A Novel Artificial Intelligence Method for Weekly Dietary Menu Planning

B. Gaál, I. Vassányi, G. Kozmann
Department of Information Systems, University of Veszprém, Hungary

Summary

Objectives: Menu planning is an important part of personalized lifestyle counseling. The paper describes the results of an automated menu generator (*MenuGene*) of the web-based lifestyle counseling system Cordelia that provides personalized advice to prevent cardiovascular diseases.

Methods: The menu generator uses genetic algorithms to prepare weekly menus for web users. The objectives are derived from personal medical data collected via forms in Cordelia, combined with general nutritional guidelines. The weekly menu is modeled as a multi-level structure.

Results: Results show that the genetic algorithm-based method succeeds in planning dietary menus that satisfy strict numerical constraints on every nutritional level (meal, daily basis, weekly basis). The rule-based assessment proved capable of manipulating the mean occurrence of the nutritional components thus providing a method for adjusting the variety and harmony of the menu plans.

Conclusions: By splitting the problem into well determined sub-problems, weekly menu plans that satisfy nutritional constraints and have well assorted components can be generated with the same method that is for daily and meal plan generation.

Keywords
Genetic algorithms, multi-objective optimization, nutrition counseling

Methods Inf Med 2005; 44: 655–64

1. Introduction

The Internet is a common medium for lifestyle counseling systems. Most systems provide only general advice in a particular field; others employ forms to categorize the user in order to give more specific information. They also often contain interactive tools for menu planning [1].

The aim of the *Cordelia* project [2] is to promote the prevention of cardiovascular diseases (CD), identified as the leading cause of death in Hungary, by providing personalized advice on various aspects of lifestyle, an important part of which is nutrition.

MenuGene, the automated menu planner integrated with *Cordelia* uses the computational potential of today's computers, which offers algorithmic solutions to hard problems. The quality of these computer-made solutions may be lower than those of qualified human professionals, but they can be computed on demand and in unlimited quantities. Nutrition counseling is one of these kinds of problems. Human professionals possibly surpass computer algorithms in quality, although research comparing performance has been ongoing since the 1960's.

In 1964 Balintfy developed a linear programming method for optimizing menus [3] and Eckstein used random search to satisfy nutrient constraints [4]. Later, artificial intelligence methods were developed mostly using Case-Based Reasoning (CBR) or Rule-Based Reasoning (RBR) or combining the two with other techniques [5]. A hybrid CBR-RBR system CAMPER [6] integrates the advantages of the two independent implementations: the case-based menu planner, CAMP [7] and PRISM [8]. A more recent CBR approach is MIKAS, menu construction using incremental knowledge acquisition system [9]. MIKAS allows the incremental development of its knowledge base. Whenever the results are unsatisfactory, an expert will manually modify the system-produced diet [10]. A web-based system that models the workflow of dietitians has recently been built in Malaysia for dietary menu generation and management [11].

The core idea of our algorithm is the hierarchical organization and parallel solution of the problem. Through the decomposition of the weekly menu planning problem, nutrient constraints can simultaneously be satisfied on the level of meals, daily plans and weekly plans. This feature, which is a novelty, makes the implementation of our method instantly applicable in practice.

2. Objectives

There is no generally accepted method for producing a good menu. Additionally, a menu plan, whether it is weekly, daily or single meal, can only be evaluated when it is fully constructed. So the basic objective of our work is to design a menu planner that also includes some method to evaluate menu plans.

2.1 Evaluation of Menu Plans

The evaluation of a meal plan has at least two aspects. Firstly, we must consider the quantity of nutrients. There are well defined constraints for the intake of nutrient components such as carbohydrates, fat or protein which can be computed for everybody, given his/her age, gender, body mass, type of work, age and diseases. Optimal and extreme values can be specified for each nutrient component. So as for quantity, the task

Received: July 29, 2004; accepted: May 16, 2005

of planning a meal can be formulated as a constraint satisfaction and optimization problem.

Secondly, the harmony of the meal's components should be considered. Plans satisfying nutritional constraints should also be appetizing. The dishes of a meal should go together. By common sense some dishes or nutrients do not appeal in the way others do. This common sense of taste and cuisine can be described by simple rules recording the components that should fit together.

There could be conflicting numerical constraints or harmony rules. A study found that menus made by professionals may fail to satisfy all of the nutrient constraints [12].

2.2 Calculation of Personalized Objectives

The information collected via web forms in *Cordelia* explores controllable and uncontrollable risk factors for CD. Controllable risk factors include smoking, high blood pressure, diabetes, high cholesterol level, obesitas, lack of physical activity, stress and oral contraceptives. Uncontrollable factors considered are age, gender and family CD history. Based on the answers, the user is classified, the classification being a combination of factors like weight, high cholesterol, etc.

MenuGene uses the user's classification and all other useful observations (like the gender) and personal preferences (set by the user) to plan daily and weekly menus. This information is used to design the actual runtime parameters (objectives) of the menu to be generated when *MenuGene* is run. The nutritional allowances are looked-up from a table similar to Dietary Reference Intakes (DRI) [13, 14].

The fact base of *MenuGene* was loaded with the data of a commercial nutritional database, developed especially for Hungarian lifestyle and cuisine, that at present contains the recipes of 569 dishes with 1054 ingredients. The database stores the nutritional composition of the ingredients. The recipes specify the quantity of each ingredient in the meal, so the nutrients of a meal can be calculated by summation. At present, the nutrients contained in the database for each ingredient are energy, protein, fat, carbohydrates, fiber, salt, water, and potassium. Additionally, the database contains the categorization of the ingredient as either of the following: cereal, vegetable, fruit, dairy, meat or egg, fat and candy. This classification is used by *MenuGene* to check whether the overall composition (with respect to the ratio of the categories) conforms to the recommendations of the "food pyramid".

3. Methods

MenuGene uses genetic algorithms for the generation of dietary plans. A genetic algorithm (GA) is an algorithm used for the solution of difficult problems by the application of the principles of evolutionary biology and computer science. Genetic algorithms use techniques such as inheritance, mutation, natural selection and recombination derived from biology. In GAs a population of abstract representations of candidate solutions (also called chromosomes, genomes or individuals) evolves toward better solutions. The evolution starts with a population containing random individuals and happens in generations, in which stochastically selected individuals are modified (via recombination or mutation) to form the population of the next iteration. The attributes (also called alleles) of the chromosomes contain the information where each attribute represents a property. Genetic algorithms are used widely in the medical field [15-18].

