Solving Large-Scale Integer Linear Programs Using A Customized Genetic Algorithm

Kalyanmoy Deb[1] and Koushik Pal[2]

[1] Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur, PIN 208 016, INDIA, deb@iitk.ac.in

[2] Department of Mathematics and Scientific Computing, Indian Institute of Technology Kanpur, Kanpur - 208016, INDIA, kapal@iitk.ac.in

KanGAL Report Number 2004003

Abstract. Many optimal scheduling and resource allocation problems involve large number of integer variables and the resulting optimization problems become integer linear programs (ILPs) having linear inequality and equality constraints. The integer restrictions of variables in these problems cause tremendous difficulty for classical optimization methods to find the optimal or a near-optimal solution. The popular branch-and-bound method is an exponential algorithm and faces difficulties in handling ILP problems having thousands or tens of thousands of variables. In this paper, we extend a previously-suggested customized GA with four variations of a multi-parent concept and significantly better results are reported. We show variations in computational time and number of function evaluations for 100 to 100,000-variable ILP problems and in all problems near-linear complexity is observed. The exploitation of linearity in objective function and constraints through genetic crossover and mutation operators is the main reason for success in solving such large-scale applications. This study should encourage further use of customized implementations of EAs in similar other applications.

Keywords: Integer linear programs, customized GAs, Large-scale optimization, computational time

1 Introduction

Optimal scheduling and resource allocation problems often arise in different real-world activities and are routinely solved using classical search and optimization algorithms including linear programming methods. The difficulties often faced in solving such problems are (i) dimensionality of the search space and (ii) integer restriction of the decision variables. If the resulting problem is linear (that is, the objective function and constraints are all linear functions of the decision variables), the linear programming (LP) approaches are ideal candidates to solve such problems (Taha, 1989). Although the first difficulty is not a matter for solving such problems using an LP, the second difficulty requires an LP approach to be used with an integer programming approach, such as the *branch-and-bound* method. Since the branch-and-bound approach requires branching every non-integer variable into two different LPs, the presence of a large number of integer decision variables demands an exponentially large number of function evaluations to solve the problem to optimality.

For past few decades, such problems have also been solved using various non-traditional methods, such as simulated annealing, genetic algorithms, tabu search etc. (Kirkpatrick, Gelatt and Vecchi, 1983, Goldberg, 1989; Glover, 1997). In some of these methods, although the second difficulty of handling integer variables is not a matter, the first difficulty of handling a large number of decision variables is not well researched. A recent paper (Deb, Reddy and Singh, 2003) clearly demonstrated that a direct use of a genetic algorithm (binary-coded or real-coded) with generic crossover and mutation is too expensive to handle integer linear programs. The study also suggested a customized GA for handling such problems and problems having as large as *one million* variables were solved with a less-than-quadratic computational time complexity. That study was probably the first attempt to solve such a large-scale application using an evolutionary algorithm.

This paper is motivated from this earlier study and a multi-parent customized GA is suggested and tested on very large-scale ILP problems. Four different variations of the proposed multi-parent recombinative GA are compared with each other and with the previous study in terms of computational time complexity and in terms of required function evaluations. For each case, optimal parent and population sizes are found experimentally by solving a casting scheduling problem, which is an ILP and a representative problem to many other real-world optimization problems. The advantage of developing a customized GA for large-scale application and the systematic parametric approach followed in this study should encourage readers to pay attention to population-based customized optimization techniques for solving real-world optimization problems.

2 Casting Scheduling as an Integer Linear Program

In a typical foundry, casting of various sizes are made from a *heat* by melting metal in a large crucible. For convenience, more than one crucible are usually used. In this study, we shall assume a foundry using two crucible of different sizes (W_I and W_{II}), so that each crucible is used on alternate days. Depending on the crucible size used on the j-th day, the number of heats (H_j) allowed per day vary. We assume that the total number of castings (R) to be made is so large that a multi-day schedule requiring a total of H heats is necessary. Let us also assume that r_k copies of order k having a weight w_k kg are to be made, such that the total number of castings is $R = \sum_{k=1}^{K} r_k$ (where K is the total number of orders) and the total amount of metal required to make all castings is $M = \sum_{k=1}^{K} r_k w_k$ kg. With these parameters, one can optimize a casting sequencing for multiple days by introducing decision variables x_{ki} denoting the number of copies of order k made from the i-th heat. The equality constraints ensure that all desired copies of each order k are made from the combination of H heats. The inequality constraints arise to ensure that total amount of metal used for pouring in each heat is less than or equal to the size of the crucible (can be either W_I or W_{II} depending on the day). We form the following ILP:

$$\left.\begin{array}{lll} \text{Maximize } \frac{1}{H}\sum_{i=1}^{H} 100 \sum_{k=1}^{K} w_k x_{ki}/W_i, & & \\ \text{Subject to } \sum_{i=1}^{H} x_{ki} = r_k, & \text{for } k = 1, 2, \ldots, K, & \\ \quad \sum_{k=1}^{K} w_k x_{ki} \le W_i, & \text{for } i = 1, 2, \ldots, H, & \\ \quad x_{ki} \ge 0, & & \\ \quad x_{ki} \text{ is an integer.} & & \end{array}\right\} \quad (1)$$

A sequence of operation can be evaluated based on the utilization of the molten metal in each heat. Since $\sum_{k=1}^{K} w_k x_{ki}$ kg of molten metal is utilized during the i-th heat, the percentage utilization of molten metal of the i-th heat is $100 \sum_{k=1}^{K} w_k x_{ki}/W_i$.

As can be seen from the above equation that the objective function and constraints are all linear in terms of the decision variable x_{ki}. There are two difficulties that an LP solver can face in order to solve the above problem:

1. The number of decision variables and constraints are usually too large to solve them using classical methods, and
2. All decision variables are integer-valued.

The total number of decision variables are $n = HK$ and total number of constraints are $(H+K)$. However, since an LP solver will first convert the inequality constraints into equality constraints using *slack* variables (in one approach), the total number of decision variables for an LP solver become $n = (K+1)H$. For example, for $K = 50$ orders requiring $H = 200$ heats to complete all castings, the LP problem involves a total of $10,000$ integer-valued decision variables, 200 slack variables and a total of 250 equality constraints. Besides, the discreteness of the decision variables causes the major hurdle. The commonly-used technique to handle such problems is the branch-and-bound (BB) method (Deb, 1995; Reklaitis, Ravindran and Ragsdell, 1983). As shown in the sketch in Figure 1, the BB method first relaxes the integer restrictions and finds the real-parameter optimal solution. Thereafter, it chooses one particular variable (for which an non-integer value is obtained) and divides (branches) the original problem into two new problems (nodes) with an additional constraint to each problem. These procedure of branching into subproblems is terminated (bounded) when feasible integer-valued optimal solutions are found on a node or when a violation of some other optimality criteria cannot justify the continuation of the node. It is clear that as the number of decision variables increases, exponentially more such branching into new LPs are required, thereby making the overall approach computationally expensive.

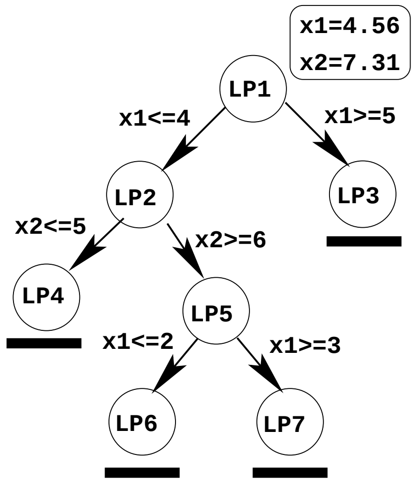

Fig. 1. A sketch of the working principle of the branch-and-bound method for a two-variable integer program. Every LP having a non-integer solution is branched into two LPs by adding an extra constraint on the variable.

3 Previously-Suggested Customized GA

It was clearly shown in another study (Deb, Reddy and Singh, 2003) that canonical binary-coded or real-coded GAs with generic crossover and mutation operators are not at all adequate for handling more than 500 variable version of the above casting scheduling problem. To solve the above problem close to optimality, that study introduced a customized GA. We first briefly describe that algorithm here.

