

allocating processors in a distributed computer system, project selection, and cutting stock problems. In our case, the MFKP is defined as follows.

Given food items of different values (qualities) and volumes (data about energy, nutrients, and non-nutrients), find the most valuable (healthy-and-balanced) composition of foods which fit in a knapsack (meal) of fixed volumes. Values are defined subjectively with respect to food functionality, seasonal availability and price. Knapsack volumes are defined by weakly-correlated dietary recommendations and guidelines, such as:

- recommended intakes of energy, nutrients and non-nutrients;
- adequate carbohydrates : protein : fat energy ratio;
- adequate ratio of essential fatty acids;
- recommended consumption of fruits and vegetables;
- restricted intake of fats and dietary cholesterol; etc.

2.1 Formal definition

We are given a knapsack of m capacities C_k for $k = 1, 2,..., m$, and n objects (food items). Each object has a value $v_i \in I^+$, $v_i > 0$, and a set of volumes $\omega_{i,k} \in R^+$, $\omega_{i,k} \geq 0$, one for each capacity. We would like to find a selection of objects $i \in I^+$, $i \geq 0$, such that $\sum_{i=1}^{n} \omega_{i,k} x_i \Theta C_k$ (Θ can be $\leq, =$ or $\geq$, $k = 1, 2,..., m$) and for which the total value, $\sum_{i=1}^{n} v_i x_i$, is maximized. The parameter $x_i = f_i P_i$ denotes the quantity of the selected object, where $f_i \in F$, F={0.25,0.5,0.75,1,1.5,2,3,…,10}, is a fraction of its portion size $P_i \in R^+$, $P_i \geq 0$.

The decision problem of the MFKP is NP-complete [2]. The only two exact algorithms that deliver optimum solutions to multiconstrained knapsack problems are based on the branch-and-bound [3] and the dynamic programming [4] approaches. On the other hand, heuristic methods for solving knapsack problems that have time complexity bounded by a polynomial in the size parameters of the problem have been known for many decades. A comprehensive review of the multiconstrained 0-1 knapsack problem and the associated heuristic algorithms is given by Chu and Beasley [5].

3 Genetic algorithm for the MFKP

We decided to compose and balance healthy meals in a heuristic way by using a genetic algorithm (GA) [6]. The theoretical foundations of this effective optimization technique were originally developed by Holland [7]. GAs are different from traditional optimization techniques as they simulate nature at a very abstract level to get solutions for a variety of demanding problems. They have been shown to be well suited for solving problems characterized by local minima. In recent years, a number of papers involving the use of GAs to solve multiconstrained knapsack problems have appeared.

The basic characteristic of GAs is that they search through an arbitrary search space both for exploration and exploitation purpose. The evolutionary process of biological organisms in nature is simulated by taking an initial population of individuals and applying genetic operators to the selected (normally highly-fit) individuals. Each individual in the population is encoded into a string (chromosome) that represents a possible (candidate) solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly-fit individuals are reproduced by exchanging genetic information with other highly-fit individuals. This produces new offspring solutions that replace either less-fit individuals or the whole population. This procedure is repeated until a satisfactory solution is found.

Although the exploration of the search space is driven by random decisions, GAs are far from random search routines. The random decisions made in GAs can be modelled using Markov chain analysis. In this way, it can be shown that GAs will converge to globally optimum solutions [8].

3.1 Direct encoding

The first step in designing a GA is to encode the candidate solutions to the problem. We applied a real-valued coding of candidate solutions to the problem.

Because people consume several thousands of food items, we decided to separate n objects into G groups, where $G \leq n$ and G is few tens. Creating a composite meal, we select at most one item from each group. Hence, in our representation, a chromosome contains G' pairs (i_g, x_{i_g}), where i_g denotes the code of the selected object from a food group g and $x_{i_g} = f_{i_g} \cdot P_{i_g}$ its quantity (Figure 1), f_{i_g} being the fraction size of i_g and P_{i_g} its default portion size. A null value $i_g = 0$ implies that no item is selected for a group g. Each of the G food groups may be omitted or repeated within a chromosome. Normally, food groups are selected with respect to the food guide pyramide [9], which is an outline of what to eat each day based on the dietary recommendations and guidelines. It is not a rigid prescription, but a general guide that let us choose healthful meals that are right for us.