GAs showed their strength in satisfying optimization problems; therefore we examined their efficiency in the generation of meal plans meeting quantitative nutrition constraints. Test software was developed to analyze the adequacy of GAs. Experiments are highlighted in the results section.

3.1 Evolutionary Operators

In order to start the search process, we first need an initial population. This may be created randomly or may be loaded from a database containing solutions of similar cases (Case Based Reasoning). The population in our tests contained 40 to 200 individuals, which are meals if we plan a single meal, daily menus if we plan a daily menu etc.

In the case of a meal plan, the population contains meals, the attributes of which are dishes. Then, in each iteration step, we perform a sequence of the evolutionary operators (crossover, mutation, selection) on the individuals. We stop the evolution process after a maximum of 1000 cycles or when no significant improvement could be achieved. The best individual of the final population is selected as the solution.

The evolutionary operators are presented through the following example. A regular Hungarian lunch consists of five parts: 1) a soup, a main dish of 2) a garnish (e.g. mashed potatoes) and 3) a topping (e.g. a slice of meat), 4) a drink, and 5) a dessert. So, a solution for a regular Hungarian lunch contains five attributes. The crossover (also called recombination) operator involves two solutions and it means that starting at a random point, their attributes are swapped. For example, if the starting point is the last attribute, then crossover means the exchange of the desserts. Mutation replaces a randomly selected dish with another one of the right sort (e.g. a soup with another soup).

Table 1 The crossover operator is shown in the table in function of the nutritional level. Legend: M – Monday, Tu – Tuesday, W – Wednesday, Th – Thursday, F – Friday, Sa – Saturday, Su – Sunday, BF – breakfast, MS – morning snack, L – lunch, AS – afternoon snack, D – dinner, S – soup, G – garnish, T – topping, Dr – drink, De – dessert, cp – crossover point

level	● parent 1	○ parent 2	◐ offspring 1	◑ offspring 2	cp
weekly	M Tu W Th F Sa Su	M Tu W Th F Sa Su	M Tu W Th F Sa Su	M Tu W Th F Sa Su	3
daily	BF MS L AS D	BF MS L AS D	BF MS L AS D	BF MS L AS D	3
meal	S G T Dr De	S G T Dr De	S G T Dr De	S G T Dr De	4
abstract	◇ ◇ ◇ ◇	◇ ◇ ◇ ◇	◇ ◇ ◇ ◇	◇ ◇ ◇ ◇	2

The single point crossover (recombination) operator is shown in Table 1. Recombination is done by randomly choosing a crossover point (cp) and creating two offsprings by exchanging the attributes of the solutions from that point on. On the weekly level, the attributes of the solutions represent daily menu plans. In the example in Table 1 we apply crossover to weekly level solutions, with a randomly chosen crossover point (cp = 3). The first offspring will contain the daily menu plans for Monday, Tuesday and Wednesday from the first parent and Thursday, Friday, Saturday and Sunday from the second parent. The genetic operators work on the abstract solution and attribute classes and do not operate on problem-specific data, thus the same method is used on every level.

3.2 Fitness Function

Whenever a new individual (offspring) is created by mutation or recombination, the fitness function assesses it according to its goodness. As for numerical constraints, the fitness function has to filter out solutions having an inadequate amount of nutrients. The *fitness* of a solution is defined by the sums of functions composed of four quadratic curves that take their maximum (0) at the specified optimum parameters, and break down abruptly over the upper and lower limit parameters (see Fig. 1.). For example, if a set of constraints (upper and lower limit, optimal value) is separately defined for carbohydrates and proteins, then the fitness is a sum of the two values taken from the two penalty functions for the carbohydrate and protein curves at the respective amounts. The actual fitness value bears no concrete physical meaning; it is used only for comparison and selection. The individuals with the highest (i.e. closest to zero) fitness values are considered the best solutions for the search problem.

The penalty function is thus designed that it should not differentiate small deviations from the optimum but be strict on values that are not in the interval defined by the constraints. The function is non-symmetric to the optimum, because the effects of the deviations from the optimal value can be different in the negative and positive case. These penalties have been derived from the manual assessment methods of our nutritional expert. This sort of penalty-based fitness function is also often applied in other multi-objective optimization techniques [19].

After the goodness is computed as a function of the numerical constraints, the fitness function examines whether the attributes of the solution are well assorted. Rules are used for classification according to the harmony of the components.

Each rule has two parts: conditions and fitness modification value. The general form of the rules is $r_i = \langle \text{condition}_1, ..., \text{condition}_n, \text{fitness modification value} \rangle$. The fitness value (which is less or equal to zero) should be divided by the modification value so that if less than one, the fitness will decrease.

Our nutrition expert organizes every food in our database into sets. The structure of these sets are very similar to those of PRISM [8].

⟨meat⟩ → ⟨white meat⟩|⟨red meat⟩

⟨white meat⟩ → ⟨chicken⟩|⟨fish⟩,
⟨red meat⟩ → ⟨beef⟩|⟨porc⟩,

⟨lunch⟩ →
⟨lunch_with_white_meat⟩|⟨lunch_with_red_ meat⟩|⟨vegetarian_lunch⟩

The conditions part of the rules on the level of meals contains one or more dish sets (e.g. dry top) or specific dishes (e.g. tomato soup). Some example rules might look like these:

r_1 = ⟨dry top, dry garnish, 0.75⟩,
r_2 = ⟨tomato soup, tomato drink, 0.6⟩,
r_3 = ⟨dry top, dry garnish, pickles, 0.8⟩,
r_4 = ⟨candy, 0.7⟩

Rule r_2 means that for each meal that contains tomato soup and tomato drink the system will replace the fitness with 60% of the original value. Rule r_3 penalizes the simultaneous occurrence of three dishes while rule r_4 penalizes solutions with any kind of candy.

Rules may be applied and configured for any level (e.g. daily level, meal level) of the algorithm. So, if the above rule r_2 is applied at the daily level, it will reduce the fitness to 60% of those daily menus that contain tomato soup and tomato drink anywhere in their meals.

Only the most appropriate rules are applied. For example, if we have a meal plan that contains tomato soup and tomato drink then from the rules r_2 = ⟨tomato soup, tomato drink, 0.6⟩ and r_5 = ⟨soups, tomato drink, 0.72⟩ only the former is applied because it is more specific. Only the strictest rule is applied from two or more rules with the same condition parts.

