Genetic algorithm for finding a good first integer solution for MILP

Kimmo Nieminen
Sampo Ruuth
Systems Analysis Laboratory
Helsinki University of Technology

István Maros
Department of Computing
Imperial College, London

Department of Computing, Imperial College
Departmental Technical Report 2003/4
ISSN 1469–4174

April 2003

Abstract

The paper proposes a genetic algorithm based method for finding a good first integer solution to mixed integer programming problems (MILP). The objective value corresponding to this solution can be used to efficiently prune the search tree in branch and bound type algorithms for MILP. Some preliminary computational results are also presented which support the view that this approach deserves some attention.

1 Introduction

A key to the success of Branch and Bound (B&B) algorithms for integer programming is the early identification of a first integer solution [4]. While there are several methods to achieve this goal, in this paper we elaborate on the idea proposed by Nieminen [5] and present a customized genetic algorithm (GA) [3] for this purpose. The origins of this work were presented by the authors at the 20th IFIP Conference on System Modelling and Optimization [2].

2 About genetic algorithms

A genetic algorithm is a heuristic search algorithm for the solution of optimization problems in which, starting from a random initial guess solution, better descendants are tried in an attempt to find one that is the best under some criteria and conditions. It is based on the idea of evolution theory that individuals having a high value of quality will survive to the next generation with greater probability.

In our interpretation of GA *genes* are sets of real numbers or bits. Individuals or *genomes* are vectors of genes. A set of genomes is a *population.* The *value* of a genome is a real valued function defined over the population. The following operations are defined for a population:

- *selection* operation selects the best genomes from the modified population for the next *generation.*
- *crossover* operation exchanges a number of genes of two genomes of a population.
- *mutation* operation changes two randomly selected genes of a genome.

Let $K =$ number of generations and $P_k = \{\mathbf{x}^i : \mathbf{x}^i \in \mathbb{R}^n,\ i = 1, \ldots, I_k\}$ the set of genomes in the k-th generation, where I_k denotes the size for $k = 1, \ldots, K$. A pseudo code for the GA can now be written as follows:

Algorithm 'Genetic Optimizer'
Create an initial population $P_1 = \{\mathbf{x}^i \in \mathbb{R}^n,\ i = 1, \ldots, I_1\}$.
For $k = 1, 2, \ldots, K$
  If a good enough solution $\mathbf{x} \in P_k$ is found then
    $\mathbf{x}^* = \mathbf{x}$, exit
  else
    Perform crossover operation for a number of genomes of population P_k. This leads to an enlargement of P_k.
    Perform mutation operation on the genomes that were created during crossover.
    Select a set of best genomes from population P_k to population P_{k+1} for the next generation.
  end If
end For.

Determining the population sizes $I_1, \ldots, I_k$ is left open in this general description.

3 Application to Branch and Bound algorithm

In this section we propose a method to use GA for finding a good first feasible integer solution to a MILP (Mixed Integer Linear Program).

$$\begin{aligned} \min \quad & \mathbf{c^T x}, && (1)\\ \text{subject to} \quad & \mathbf{Ax} = \mathbf{b}, && (2)\\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}, && (3)\\ & x_j \ \text{ integer}, \ \ j = 1, \ldots, p \leq n, && (4) \end{aligned}$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$ and all vectors are of appropriate dimension.

In our case a gene is a feasible integer value of an integer variable satisfying (3), a genome (individual) is a feasible vector that satisfies (3) and (4). Note, such a vector has $n - p > 0$ non-integer components if $p < n$.

The idea is to design and use a genetic algorithm that finds a (hopefully) good integer feasible solution to the problem. The solution does not have to be basic as only the value of the objective function will be used.

If the GA terminates with an integer feasible solution the corresponding objective value is taken as the *incumbent solution* at the beginning of the Branch and Bound (B&B) procedure. If it is good enough large branches of the tree can be eliminated from the search, thus contributing to the overall efficiency of the B&B. We propose the following algorithm, which is also shown in Figure 1, to achieve this goal. The different procedures in the steps are described in subsequent sections.