The number of possible solutions is approximately $(\frac{n}{G})^{G'|F|}$, where n denotes the number of food items, G the number of food groups, G' the number of food groups captured within a chromosome and F the set of

fraction sizes. Normally, n is in the range of few hundreds to few ten thousands, G is few tens, and $|F|$ is approximately ten.

g	1		…	G'	
(ig, x_{ig})	1060 milk	0.5·244		19034 popcorn	5·8

Figure 1: Description of a chromosome (candidate solution).

In our implementation, the GA starts either with a random population of candidate solutions or a population of solutions (healthy meals) known from experience. The population size is few tens and remains constant over all populations.

3.2 Fitness evaluation

Each solution of the candidate population is evaluated using the following fitness (objective) function:

$$f(\vec{i}) = \sum_{g=1}^{G} v_{i_g} f_{i_g}, \quad (1)$$

where higher value of v_{i_g} means better food quality. The aim of the GA is to maximize this fitness function.

3.3 Infeasible solutions

A chromosome might represent an infeasible solution. This is a solution for which at least one knapsack constraint is violated. There are several ways of dealing with infeasible solutions in GAs [10], i.e.,

- by penalizing infeasible solutions,
- by incorporating a repair operator that transforms an infeasible solution into a feasible one, or
- by using an order-based encoding instead of a direct one.

In our case, we applied the first two approaches. First, we incorporated a penalty term into the fitness function (1) to penalize the fitness of infeasible solutions, without distorting the fitness landscape:

$$f(\vec{i}) = \sum_{g=1}^{G} v_{i_g} f_{i_g} - p(\vec{i}). \quad (2)$$

The penalty term $p(\vec{i})$ is defined in a static way by adding a metric based on a number of constraints violated:

$$p(\vec{i}) = \sum_{k=1}^{m} \mathrm{X}_k \cdot \delta_{\vec{i},k}, \quad (3)$$

where $\delta_{\vec{i},k} = \begin{cases} 1, \text{if } \vec{i} \text{ violates a constraint } k \\ 0, \text{if } \vec{i} \text{ satisfies a constraint } k \end{cases}$

and X_k is the weight of a constraint k.

The next step was to transform a percentage of infeasible solutions into feasible ones by using a greedy repair operator. This operator consists of the following phases:

1. Rank the problem constraints violated by the infeasible solution in the decreasing order of their weights X_k, $k = \{1, ..., m\}$;
2. For each violated problem constraint, starting with the most critical one, sort food items in the chromosome in
   a. the increasing order of their values, and
   b. the decreasing or increasing order of their weights for the exceeded or deceeded constraints, respectively;
3. For each food item in the sorted chromosome, starting with the weakest one, find an alternative item with better nutrient profile for a given constraint. The alternative is searched either in the item's neighborhood or by random in a given food group, depending on the probability of repair. Namely, food items are ordered in each food group so that similar foods are close to each other.
4. Repeat the local-improvement operation of Step 3 at a given repair rate, or until a given problem constraint is satisfied.
5. Go to Step 2.

To prevent premature convergence of the GA, some infeasible solutions were left unrepaired in the population. Allowing a small percentage of infeasible solutions to join the population is longed-for because optimum solutions frequently lie on the boundaries of feasible regions [11].

3.4 Parent selection

To create new candidate solutions, two chromosomes have to be selected from the current population as parents. In our implementation of the GA, best-ranked (highly-fit) feasible solutions are more likely to be selected for reproduction because we apply the elitism strategy, where a number of least-fit members of the current candidate population are interchanged with an equal number of the best-ranked chromosomes. This strategy increases the performance of the GA, because it prevents losing the best-found feasible solutions.