3.3 Divide and Conquer

There are several common features in the different nutritional levels (meal plan, daily plan, weekly plan). For example, both a meal and a daily plan can be considered a solution of a GA, where the attributes of daily plans are meals and the attributes of meals are dishes (see Fig. 2.). The problem

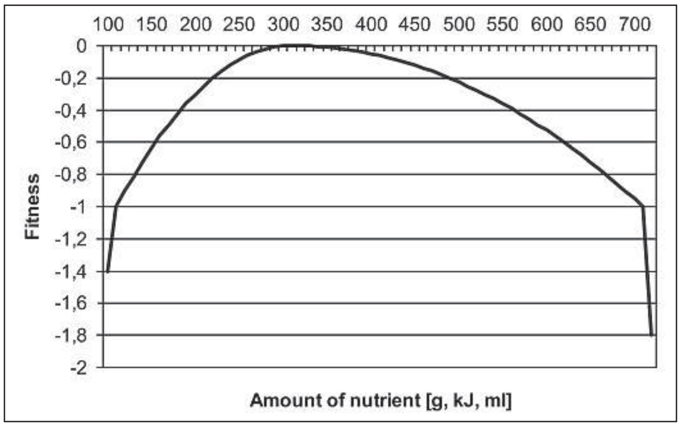

Fig. 1
The Fitness function with optimum = 300, lower limit = 100, upper limit = 700

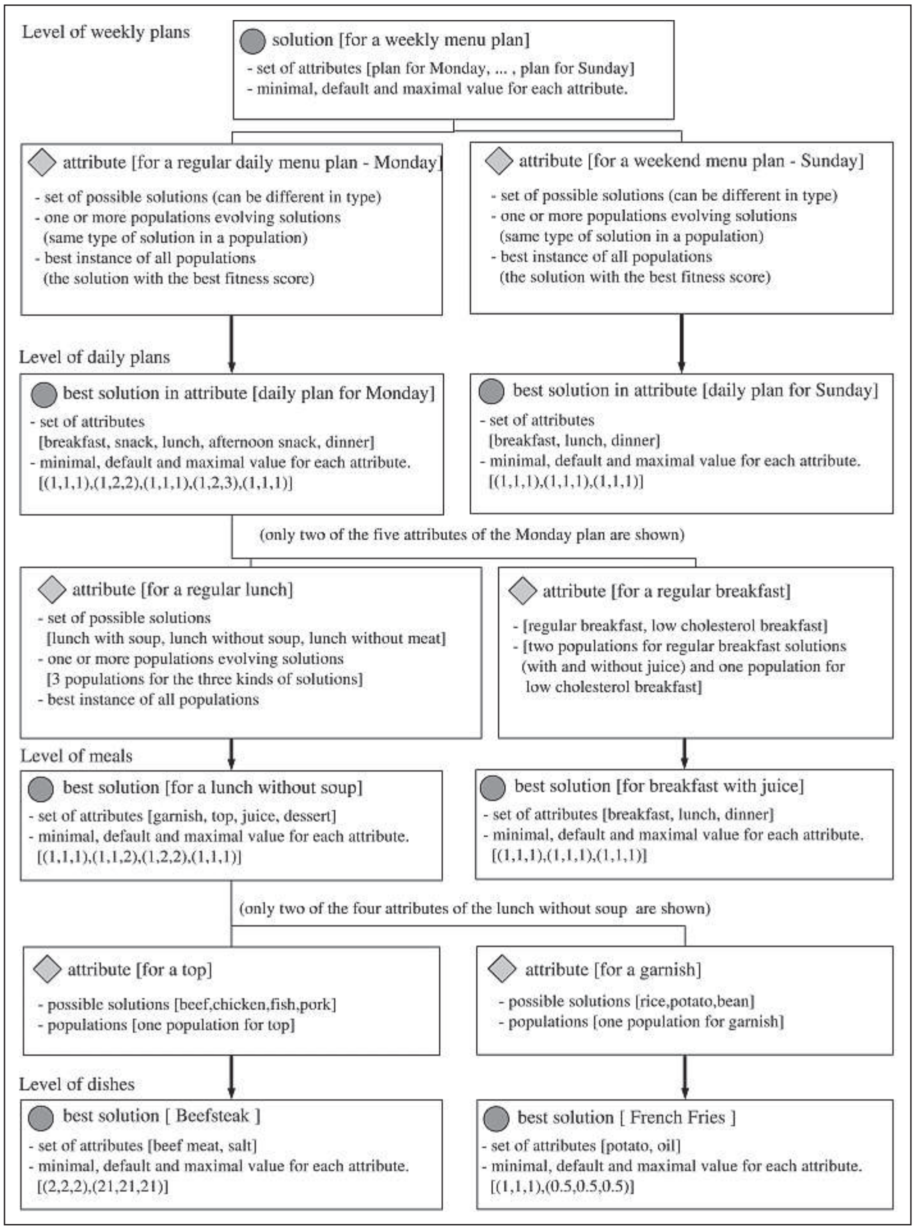
Level of weekly plans
solution [for a weekly menu plan]
- set of attributes [plan for Monday, ... , plan for Sunday]
- minimal, default and maximal value for each attribute.
attribute [for a regular daily menu plan - Monday]
- set of possible solutions (can be different in type)
- one or more populations evolving solutions
(same type of solution in a population)
- best instance of all populations
(the solution with the best fitness score)
attribute [for a weekend menu plan - Sunday]
- set of possible solutions (can be different in type)
- one or more populations evolving solutions
(same type of solution in a population)
- best instance of all populations
(the solution with the best fitness score)
Level of daily plans
best solution in attribute [daily plan for Monday]
- set of attributes
[breakfast, snack, lunch, afternoon snack, dinner]
- minimal, default and maximal value for each attribute.
[(1,1,1),(1,2,2),(1,1,1),(1,2,3),(1,1,1)]
best solution in attribute [daily plan for Sunday]
- set of attributes
[breakfast, lunch, dinner]
- minimal, default and maximal value for each attribute.
[(1,1,1),(1,1,1),(1,1,1)]
(only two of the five attributes of the Monday plan are shown)
attribute [for a regular lunch]
- set of possible solutions
[lunch with soup, lunch without soup, lunch without meat]
- one or more populations evolving solutions
[3 populations for the three kinds of solutions]
- best instance of all populations
attribute [for a regular breakfast]
- [regular breakfast, low cholesterol breakfast]
- [two populations for regular breakfast solutions
(with and without juice) and one population for
low cholesterol breakfast]
Level of meals
best solution [for a lunch without soup]
- set of attributes [garnish, top, juice, dessert]
- minimal, default and maximal value for each attribute.
[(1,1,1),(1,1,2),(1,2,2),(1,1,1)]
best solution [for breakfast with juice]
- set of attributes [breakfast, lunch, dinner]
- minimal, default and maximal value for each attribute.
[(1,1,1),(1,1,1),(1,1,1)]
(only two of the four attributes of the lunch without soup are shown)
attribute [for a top]
- possible solutions [beef,chicken,fish,pork]
- populations [one population for top]
attribute [for a garnish]
- possible solutions [rice,potato,bean]
- populations [one population for garnish]
Level of dishes
best solution [Beefsteak]
- set of attributes [beef meat, salt]
- minimal, default and maximal value for each attribute.
[(2,2,2),(21,21,21)]
best solution [French Fries]
- set of attributes [potato, oil]
- minimal, default and maximal value for each attribute.
[(1,1,1),(0.5,0.5,0.5)]