Algorithm 'GA for MILP'

1. Solve the relaxed LP problem. Set $k = 1, \ S = \emptyset$. Form an initial population P_1.
2. Select a remaining genome from population P_k. Fix genes and calculate the value of the selected genome.
3. If the solution is feasible satisfying (2) and (3) goto 4 else goto 5.
4. Save the solution in S if it is good enough.
5. If there are genomes left in the population goto 2 else goto 6.
6. If $k = K$ exit to B&B else goto 7.
7. Add the set of saved feasible solutions S to population P_k; $P_k := P_k \cup S$.
8. $k := k + 1$. Determine the next generation P_k using crossover, mutation and selection operations, goto 2.

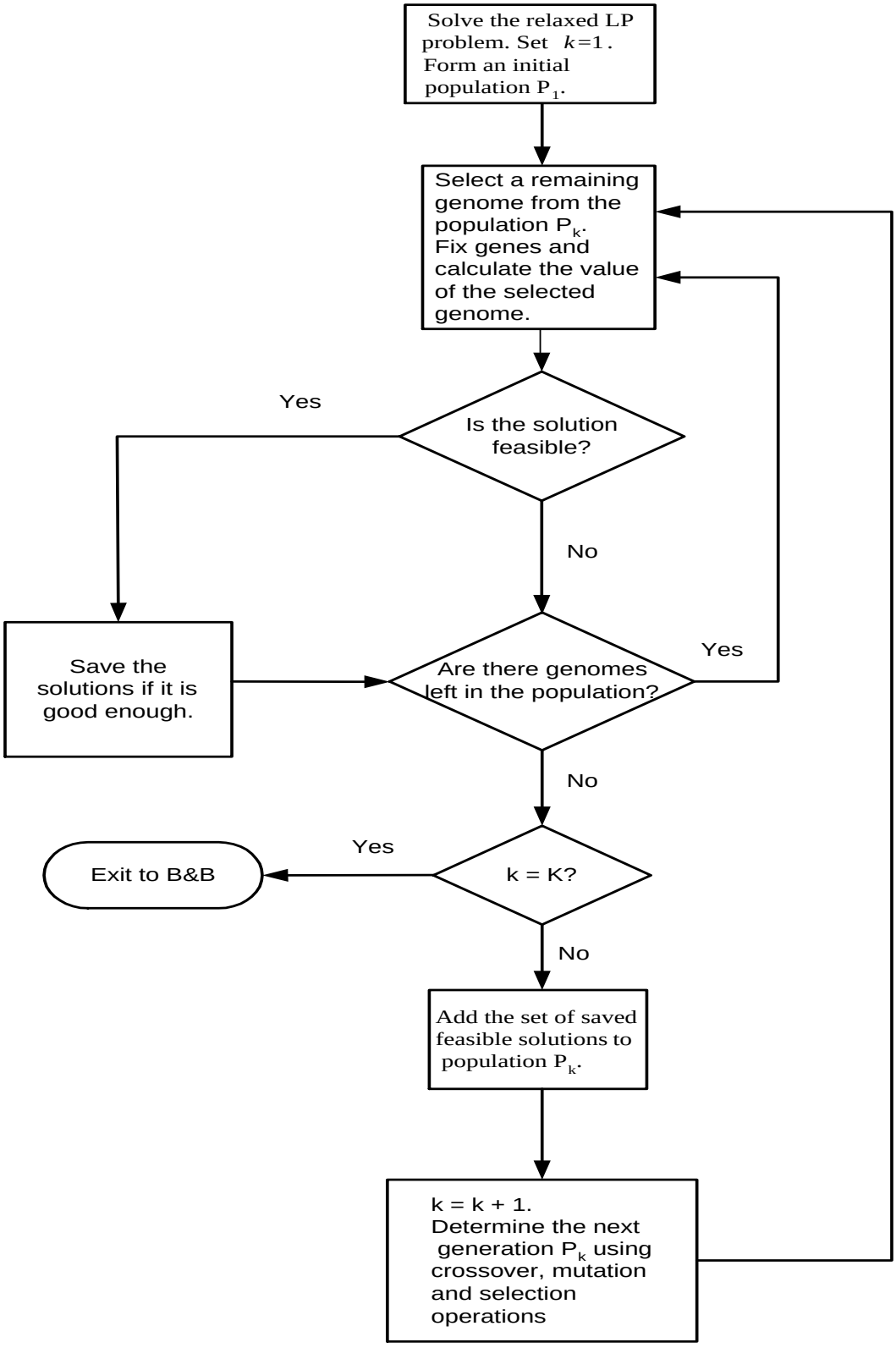

Figure 1: Block diagram for finding a good feasible integer solution by GA.