The parent selection is realized via the tournament approach. This is based upon an idea of forming two pools of candidate solutions, each consisting of the same number of chromosomes. Two solutions with the best fitness, each taken from one of the tournament pools, are chosen to be parents. Using a larger size of the tournament pools has the effect of increasing selection pressure on the more-fit solutions. The problem of getting stuck in a local-optimum solution can happen. To avoid this problem, we adopt the standard (binary) tournament selection technique and realize the elitism through the interchange ratio of least-fit to best-ranked solutions. This ratio is in the order of 4 or 6 down to 1 chromosome per population, depending on the population size.

3.5 Crossover and mutation

In crossover, a selected pair of chromosomes are mated to produce an offspring that replaces the least-fit solution in a given population if its fitness ranks above. This steady-state approach may perform better than generational GAs because it better retains feasible solutions found in the populations and may have higher selection pressure [12]. We apply a uniform crossover operator to produce a solution that preserves the genetic material from both parents. Each element of the offspring's chromosome (a pair of the food item's code and its quantity) is created by copying the corresponding element from one of the parents, chosen according to a binary random number generator $[0,1]$. In our implementation of the GA, using a two-point crossover operator can also perform crossover. In this approach, copying the corresponding elements from one parent, and all the others by copying the corresponding elements from other parent, creates elements of the offspring's chromosome between two points, selected by using a crossover probability.

Once the offspring has been generated through the selection and the crossover, mutation is performed on few randomly selected elements of the chromosome. Each element is mutated in one of the following ways chosen at random,

- by local-improvement operation of the greedy repair algorithm (Step 3, Section 3.3), i.e., the code of a given food item and its quantity are replaced with a close alternative item from the same food group and its recommended quantity (with the default fraction size of 1), respectively;
- by multiplying the size of the food item's portion by a randomly selected fraction factor from F={0.25, 0.5, 0.75, 1, 1.5, 2, 3, …, 10}.

The fixed rate of mutation is set to be a small value (in the order of 1 or 2 elements per chromosome).

3.6 Termination criteria

The GA terminates its operations when the system is assumed to be in a stable state, i.e., an optimum feasible solution has been found (a wanted-solution approach), or a certain number of populations have been generated and evaluated (a time-out approach).

4 Evaluation of the GA for the MFKP

In order to evaluate the proposed evolutionary method for composing and balancing healthy meals, we optimized a set of randomly generated meals using the GA. We used the USDA nutrition database, Release 16 [13], which is the major source of food composition data in the United States and is available free of charge. It includes nutrient profiles for more than 6500 food items that are grouped into 23 food groups. The items are ordered so that similar foods are grouped together. Each nutrient profile contains more than 30 values, such as macronutrients, elements, vitamins, etc. We used these data as weights. Values of food items were defined subjectivelly, considering their functionality and availability in a given season. Seasonal foods from the list of functional foods (e.g., apples, broccoli and red wine in automn, avocado in winter, blueberries in summer, etc.) were assigned the highest values. We considered the following representative dietary recommendations and guidelines for:

- the energy value of a meal;
- the carbohydrates : protein : fat energy ratio;
- the intake of dietary cholesterol per meal;
- the intake of vitamin E per meal;
- the linoleic (omega-6) fatty acid : alpha-linoleic (omega-3) fatty acid ratio.

These constraints are equality constraints, except the one on dietary cholesterol that is an inequality (less-than-equal) constraint.

We developed software that implements the GA for composing and balancing healthy meals using the Borland Delphi programming tool. It runs under the Microsoft Windows operating systems on a Pentium PC. The USDA database is used in a Microsoft Access format.

4.1 Experiments and results

After several runs of the program, the most advantageous settings for the GA were defined (Table1), and a set of good solutions to the problem of composing and balancing healthy meals was collected.

Table1: Settings for the GA parameters.

Parameter	*Value*
Population size	20
Repair probability	0.5
Repair rate	2
Elitism ratio	$0.2 \div 0.05$
Tournament pool size	2
Crossover probability	0.7
Mutation rate	0.05
Termination criterion	1000 evaluations

It has proved that a repair operator has to be used in addition to the static penalty function. Otherwise, the search gets stuck in a local minimum without finding a good feasible solution before the termination happens. We estimated that approximately each third candidate solution was infeasible and required repairing. Although the worst-case time complexity of the repair algorithm is $\sum_{g=1}^{G'} O(n_g \cdot m)$, where n_g denotes the number of food items in a food group g and m the number of constraints, in practice $O(m)$ steps were needed to find a close feasible solution. However, the most difficult task in greedy repair was to derive the proper constraint weights X_k, $k = \{1, ..., m\}$ because some of the constraints are weakly-correlated. We defined the highest weight to the

constraint on dietary cholesterol and the lowest to the constraint on the energy intake. In between were the second and the fourth constraints, i.e., the ratios of macronutrients and essential fatty acids, respectively.