Fig. 2 The multi-level structure of the algorithm. Concrete examples are in square brackets.

of generating weekly menus can be divided into sub-problems, which in turn can be solved the same way using GAs. Recently a similar approach, a multi-level GA, was presented and tested on a multi-objective optimization problem [20]. For solving the problem, we created a C++ framework called GSLib, which uses the functions provided by the GALib [21] genetic algorithm library for running a standard, parameterized evolution process on the current population. GSLib is abstracted from the menu generation problem and uses abstract classes such as "solution" and "attribute" to represent the information related to the optimization and constraint satisfaction problems. The role of GSLib is the algorithmic setup, and the multi-level divide/conquer style scheduling and operation of the abstract evolution processes at various levels in the test system Menugene. For a description of possible scheduling strategies see Section 3.4.

Because of the abstract framework, every kind of meal can be represented and every kind of plan can be generated irrespectively of national cuisine, eating habits and nutrition database. For example, in Hungarian hospitals, "hot dinners" containing at least a hot soup are served two times a week. A solution type for this kind of weekly plan can be easily recorded in our database. It would contain five "regular daily plan with cold dinner" attributes for every day except Thursday and Sunday where there would be attributes for "regular daily plan with hot dinner". Another example would be a school cafeteria, where only breakfasts, morning snacks and lunches are served. Five attributes for "daily plan for school refectories" would make up the solution for a weekly plan (only for weekdays).

3.4. Simultaneous Generation of Different Level Menu Plans

The generation of weekly menu plans could be done in sequential form. Meal level GAs could create meals from dishes and these meals could then be used as a fact base to generate daily plans. The same applies for weekly menu plan generation. However, our method can run the different level GAs in a concerted manner. Table 2 gives an overview of the currently implemented *mutation-based* GA scheduling strategy and other alternatives as well.

Columns 4 and 5 of Table 2 show the sequence of how GSLib fires the evolution processes on the various levels for the top-down and bottom-up strategies, respectively. For example, the top-down strategy starts with an initial population (loaded randomly or from the case-base) and goes on by evolving the weekly level for a given number of iteration steps (multi-level iteration step: 1). Then, the evolution proceeds on the level of daily plans (2) by evolving the attributes of the weekly menu plans. After the second level, the process continues on the level of meals (3), and after that, the evolution restarts from the weekly level (4). The multi-level scheduling for credit propagation and mutation-based strategy are also described in Table 2. In any strategy, there is no evolution on the level of dishes, nourishments, and nourishment components. The estimated number of instances of the solution and attribute classes is shown in

Table 2 The multi-level GA scheduling strategies (top-down, bottom-up, credit propagation, mutation-based) in function of algorithmic levels

Level	Object	Number of instances	Top-down			Bottom-up			Credit propagation	Mutation-based
Weekly	Population	1	1.	4.	7.	3.	6.	9.	Start with a defined amount of credit and decide on each level what to use it for 1. for evolving the current level	Fire evolution on lower levels when mutation occurs
	Sol	40								
	Attr	280								
Daily	Sol	14000	2.	5.	8.	2.	5.	8.	2. use part of the credit and share the other part among the lower level objects	
	Attr	70000								
Meal	Sol	84e+05	3.	6.	9.	1.	4.	7.	3. share all of the credit among the lower level objects	normal mutation
	Attr	336e+05								
Dish	Sol	756e+07	No evolutionary process on these levels							
	Attr	6804e+07								
Nourishment	Sol	68e+09								
	Attr	544e+09								
Nourishment component	Sol	544e+09								

column 3. Note that by exploiting a copy-on-write technique, most instances of the classes are virtual and stored in the same memory location.

In contrast to the sequential method, we keep all of the adjustable parameters of the various levels in the memory to provide a larger search space in which generally better solutions can be found. So, if a GA evolving daily menu plans can't satisfy its constraints and rules, its fact base (which consists of meals) can be improved by further evolving the populations on the level of meals.

3.5 Case-based Initialization of Initial Populations

Solutions found for a given problem (for example: low cholesterol breakfast plans) can be reused by loading them to the appropriate initial populations. We therefore store some of the best solutions found for each sub-problem. Whenever Menugene begins to create a new plan which should satisfy some constraints, it searches its database for solutions that were generated with similar constraints and loads them as an initial/startup population.

4. Results

4.1 Optimal Crossover and Mutation Rates

We performed tests to explore the best algorithmic setup. First we analyzed the effect of the crossover and mutation probabilities on fitness. We used the same randomly generated initial populations for the tests, and averaged the results of ten runs in each configuration.

The results showed that while the probability of the crossover does not influence the fitness too much, a mutation rate well above 10% is desirable, particularly for smaller populations. This result is surprising at first, as the literature of GA generally does not recommend mutation rates above 0.1 … 0.5%. However, due to the relatively large number of possible alleles, we need high mutation rates to ensure that all candidate alleles are actually considered in the evolution process.

4.2 Fast Convergence to Best Available Solution

Runtime performance is an important factor for MenuGene as it must run as an on-line service in Cordelia. Runtime is determined by the number of operations to be per-

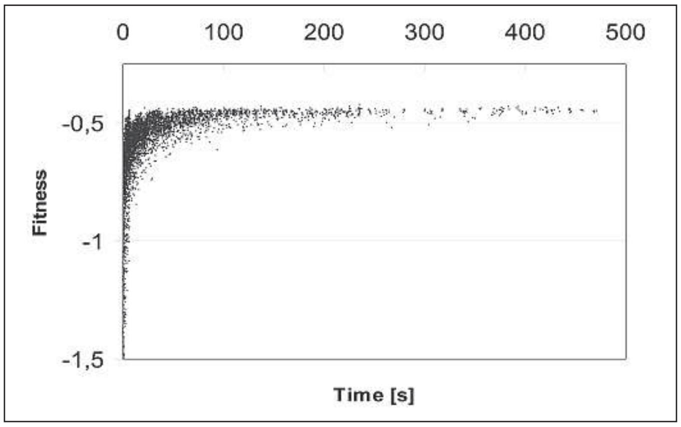

Fig. 3
Runtime of *MenuGene* in various algorithmic set-ups. Each dot represents the result of a test run.