3.1 Generating the initial population

Choosing random integer vectors from the feasible interval $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ generates the initial population. If the feasible interval is unbounded for a variable it is replaced by a bounded interval which includes the relaxed LP solution value of that variable. An example is given in Figure 2. There are many other, and certainly more sophisticated, ways to generate the initial population.

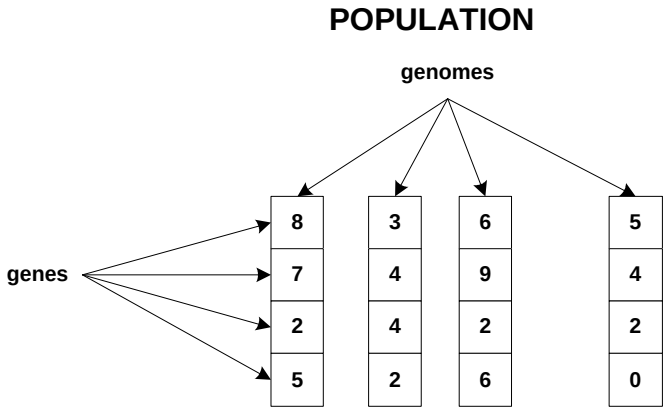

Figure 2: Generating the initial population; an example with $0 \leq x_j \leq 10,\ j = 1, \ldots, 4$.

3.2 Calculating the value of a genome

Throughout the paper by relaxed problem we mean the continuous LP where non-fixed integer variables are treated as continuous. Fixing is explained below. The relaxed continuous problem with (2) and (3) is solved by the simplex method.

Given a genome, its integer variables are not enforced straight away but fixed incrementally one after the other at the values determined by the GA. The order of fixings is determined randomly. If, as the result of more genes being fixed, the solution becomes infeasible then an attempt is made to regain feasibility which may or may not be successful. The steps of the algorithm are given below and also shown in Figure 3.

Algorithm 'Value of a Genome'

Assumption: The LP relaxation of the original problem is solved and the solution is saved.

1. Select randomly the fixing order of genes of the i-th genome.
   Set $r := 1$.
2. Fix the r-th gene. Leave the unfixed genes relaxed. Solve this relaxed problem by the dual simplex method.
3. If the solution is feasible satisfying (2) and (3) goto 4 else goto 5.
4. If $r < p$ (there are still unfixed genes in the genome) then $r := r + 1$, goto 2 else goto 7.
5. Take the last fixed gene (that caused infeasibility) and assign to it the value (rounded to the nearest integer) it had in the previous (still feasible) solution. Leave the remaining variables unfixed. Solve the problem by the dual simplex. If the solution is feasible satisfying (2) and (3) goto 4 else goto 6.
6. Fix all the remaining genes at their nearest integer value obtained from the last solution. Solve the problem with the dual simplex algorithm.
7. Calculating the value of the i-th genome is carried out in the following way: If the solution is infeasible the value of the genome is defined as the sum of the infeasibilities. If the solution is feasible (the sum of the infeasibilities is equal to zero) the value of the genome is defined to be the optimal value of the objective function. Therefore, the value of a genome is represented by a pair (q, v), where q is a $0/1$ indicator variable (0 = feasible, 1 = infeasible) and v is the value of the objective function (if $q = 0$) or the sum of infeasibilities (if $q = 1$).

Remark: If $p = n$ (all variable are integer) Step 6 is just a substitution of the solution into $\mathbf{Ax}$.

To illustrate the operation of algorithm 'Value of a Genome' let us consider the following example. Suppose there are four integer variables as in Figure 2 and we are calculating the value of the first genome (8, 7, 2, 5). Assume in the solution of the LP relaxation of the original problem the corresponding variables take the following values: (3.5, 2.2, 5.3, 1.9). Let the fixing order be 1 2 3 4.