In Tables 2 and 3, a good (feasible) daily-meal solution generated from an initial population of random candidate solutions that satisfies the selected problem constraints and its nutritional profile are presented. In fact, these initial candidate solutions were all infeasible. In the solution, the quantities of foods were selected as multipliers of the portion sizes. Larger portions of more than 100 grams were multiplied by a fraction factor from $\{0.25, 0.5, 0.75, 1, 1.5, 2\}$ and smaller portions of few grams by a factor from $\{1, ..., 10\}$, respectively. In Table 2, selected foods are specified with a short description instead of a code.

Table 2: A good daily-meal solution.

Grams	*Food item*
	Breakfast
72	ORANGE DRK,BRKFST TYPE,W/ JUC & PULP, FRZ CONC
38	CEREALS,MALTEX,DRY
	Lunch
141	WEIGHT WATCHERS ON-THE-GO CHICK,BROCLI&CHDR POCKT SNDWCH,FRZ
	Supper
496	SOUP,TOMATO,LO NA,W/H2O
80	FAST FOODS,POTATO,MASHED
31	TURKEY,YOUNG HEN,SKN ONLY,CKD,RSTD
1	GINGER,GROUND
123	BEETS,CND,REG PK,SOL&LIQUIDS
28	CAKE,CHERRY FUDGE W/CHOC FRSTNG
31	ENSURE PLUS,LIQ NUTR
	Snack
150	BANANAS,RAW
21,5	SOYBEANS,MATURE SEEDS,RSTD,SALTED
	Dinner
195	RICE,BROWN,LONG-GRAIN,CKD
85	MACKEREL,ATLANTIC,RAW
42	ALMOND PASTE

Table 3: Nutritional profile of the daily-meal solution.

.

RESULTS	*Recommended*	*Achieved*
Value	$\geq (3 \cdot 0{,}5 \cdot 23)$	✓
Energy (Kcal)	1800 ± 100	1875,7
Protein (% of energy)	$10 \div 15$	14,5
Total lipids (% of energy)	$20 \div 30$	29,8
Carbohydrates (% of energy)	$50 \div 60$	55,7
Dietary cholesterol (mg)	≤ 300	120,4
Vitamin E (mg)	15 ± 2	14,1
Saturated FAs (% of energy)	≤ 10	7,5
ω-6 FA + ω-3 FA (g)	$(11 + 5{,}5) \pm 2$	17,4

In Figure 1 performance of the proposed balancing method, based on measurements of the number of times the candidate solutions were evaluated to come within a certain fraction of the optimum, are presented. From the direct comparison, it can be seen that the repair technique for dealing with infeasible solutions performed better than the penalty one. Namely, repair of infeasible candidate solutions assured faster generation of best-ranked feasible solutions.

The program takes 8 seconds to generate an initial population of 20 candidate solutions and 300 milliseconds to calculate the quality, the fitness and the penalty of a given solution. Repair is more time consuming as each local-improvement operation (Steps 3 and 4, Section 3.3) takes 8 seconds at most, replacing one third of objects in a given infeasible solution at the selected repair rate. Considering one third of infeasible candidate solutions in each population, the total repair time is approximately 1 minute per population.