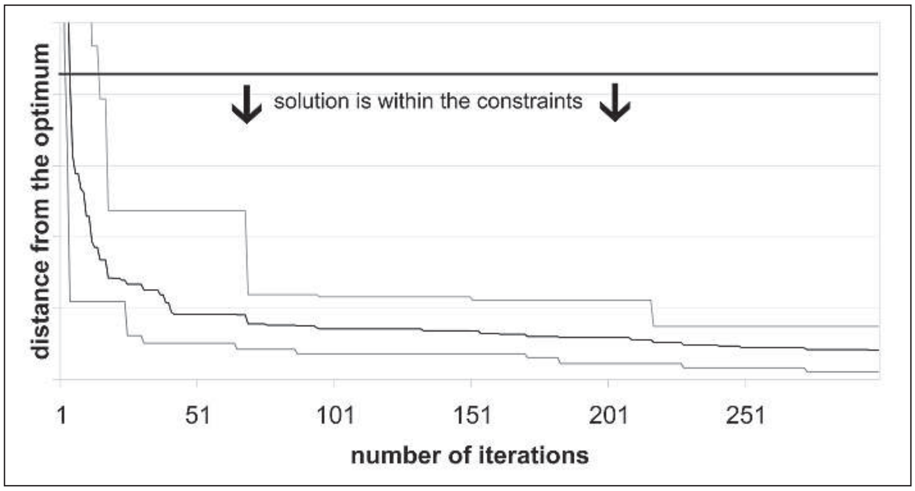

Fig. 4 Distance from the optimum for the worst (upper curve), average (middle curve) and best (lower curve) solution of ten runs, in the function of the number of iterations.

formed in each generation. Not surprisingly, we observed a strong linear connection between the probabilities of mutation and crossover, and runtime. However, our main concern is the quality of the solution. So we examined the connection between the runtime and the quality of the solution in a wide range of algorithmic setups (adjusted parameters were number of iteration, population size, probability of mutation and crossover).

As Figure 3 shows, although the quality of the solutions improves with longer runtimes (whether it is the effect of any of the adjusted parameters), the pace of improvement is very slow after a certain time, and, on the other hand, a solution of quite good quality is produced within this time. This means that it is enough to run the algorithm until the envelope curve starts to saturate.

The convergence of the algorithm was measured with test runs generating meal plans. Constraints were energy (min = 4190 kJ, opt = 4200 kJ, max = 4210 kJ) and protein (29 g, 29.27 g, 29.5 g), population size was 200, and probability of crossover and mutation was 0.9 and 0.2. Results show that with two constraints, a satisfactory solution was found in 6.2 iterations on average (14 iterations in the worst case). As Figure 4 shows, there is hardly any improvement in the quality of the solution after 250 iterations, so a nearly optimal plan can be found in this time.

4.3 Reaction to the Gradual Diminution of Constraints

We tested the reaction of the algorithm to the gradual diminution of constraints. Minimal and maximal values were two times the size of the suggested at the start and were gradually decreased to be virtually equal from the aspect of human nutrition. More than 150,000 tests were run. The tests showed that our method is capable of generating nutritional plans, even where the minimal and maximal allowed values of one or two constraints are virtually equal and the algorithm finds a nearly optimal solution when there are three or four constraints of this kind. According to our nutritionist, there is no need for constraints with virtually equal minimal and maximal values, and in most pathological cases the strict regulation of four parameters is sufficient. Our method has been proven capable of generating menus with meal plans that satisfy all constraints for non-pathological nutrition.

4.4 Results of the Multi-level Method

Generated plans satisfy numerical constraints on the level of meals, daily plans and weekly plans. The multi-level generation was tested with random and real world data. The tests showed that for a mainstream desktop personal computer it takes between ten and fifteen minutes to generate a weekly menu plan with a randomly initialized population. The weekly menus satisfied numerical constraints on the level of meals, daily plans and weekly plans. Our tests showed that the rule-based classification method successfully omits components that don't go well together.

The case-based initialization of the start-up population increases the speed of the generation process. Whenever a solution is needed for a plan with constraints that has been made previously it would be enough to use the solution that can be found in the case-base for these constraints. However, with some iteration, the algorithm may find better solutions than are in its initial population at startup. If there was no improvement in the best solution stored in the database for a particular plan, it can be assumed that one of the best solutions was found for that menu plan.

4.5 Variety

Variety is a key factor when considering dietary plans. GAs use random choice for guiding the evolution process for near-optimum search, so if the search space is large enough, the solution found should be close to the optimum, but need not be similar in several consecutive runs. However, GAs are known for finding near-optimal solutions, so if there are strict numerical constraints, then it can easily happen that only a small subset of solutions satisfies them (which are close enough to the optimum), and the probability that the solutions don't have similar attributes is marginal. So, a method for adjusting the expected occurrence of the alleles of the GA is needed for providing sufficient variety in the menu plans.

We measured the variety of menu plans with constraints for regular dietary plans for women aged between 19 and 31 with mental occupation. The variety of the allele that represents one of the 150 possible soups for a regular lunch is shown in Figure 5. The figure shows the occurrence (ordered by frequency) of each of the 120 soups that were present more than 15 times (0.1%) in the best solutions in 15,000 test runs. The most frequent soup in the best solutions of 15,000 runs was present 482 times (~3.2%), the second 426 times (~2.8%) and the 50th 102 times (~0.7%). Figure 5. also shows (lower part) the goodness of the best solution to which the corresponding alleles belong. The goodness of an allele is computed by summing its best fitness (i.e. the best fitness value of all solutions the allele was part of) with the weighted best fitness values of its eight neighbors. The goodness of the ith allele is defined as: $g[i] = f[i] + \sum_{j=1}^{4} [(1 - 0.2\,j) \cdot (f[i-j] + f[i-j])]$, when f[i] is the fitness of the ith allele. The trend curve (lower part) shows that solutions containing frequent alleles generally have better goodness. The results show that the algorithm uses alleles appearing in good solutions more often and the frequency of usage is approximately inversely proportional to the fitness of the best solution generated by using the particular allele.