Fix the first ($r = 1$) gene to 8. Assume the solution of the relaxed problem, in which the first integer variable is fixed to 8 and the remaining integer variables are still relaxed, is *feasible*: (8, 3.2, 2.3, 3.9). Next, fix the second ($r = 2$) gene to 7. Let the *feasible solution* of the relaxed problem, in which the second integer variable is fixed to 7 and the remaining integer variables are still relaxed, be (8, 7, 4.8, 2.9). Then fix the third gene to 2. Now suppose that the corresponding solution is *infeasible*. According to Step 5 of the algorithm the third integer variable will be fixed to value 5 (4.8 rounded to 5) instead of 2 (which would be the preassigned value from the genome). The problem in which the first three genes are fixed at (8, 7, 5) and the fourth is relaxed is solved.

Now, there are two possibilities.

1. If the solution is *infeasible* then, by Step 6, fix also the remaining integer variables (in this case there is only one variable unfixed, namely variable 4) to their rounded integer values (here the fourth variable = 3) and solve the problem in order to be able to calculate the value of the genome (sum of infeasibilities). Note, this genome is now (8, 7, 5, 3) which is different from the original (8, 7, 2, 5) but it most likely has a better infeasibility measure.

2. If the solution is *feasible* being, e.g., (8, 7, 5, 2.2) then continue as before, i.e., follow Steps 4 and 2: fix the fourth gene to 5 and solve the relaxed problem (in this case all the integer variables are fixed). Now suppose that this solution is infeasible. Again, according to Step 6 of the algorithm the fourth integer variable is fixed to value 2 (2.2 rounded to 2) instead of 5. If the solution of the relaxed problem (again all the integer variables are fixed) is infeasible then calculate the value of the genome (sum of infeasibilities). If the (8, 7, 5, 2) solution is feasible then calculate the value of the genome (the value of the objective function).

The calculation of the value of a genome may require the solution of several LP problems. However, these problems are very much related and the optimal basis of the previous problem is usually an excellent starting basis to the new one. As a result, only very few iterations are needed for reoptimization. On the other hand, the quality of the genome for the new generation is very likely much better. The usefulness of the above algorithm can be assessed by computational testing which we present in section 4.