3.1 Customized Initial Population

The initial population, instead of being created at random, is created in a way so as to satisfy all the equality constraints in each solution. Although it may sound impossible, the linearity of the equality constraints can be exploited and a simple procedure can be adopted. First, every variable x_{ki} is initialized within $[0, a]$ (a being an upper limit to the number of copies of an order k that can be made from one heat). Thereafter, we normalize each entry x_{ki} such that the total number of copies allocated for an order k is the same as that desired (r_k):

$$x_{ki} \leftarrow \frac{x_{ki}}{\sum_{i=1}^{H} x_{ki}} r_k. \tag{2}$$

3.2 Customized Recombination Operator

The main purpose of a recombination operator is to recombine partial good information of two or more solutions and create an offspring solution. In the context of the ILP problem stated in equation 1, a solution can be called good, if the overall utilization of molten metal is close to 100%, so that the corresponding casting strategy causes minimal waste of energy. However, a good (but not optimal) solution may not have close to 100% metal utilization for all its heats, but may have close to 100% utilization in some of its heats. These partial solutions, in which a good utilization has been already achieved, are *building blocks* of the problem and should be propagated from parent solutions to their offsprings. Thus, if two parent schedules are compared heat-wise and all x_{ki} values for the better i-th heat parent are copied to the child schedule, a literal *jump* towards the real optimum solution can be expected with such a recombination operator. However, it is intuitive that such a child may not satisfy the equality and inequality constraints, although both parents may be feasible. The created schedule was attempted to be made feasible by using two mutation operators.

3.3 Customized Mutation Operators

In the first mutation operator, the decision variables of a solution are altered so as to satisfy the *equality* constraints. For a solution, all x_{ki} values for each order k are added and compared with the desired number of castings r_k. If the added quantity is larger than r_k, we have to reduce x_{ki} from one of the heats. Here, we choose a heat which violates the inequality constraints maximally so that a fix-up for the equality constraints is achieved by modifying the worst-violated inequality constraints. On the other hand, if the added quantity is smaller than r_k then the reverse of the above procedure is followed. For details, readers may refer to the original study (Deb, Reddy and Singh, 2003).

Next, the modified solution is sent to the second mutation operator which attempts to alter the decision variables further without violating the equality constraints in order to satisfy the inequality constraints. The second mutation operator is started using the first heat and continued serially to the final heat. Each heat is checked for its inequality constraint violation. If the constraint is satisfied, there is no need to modify the heat. Otherwise, we look for the heat ($i_{\min}$) which has the minimum feasible utilization. To satisfy the constraint, it is necessary to transfer one copy of an order k (randomly chosen) from the i-th heat to the $i_{\min}$-th heat. After all infeasible heats are rectified as above, the feasibility of the overall solution is checked. If any solution is found infeasible at the end, a constraint violation equal to the sum of the normalized violations of all infeasible heats is assigned.

3.4 Salient Results Reported

The above customized GA was applied to the ILP problem having 100 to 1,006,750 (more than one million!) variables and the obtained scalability plots are reproduced in Figures 2 and 3. It is interesting to note that the proposed GA can solve a problem having as much as 1,006,750 integer decision variables in only about $1.6(10^7)$ normalized heat-updates in finding a solution with 99.84% efficiency.

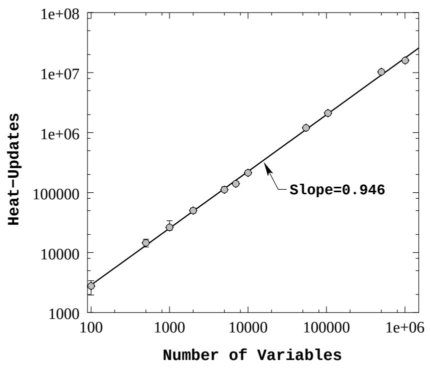

Fig. 2. Average, minimum, and maximum heat-updates needed to find a solution having at least 99.7% utilization of molten metal for different problem sizes.

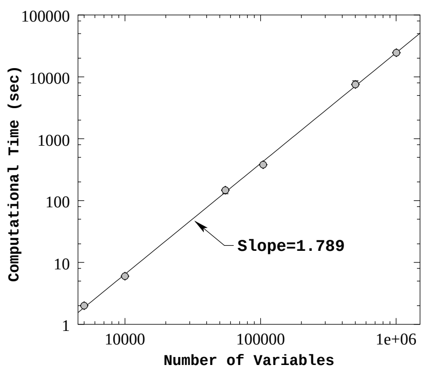

Fig. 3. Average, minimum, and maximum computational time needed to find the optimum solution.

Figure 3 shows the computational time needed to find the same optimized solutions on a 1.7 GHz Pentium IV processor. The figure shows a sub-quadratic computational time complexity of the proposed algorithm. The log-log plot shows a complexity of $O(n^{1.789})$, which is less than quadratic to the number of decision variables.