However, it may happen that a run of the algorithm with properly configured nutritional constraints and rules results in a dietary plan that contains several occurrences of same dishes or dishes made from the same ingredients. Therefore, we allow more general rules, like $r_s = \langle ?, ?, 0.5 \rangle$ to be recorded in our rule-base, which also get pre-processed during the initialization of the algorithm. Rule r_s will penalize every solution that has the same value (solution) represented by its attributes more than one time. So, if r_s is imposed on a daily menu plan which contains orange drink for breakfast and lunch as well, then the fitness of this daily plan will be reduced by 50%.

We measured the effect of the rules on the variety and mean occurrence of the alleles (drinks) considering the solutions for the meal plan (lunch). The results of the statistical analysis are shown in Table 3. Two rules (r_A, r_B) where imposed on two alleles, respectively. The strictness of the rules was decreased from 100% to 75%, 50% and finally to 25%, giving a total of 16 configurations. Rule r_A penalized the solutions that contained drink "A" while r_B penalized solutions with drink "B". The relative occurrences of "A" are shown in the function of the strictness of the rules in Figure 6.

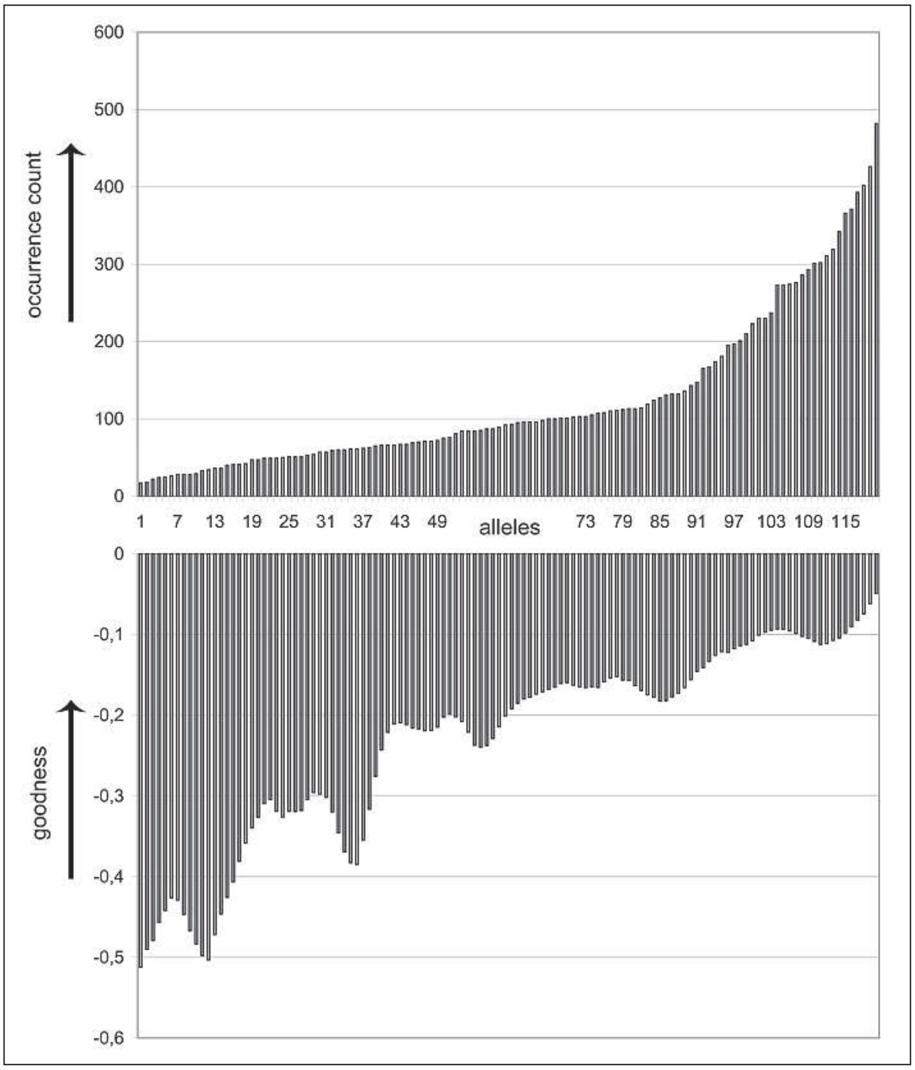

Fig. 5 The occurrences (ordered by relative frequency, upper figure) of 120 of the 150 possible alleles (soups) in the best solutions (lunches) of 15,000 runs. The lower figure shows the goodness of the best solution which the respective soup was part of.

Table 3 Statistical analysis of the distribution of the potential alleles (drinks) in the best solutions (lunches) and the mean occurrence of the alleles (A, B) on which the rules were imposed.

		A													
		100%				75%				50%				25%	
B	100%	A	100%	KS	3.1822e-004	A	69,50%	KS	5.3952e-005	A	31,42%	KS	3.1822e-004	A	7,80%
				T	7.4482e-006			T	4.5663e-007			T	9.4052e-005		
		B	100%	Sr	0.0020	B	139,71%	Sr	0.0020	B	175,00%	Sr	0.0020	B	186,76%
		Sr	0.0020			Sr	0.0020			Sr	0.0020			Sr	0.0020
	75%	A	103,44%	KS	2.1326e-007	A	69,50%	KS	9.4868e-007	A	38,53%	KS	0.0031	A	10,09%
				T	6.4855e-006			T	9.9226e-005			T	3.8001e-006		
		B	32,35%	Sr	0.0020	B	32,35%	Sr	0.0020	B	33,82%	Sr	0.0020	B	41,18%
		Sr	-			Sr	0.0078			Sr	-			Sr	0.0059
	50%	A	96,79%	KS	3.1822e-004	A	71,33%	KS	7.5721e-008	A	35,78%	KS	5.8152e-007	A	9,63%
				T	2.1538e-004			T	2.0654e-007			T	4.8058e-005		
		B	17,65%	Sr	0.0039	B	13,24%	Sr	0.0020	B	17,65%	Sr	0.0020	B	14,71%
		Sr	0.0156			Sr	-			Sr	0.0313			Sr	0.0156
	25%	A	113,76%	KS	2.7488e-007	A	80,73%	KS	1.2116e-005	A	35,55%	KS	1.2116e-005	A	8,26%
				T	3.4505e-004			T	- (*)			T	- (*)		
		B	1,47%	Sr	0.0020	B	5,88%	Sr	0.0020	B	5,88%	Sr	0.0020	B	4,41%