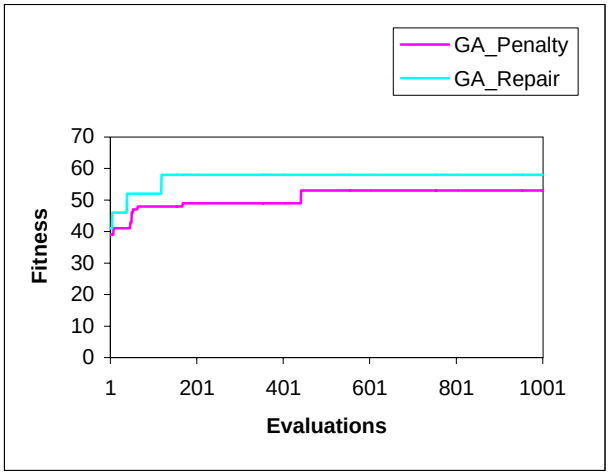

Figure 1: Performance of the meal composing and balancing GA method.

5 Conclusions

In the paper we have presented a heuristic method for composing and balancing healthy meals that considers several constraints and the quality of foods, with the tendancy of including as much as seasonal functional foods as possible. The method is based on a steady-state genetic algorithm that is a particular evolutionary algorithm. It uses direct (real-valued) coding of problem solutions, the elitism and the tournament approach for selection of parents, uniform and two-point crossover, and local-improvement mutation. All the GA parameters are fixed for all populations and candidate solutions. Infeasible solutions are penalized by a static function based on the number of constraints violated. Some infeasible solutions are further repaired so that the least-quality food items are replaced with more appropriate items or their quantities are modified by a multiple of the predefined portion size. Repair is performed in a greedy way. We collected the experimental results by running the GA from an initial population of random candidate solutions. From the results it has been shown that the problem of infeasible solutions could be solved more efficiently by applying a repair operator than merely by

penalizing. We also proved that few infeasible solutions are welcome to the population.

Including some real meals known to be healthy from experience into the initial population, the method could perform better. We could also experiment with other (self-adaptive) penalty functions to reduce the cost of repair, and with an order-based encoding of candidate solutions to the problem. Last but not least, the USDA database need to be replaced with the Slovene national nutritional database.

Acknowledgement

The work presented in this paper has been supported by the Slovenian Ministry of Health (project Development of a public domain server application for food analysis and optimization that considers modern dietary recommendations).

References

[1] POKORN, D., *Dietetika*, DZS, 1999 (in Slovene).

[2] GAREY, M.R. and JOHNSON, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[3] GAVISH, B. and PIRKUL, H., "Efficient Algorithms for Solving Multiconstrained Zero-One Knapsack problems to Optimality", *Mathematical Programming* 31, pp. 78-105, 1985.

[4] SOYSTER, A.L., LEV, B., and SIVKA, W., "Zero-One Programming with Many Variables and Few Constraints", *European Journal of Operational Research* 2, pp. 195-201, 1978.

[5] CHU, P.C. and BEASLEY, J.E., "A Genetic Algorithm for the Multidimensional Knapsack Problem", Journal of Heuristics, 4: 63-86, 1998.

[6] GOLDBERG, D.E., *Genetic Algorithms in Search, optimization, and Machine Learning*. Addison-Wesley, 1989.

[7] HOLLAND, J.H., *Adaptation in Neutral and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.

[8] KARR, C.L., YAKUSHIN, I., and NICOLOSI, K., "Solving inverse initial-value, boundary-value problems via genetic algorithm", *Eng. Applicat. Artif. Intell., 13(6):625-633*, Dec. 2000.

[9] WHITNEY, E.N., CATALDO, C.B., and ROLFES, S.R., *Understanding Normal and Clinical Nutrition*. Wadsworth, Thomson Learning, 2002.

[10] MICHALEWICZ, Z., *Genetic Algorithms+Data Structures=Evolution Programs*. Springer Verlag, 1996.

[11] SIEDLECKI, W. and SKLANSKY, J., "Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition". In *Proc. of the Third International Conference on Genetic Algorithms*, pp.141-150, 1989.

[12] CHAFEKAR, D., XUAN, J., and RASHEED, K., "Constrained Multi-Objective Optimization Using Steady State Genetic Algorithms". In *Erick Cantú-Paz et al (Editors): Genetic and Evolutionary Computation---GECCO 2003, Proceedings, Part I*, pp. 813--824, Springer. Lecture Notes in Computer Science, Vol. 2723, July 2003.

[13] http://www.nal.usda.gov/fnic/foodcomp