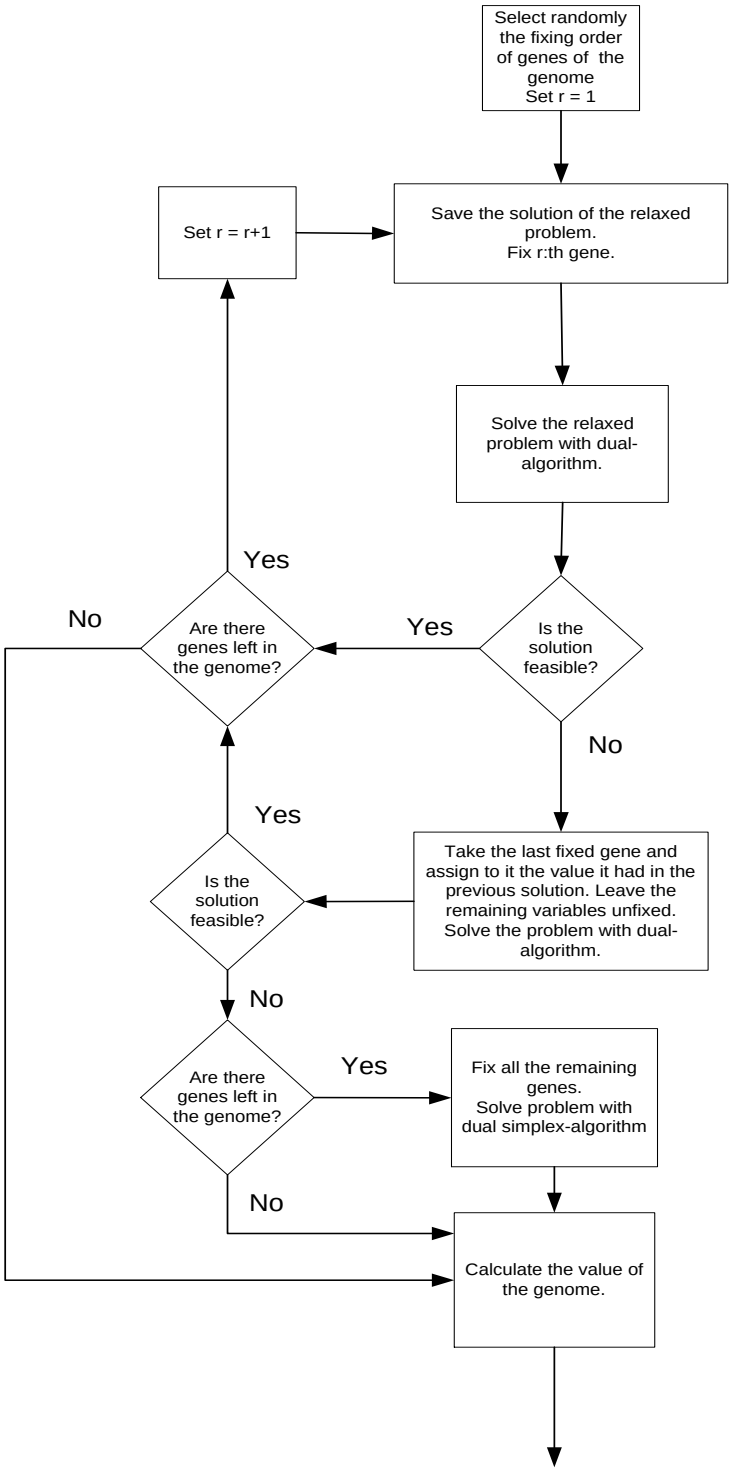

Figure 3: Fix and calculate the value of the selected genome.

3.3 Generating the new population

Select two genomes from the population of the current generation one which is the best genome (parent 1) and the other a randomly selected genome (parent 2). Make a crossover operation between the two genomes producing two new genomes (child 1 and child 2). Make a mutation operation to both of the children (Figure 4). Perform selection to obtain the new generation.

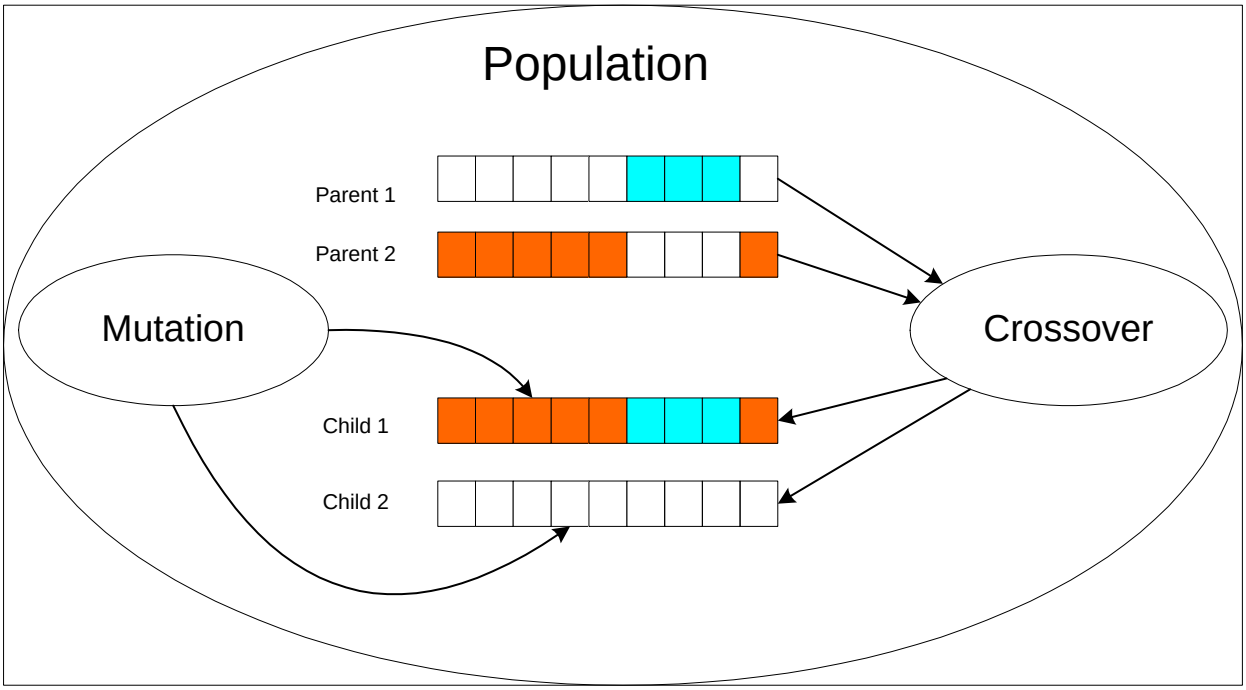

Figure 4: Generating the new population by crossover and mutation operations.