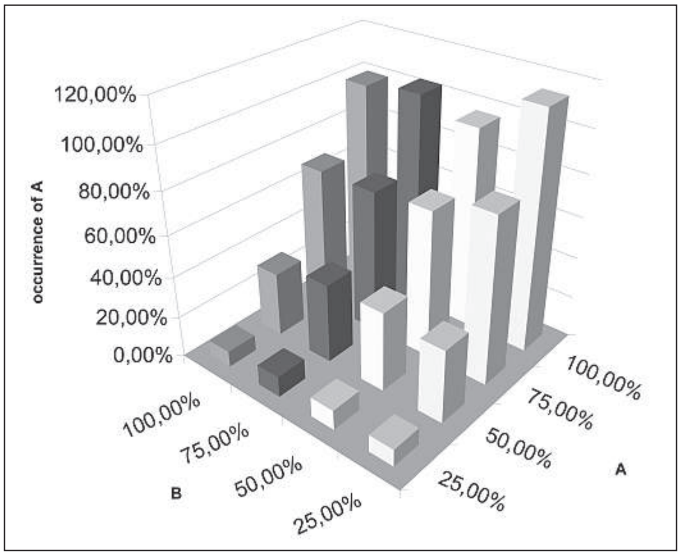

Fig. 6 The relative occurrence of a particular solution (A) in function of the strictness of two rules (penalizing solutions A and B).

We employed a two-sample Kolmogorov-Smirnov goodness-of-fit hypothesis test with the significance level of 5% to the two random samples created by recording the alleles representing drinks in neighboring configurations, running 1000 times each (using ten random populations, running each one 100 times). The table lists those P values (denoted with KS) for which the Kolmogorov-Smirnov test has shown significant difference in the distribution of the two independent samples.

We observed significant differences for all of the test pairs with respect to increasing the strictness of r_A (rows in Table 3). However, the same was not true for all of the pairs with respect to increasing the strictness of r_B (columns in Table 3.), so P values are not listed for such pairs. The explanation of this phenomenon is hidden in the differences between the occurrences of alleles A and B without penalties, which are 436 (43.6%) for "A" and 68 (6.8%) for "B", out of the 1000 possible. Since the number of instances of "A" is comparable to the possible instances, rule r_A not only changes the mean occurrence of "A", but significantly changes the distribution of the alleles. In case of penalizing the meals with drink "A" by 75%, the occurrence count of "A" decreases by 133 (~30%) from 436 (100%) to 303 (~70%). As Figure 7 shows, the 103 occurrences are shared somewhat proportional among the other alleles ("A" is 15th, "B" is the 12th allele in Fig. 7).

We performed a single sample Lilliefors hypothesis test of composite normality on the samples with an element size of 10 on 100 runs of the algorithm with 10 different starting populations and counting the occurrences of "A" and "B". The distribution of the occurrences of "A" in function of the starting populations proved normal, except for one case. Again, due to the few occurrences of "B" in the test runs, we could not determine its distribution. We paired the samples of neighboring configurations and if both had normal distribution with a significance level of 5%, we employed paired t-tests to check, if there is a significant difference in the mean occurrences of alleles "A" and "B". The results of the paired t-tests are shown in Table 3, denoted with T. In case of significant differences, the corresponding P values are also listed. If one of the samples did not have a normal distribution we marked the case with an asterisk (*).

Since the sample distribution was not known for more than half of the samples, we employed the Wilcoxon signed rank test of equality of medians with the significance level of 5% on each sample pair to measure whether there is significant difference between the mean occurrences. Results are shown in Table 3 with the corresponding P values, and are denoted with *Sr*. There were only three situations where there was no significant difference between the mean occurrences. These cases are marked with a hyphen (-).

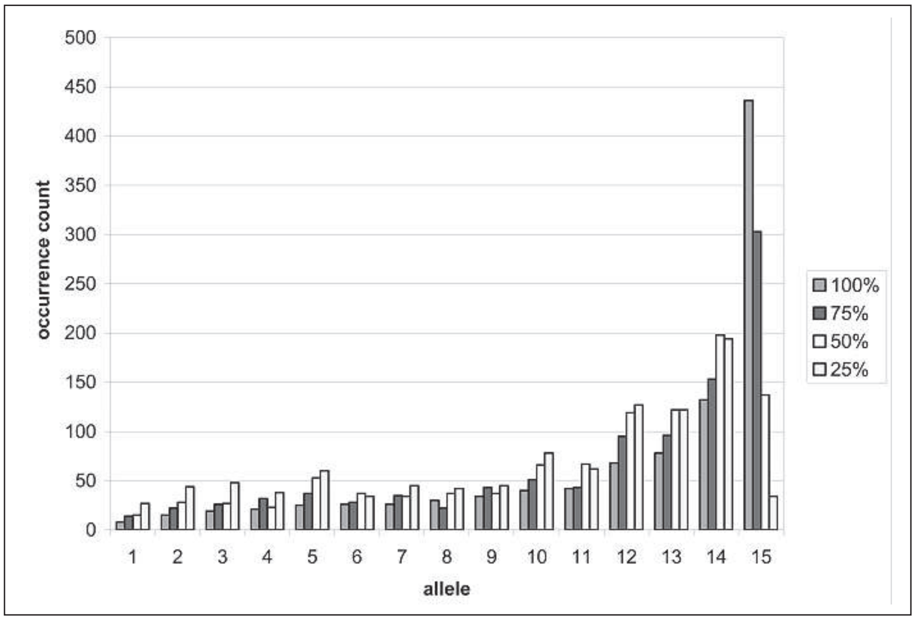

Fig. 7 Occurrence counts of the 15 possible alleles (drinks) in a solution (for lunch) in 1000 runs in function of the strictness of the rule imposed on the 15th allele

5. Discussion

The single level tests showed that GAs are capable of generating menu plans that satisfy strict nutritional constraints. The tests performed with the multi-level implementation show that constraint satisfaction remains the same on every level. From the aspect of nutritional constraints, our method outperforms present-day nutrition planning systems. Recently developed nutrition planning systems such as CAMPER [6] maintain the constraints only on a daily basis, in contrast, Menugene satisfies constraints on the meal-by-meal, daily and weekly levels.

The high optimal mutation rate can be surprising at first. However, the optimal mutation probability should be high because a population with 100 or 200 individuals can't contain much genetic flavor in the case of the menu generation problem. Mutation is the only operator which can introduce new genetic information in the population. Until solutions with proper genetic parameters appear in the population, the relatively high mutation probability is needed.