Crossover operation. Crossover operation generates two children from two parents. The children inherit genes randomly from the parents. Whether the crossover is made at all is determined by the parameter p_c (usually in the interval $[0.5, 1.0]$). If $p_c = 1.0$ crossover is always made. If $p_c < 1.0$ crossover is made with probability p_c. If the crossover operation is not made to the parents the genes are copied to the children unchanged. If crossover is made to the parents then in this algorithm the 'method of two points' is used in which two randomly selected genes the starting point (6 in Figure 4) and the ending point (8 in Figure 4) are determined for parent 2. After this the genes between these points are exchanged between the parents.

In Figure 5 we present a more detailed example of crossover. In this example only one gene is changed between the parents (the starting and ending points are the same = 2, the corresponding values of the changing genes are 7 and 4).

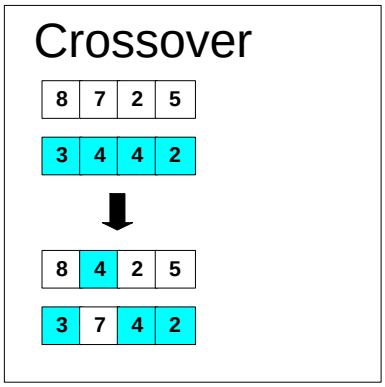

Figure 5: Crossover operation.

Mutation operation. When the children genomes have been created both of them undergo mutation operations. The number of the operations n_{op} is determined by the parameter p_m (usually in the interval $[0.01, 0.2]$) and is calculated as follows:

$$n_{op} = p \times p_m$$

where

$\lceil n_{op} \rceil$ = number of mutations,
p = size of the genome,
p_m = user parameter, defined above.

The mutation operations are performed by selecting n_{op} random genes of the genome and replacing the value of the selected gene by a random i integer from the feasible interval of the corresponding integer variable (Figure 6).

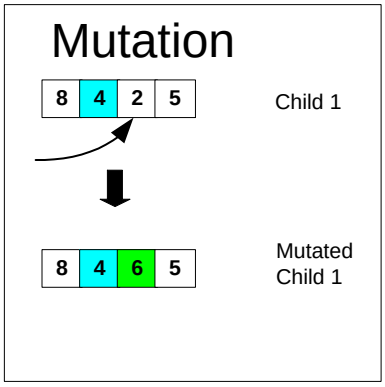

Figure 6: Mutation operation.

Selection operation Select a set of best (least infeasible) genomes from population P_k to form population P_{k+1} of next generation.

4 Experimental results

In this section we give a brief account of our first computational tests. We have compared the GA approach with the depth-first (DF) strategy to obtain a first integer solution to MILP problems. In the next subsection we briefly outline the DF we used.

4.1 Node selection with depth-first strategy

Suppose the optimum linear program at a node is defined as follows:

$$\begin{aligned} &\min \quad z = \bar{c}_0 + \sum_{j \in \mathcal{R}} \bar{c}_j x_j \\ &\text{subject to} \\ &x_i = x_i^* - \sum_{j \in \mathcal{R}} \alpha_{ij} x_j, \quad i \in \mathcal{B} \\ &x_i \geq 0, \quad i \in \mathcal{B} \\ &x_j \geq 0, \quad j \in \mathcal{R} \end{aligned}$$

where $\mathcal{B}$ and $\mathcal{R}$ denote the index set of basic and nonbasic variables, resp., $\bar{c}_j = c_j - z_j$ is the reduced cost of nonbasic variable x_j, $j \in \mathcal{R}$ with $z_j = c_{\mathcal{B}}^T B^{-1} a_j$, B is the current basis, a_j is the j-th column of matrix $\mathbf{A}$ and α_{ij} is a matrix element in current simplex tableau. The current optimum solution is thus given by

$$\begin{aligned} &z = \bar{c}_0 \\ &x_i = x_i^*, \quad i \in \mathcal{B} \\ &x_j = 0, \quad j \in \mathcal{R} \end{aligned}$$