It is not necessary to generate the parameters of a nutritional plan simultaneously. If one or more parameters have been previously set, the unassigned ones can be generated in a way that the whole plan satisfies the relevant constraints. A practical application of this feature is when one eats at his/her workplace and can't choose his/her lunch for the weekdays. In this case, the lunches of the weekly plan can be defined by the end user at the beginning of the week and MenuGene can develop the whole weekly menu plan without changing these lunches. Parameters of the plan can be changed at any time; the algorithm is capable of adjusting the weekly plan to compensate for deviations. From a quantitative point of view, it is more important to keep the nutritional constraints on a weekly basis than to keep them on a daily basis or in a meal. For this reason, MenuGene allows relatively more deviation from the optimal values on the lower levels (day/meal), and tries to keep the strict constraints on a weekly basis.

The advantage of our approach is that it uses the same algorithm on every level, thus the hierarchical structure is easily expandable. The method is capable of controlling the nutrition on longer periods. Monthly optimizations could be performed without the need to plan the whole monthly plan in one run. After the first week, the plan for the second week can be made with the previous weekly plan in mind.

Harmony is more important on lower levels. For example, a meal or a daily plan with two dishes or meals with tomato is not well assorted. These plans can successfully be omitted with simple rules. We developed the data structure of MenuGene in a way that it can distinguish rules loaded by experts and users. In this way, users can define their personalized rules. These rules are only used for the user that defined them and have lower priority than the rules given by experts. As MenuGene uses the rules as parameters, the rule-base of the system can be developed while the system is being used. Incremental development of the rule-base is similar to that implemented in MIKAS [8]. The increasing number of rules doesn't have an impact on the generation time of a plan because the rule-base of the system is preprocessed.

6. Conclusions

The paper described the results of the automatic, parameterized menu planner *MenuGene* that uses a multi-level multi-objective genetic algorithm for a near-optimum search. A genetic algorithm-based method for weekly dietary plan design was presented with a fitness function that classifies menus according to the amount of nutrients and harmony. An abstract scheme was proposed for the consistent handling of the different level problems, for the implementation of crossover and mutation and for the coding of chromosomes as well as a fitness function. Algorithmic tests revealed that a relatively high mutation rate is desirable and that after a certain time, the quality of the solution does not improve much. Our tests showed that GAs generally produce high variety, at least in non-overparameterized configurations, but in any case, rules can be used to employ the desired level of variety and harmony.

Future work on *MenuGene* includes the development of parallel computation methods for improving the runtime of the coevolution process and the continuous improvement of *MenuGene's* case-base and rule-base with a web-based application that was developed for human experts. The system can be tested at http://menugene.irt.vein.hu.

Acknowledgments
The work presented was supported by National Research and Development Program NKFP 2/052/2001 and the Hungarian Ministry of Health.

References

1. The interactive menu planner of the National Heart, Lung, and Blood Institute at http://hin.nhlbi.nih.gov/menuplanner/ [Verified April 11, 2005]
2. The Cordelia Dietary and Lifestyle counseling project at http://cordelia.vein.hu/ [Verified April 11, 2005]
3. Balintfy JL.Menu Planning by Computer. Communications of the ACM, vol. 7, no. 4, pp 255-9. April, 1964
4. Eckstein EF. Menu planning by computer: the random approach. J Am Diet Assoc 1967; 51 (6): 529-33.
5. Hinrichs RR. Problem Solving in Open Worlds: A Case Study in Design. Northvale, NJ: Erlbaum; 1992.
6. Marling CR, Petot GJ, Sterling LS. Integrating Case-Based and Rule-Based Reasoning to Meet Multiple Design Constraints. Computational Intelligenc 1999; 15: 308-12.
7. GJ Petot , CR Marling, Sterling L. An artificial intelligence system for computer-assisted menu planning. Journal of the American Dietetic Association1998; 98: 1009-14.
8. Kovacic KJ. Using common-sense knowledge for computer menu planning [PhD dissertation]. Cleveland, Ohio: Case Western Reserve University; 1995.
9. Khan AS, Hoffmann A. An advanced artificial intelligence tool for menu design. Nutr Health 2003; 17 (1): 43-53.
10. Khan AS, Hoffmann A. Building a case-based diet recommendation system without a knowledge engineer.Artif Intell Med. 2003;27 (2): 155-79.
11. Noah S, Abdullah S, Shahar S, Abdul-Hamid H, Khairudin N, Yusoff M, Ghazali R, Mohd-Yusoff N, Shafii N, Abdul-Manaf Z. DietPal: A Web-Based Dietary Menu-Generating and Management System. Journal of Medical Internet Research 2004; 6 (1): e4.
12. Dollahite J, Franklin D, McNew R. Problems encountered in meeting the Recommended Dietary Allowances for menus designed according to the Dietary Guidelines for Americans. J Am Diet Assoc 1995; 95 (3): 341-4, 347; quiz 345-6.
13. Food and Nutrition Board (FNB), Institute of Medicine (IOM): Dietary Reference Intakes: Applications in Dietary Planning. Washington, DC: National Academy Press; 2003

14. Food and Nutrition Board (FNB), Institute of Medicine (IOM): Dietary Reference Intakes for Energy, Carbohydrate, Fiber, Fat, Fatty Acids, Cholesterol, Protein, and Amino Acids (Macronutrients). Washington,DC: National Academy Press; 2002
15. Bucolo M, Fortuna L, Frasca M, La Rosa M, Virzi MC, Shannahoff-Khalsa D. A nonlinear circuit architecture for magnetoencephalographic signal analysis.Methods Inf Med. 2004; 43 (1): 89-93.
16. Laurikkala J, Juhola M, Lammi S, Viikki K. Comparison of genetic algorithms and other classification methods in the diagnosis of female urinary incontinence. Methods Inf Med 1999; 38 (2): 125-31.
17. Pena-Reyes CA, Sipper M. Evolutionary computation in medicine: an overview. Artificial Intelligence in Medicine 2000; 19: 1-23.
18. Heckerling PS, Gerber BS, Tape TG, Wigton RS. Selection of Predictor Variables for Pneumonia Using Neural Networks and Genetic Algorithms. Methods Inf Med 2005; 44: 89-97.
19. Coello Coello CA.A comprehensive survey of evolutionary-based multiobjective optimization techniques, Int J Knowledge Inform Syst 1999; 1: 269-309.
20. Multi-level Multi-objective Genetic Algorithm Using Entropy to Preserve Diversity. EMO 2003, LNCS 2632, 2003. pp 148-61.
21. The M.I.T. GALib C++ Library of Genetic Algorithm Components at http://lancet.mit.edu/ga/ [verified April 11, 2005]

Correspondence to:
Balázs Gaál
Department of Information Systems
University of Veszprém
Egyetem u. 10
8201 Veszprém
Hungary
E-mail: bgaal@irt.vein.hu