Suppose x_k, $k \in \mathcal{B}$, is an integer variable whose value x_k^* is fractional. Hence

$$x_k^* = [x_k^*] + f_k, \quad 0 < f_k < 1$$

The branching rule imposes the following restrictions:

$$x_k \leq [x_k^*] \tag{5}$$

or

$$x_k \geq [x_k^*] + 1 \tag{6}$$

In the depth-first (DF) strategy we solve the problems in three different ways. In the first case, we always select the lower child node determined by (5). In the second, always the upper child node defined by (6) is chosen. In the third case a node is selected according to the smallest penalty (see below) described in [6]. Let P_u and P_d be the up and down penalties that are used to estimate the upper bound when x_k is "upped" to at least $[x_k^*] + 1$ or "downed" to at least $[x_k^*]$. The penalties are calculated as follows:

$$\begin{aligned} P_u &= \min_{j \in \mathcal{J}^+} \{\bar{c}_j f_k / \alpha_{kj}\} \\ P_d &= \min_{j \in \mathcal{J}^-} \{\bar{c}_j (f_k - 1) / \alpha_{kj}\} \end{aligned}$$

	Number of					
Problem	Rows	Cols	Int vars	0/1 vars	Cont vars	Best Solution
bbb150	600	899	300	300	599	LP:-32277.70
bell3a	123	133	71	39	62	878430.32
bell5	91	104	58	30	46	8966406.49
dcmulti	290	548	75	75	473	188182
fiber	363	1298	1254	1254	44	405935.18
flugp	18	18	11	0	7	1201500
gt2	29	188	188	24	0	21166
gen	780	870	150	144	720	112313
l152lav	97	1989	1989	1989	0	4722
markshare1	69	62	50	50	12	0.0
misc06	820	1808	112	112	1696	12850.86
p0282	241	282	282	282	0	258411
pk1	45	86	55	55	31	11
rout	291	556	315	300	241	1077.56
set1ch	492	712	240	240	472	54537.75
vpm1	234	378	168	168	210	20

Table 1: Problem characteristics of the first test set. In all cases, the objective is minimized. 'Best Solution' is the known best integer solution.

where $\mathcal{J}^+(\mathcal{J}^-) = \{j \in \mathcal{R} \mid \alpha_{kj} > 0(< 0)\}$. In the forthcoming tables we included the best result obtained by the three choices for each problem.

In the experiments we have used two sets of problems outlined below.

The first set of test problems was chosen from the standard MIPLIB3 library. The statistics of the selected problems are given in Table 1. Most of these models are binary or almost binary mixed integer problems. Table 2 shows the times in seconds (on a Pentium 400MHz PC) of finding integer solutions using DF and GA strategies.

	First		Second		Third		GA	
Problem	int sol	int time	int sol	int time	int sol	int time	int sol	int time
bbb150	-32214.95	13.00	-32216.34	22.60	-32217.50	464.00	-32218.23	13.60
bell3a	infinite	>35 min					948806	18.38
bell5	9107149	0.36		> 180.00			9073983	1.60
dcmulti	194878.81	0.23	189994.42	0.27	188361.80	54.80	188794.20	39.90
fiber	2817425.55	0.38	1988836.00	58.20	1749936.30	143.00	778389.00	3.80
flugp	1201500	0.98					1201500	1.13
gt2	154664	0.03	140642	0.08	73847	220.00	69183	48.55
gen	112412	2.70					114219	14.37
l152lav	4746.00	0.74	4737.00	1.44	4735.00	3.42	4848.50	6.41
markshare1	355.00	0.02	142.00	0.09	30.00	24.90	29.00	13.60
misc06	12864.51	0.36	12850.86	23.90			12870.32	1.64
p02822	392115.00	0.02	385491.00	1.50		> 300.00	335587.30	10.70
pk1	53	0.08	48	0.13	42	0.24	38	0.17
rout	2194.43	4.53	2105.26	93.65	1862.00	360.00	1559.00	20.50
set1ch	88622.75	0.95	87030.25	1.70	86489.50	256.00	79761.25	6.60
vpm1	24	0.03	23	0.36	22	1.20	24	0.23

Table 2: Times in seconds. Columns 'First', 'Second' and 'Third' refer to the B&B runs with depth-first strategy, 'GA' columns contain results obtained by the genetic algorithm.The use of > character means that a better integer solutions than the previous one has not been found within the time given after the character.

Problem	Number of Rows	Cols	Int vars	0/1 vars	Cont vars	Best Solution
Afiroi	27	32	13	0	19	-417.64
AGGi	488	163	81	0	82	-35991578.65
Boeing1i	351	384	180	0	204	LP: -335.21
FIT1Pi	627	1667	45	0	1622	9155.74
Gangesi	1309	1681	64	0	1617	-109573.72
GFRD-PNCi	616	1092	66	0	1026	6902242.27
Grow22i	946	440	41	0	399	LP: -160834336.00

Table 3: Problem characteristics of the second test set. In all cases, the objective is minimized. 'Best solution' is the MILP optimal solution unless the value is preceded by LP, in which case it is the LP relaxation.

In one case (`bell3a`), B&B with DF was unable to find an integer solution in the given amount of time. This fact is noted by the infinite value of the integer solution. GA was able to find a solution that satisfied the integrality constraints to the time limits.

For problem `fiber` the first integer solution with DF strategy was 2817425.55 obtained in 0.38 seconds. The second integer solution was 1988836.00 within 58.20 seconds. With the GA strategy we found the first integer solution of 778389 in 3.8 seconds. For problem `gen` DF algorithm is better than GA. In most of the cases, however, GA seems to give better solutions in shorter time especially for harder problems. Our implementation is rather experimental, we have used an unsophisticated B&B-algorithm. As a consequence, it is rather slow and we could not solve all the problems in MIPLIB3.

Models of the second set of test problems are from `netlib/lp/data` [1]. To make them MILP problems a subset of the variables have been changed to integer variables in each selected problem. The problems statistics are shown in Table 3.

Looking at the results in Table 4 we can say that GA is roughly more than twice as fast as DF for most of the models.

5 Conclusion, further research

The results of the first experiments show that an average of 50% reduction in solution time can be achieved with the proposed method compared to using the depth-first strategy for finding the first integer feasible solution. As the results are encouraging enough, we find it justified to further analyze and refine this approach.

First, the idea of an adaptive choice of the the initial population needs to be addressed. Obviously, it has a huge impact on the quality of the solution and also on the speed of the operation of GA. Making it problem dependent rather than completely random promises substantial benefits.

Next, we plan to work around the selection, crossover and mutation rules to make

Problem	Depth-first first int sol	time	Genetic first int sol	time
Afiroi	-417.64	0.03	-264.95	0.30
AGGi	infinite	>600	-35678910.00	6.00
Boeing1i	-288.00	2.20	-290.00	2.32
FIT1Pi	9305.00	75.50	9174.21	30.12
Gangesi	-109566.41	10.61	-109578.96	5.66
GFRD-PNCi	6902246.20	20.10	6902244.40	0.70
Grow22i	-160822696.40	136.00	-160834185.37	13.53

Table 4: Times in seconds to obtain the first integer solution.

them more tuned to the problems.

As the GA approach is a sort of heuristics no guarantees can be provided for its performance. However, we believe any new lead in the very important issue of finding a good first feasible integer solution is worth investigating.

References

[1] Gay, D.M., "Electronic mail distribution of linear programming test problems", *COAL Newsletter*, Mathematical Programming Society, Vol. 13, 1985, pp. 10–12.

[2] Maros, I., Nieminen, K, Ruuth, S., *Genetic algorithm for finding the first integer solution in Branch and Bound algorithms*, 20th IFIP Conference on System Modelling and Optimization, Trier, Germany, July 2001.

[3] Mitchell, M., *An Introduction to Genetic Algorithms*, Cambridge, MA: The MIT Press, 1996.

[4] Nemhauser G. L., Wolsey L. A., *Integer and Combinatorial Optimization*, John Wiley and Sons, Inc., New York, 1988.

[5] Nieminen, K., *Development of Mixed Integer Programming methods*, Master's Thesis, Systems Analysis Laboratory, Helsinki University of Technology, 2002.

[6] Taha, H., *Integer Programming*, Academic Press, Inc. , New York, 1975.