4 Extensions of the Customized GA

In this section, we introduced four different extensions to the above GA. In all extensions, we have slightly modified the second mutation operator as follows. In addition to choosing a casting (k) randomly to try to satisfy a violated inequality constraint (as described above), we choose the heaviest (largest w_k) possible casting for the exchange of a copy. This should make a bigger impact in preventing some potential solutions from getting lost and hence lead to faster convergence.

4.1 Multi-Parent Customized GA

In this extension, in addition to considering only two parents in the recombination operator, we consider μ random parents from the pool for creating one child solution. Like before, all μ parents are compared heat-wise and the components (x_{ki}) corresponding to the best individual utilization of the i-th heat are copied in the child. Since more parents are used in creating the child, a better child than that obtained in the original study is expected to be created. However, the use of too many parents may cause a loss of diversity among created solutions, thereby slowing down the progress towards the true optimum solution.

We first compute an estimate of the required number of heats by using 99.7% capacity of the crucibles. Each run is started with a population of size 30 and is run for a maximum of 40 generations till a feasible solution with utilization of 99.7% is found. If such a solution is not found, we restart the procedure with a new population. We repeat this a maximum of five times. If no such solution is found after five iterations, the number of heats is increased and the whole procedure is repeated. If we get such a solution within five iterations, we decrease the number of heats and redo the process. We continue reducing the number of heats till no such feasible solution is found. The number of heats we mentioned in the paper is the minimum number of heats needed to find a high-quality feasible solution. When a number of heats is successful, we compute the heat-update for the best solution. Since the above procedure requires larger-heat runs, we find the minimum, average and maximum values of such heat-updates for all previous successful runs (with larger heats) and present here. The computational time is the time required to find the feasible solution in each run. The average, minimum and maximum time of all such successful runs (with varying number of heats) are presented here.

By keeping the population size fixed at 30 and by varying the parent size from three to 10 on the 50,000-variable ILP problem (given in equation 1), we observe from Table 1 that the use of seven parents require smaller average number of function evaluations with minimal number of heats.

Table 1. Function evaluations and computational time recorded for different parent size in the multi-parent approach in solving the 50,000-variable ILP problem. A population size of 30 is used.

	Util.	Heats	Function Evaluations			Computational Time (sec)		
μ	(%)	H	Avg.	Max.	Min.	Avg.	Max.	Min.
3	99.97	5487	58491.4	61838.5	54046.9	94.3	129	68
5	99.97	5487	50974.2	59369.3	47462.6	129.9	169	67
6	99.97	5487	54211.6	54485.9	46913.8	149.3	227	65
7	99.97	5487	**48505.1**	54705.4	45158	143.7	227	65
8	99.95	5489	48797.2	52914	45942.9	132.57	237	78
9	99.95	5489	50279.2	55384	47973.9	138.23	225	81
10	99.93	5490	50068.8	53582.4	45621.9	141.2	273	75

In this problem, $R = 29,466$ castings for $K = 10$ different shapes, each having different weights and orders (as shown below), are to be made:

Casting (k)	1	2	3	4	5	6	7	8	9	10
Copies (r_k)	3427	3329	3327	3229	3429	3526	3122	1322	2529	2226
Weight (kg) (w_k)	175	150	85	75	95	75	95	200	85	50

This requires a total of $M = 3,101,505$ kg molten metal. By using $W_I = 650$ kg and $W_{II} = 500$ kg and corresponding maximum daily heats $H_I = 10$ and

$H_{II} = 13$, it can be calculated that at least 5,487 heats involving 478 days will be required to melt the required amount of metal. This can be explained as follows. 239 days (or 239×10 or 2,390 heats) of W_I requires $239 \times (650 \times 10)$ or 1,553,500 kg and 238 days (or 238×13 or 3,094 heats) of W_{II} requires $238 \times (500 \times 13)$ or 1,553,500 kg metal. The remaining 1,005 kg metal would require at least three more heats involving $W_{II} = 500$ kg crucible. Thus, a total of minimum $2,390+3,094+3$ or $5,487$ heats are necessary to complete the job. Table 1 shows that runs with parent size (μ) varying from three to seven are able to find this optimum solution, although the number of function evaluations needed to arrive at this solution is more for smaller parents. The table clearly indicates that the seven-parent strategy is optimal for this problem having 50,000 variables.

Next, we perform a systematic study by varying the size of the problem from 100 to 100,000. In each case, we fix a parent size of seven and a population size of 30. The average, minimum, and worst number of function evaluations and computational times are tabulated in Table 2. The table also shows the percentage utilization and number of heats obtained for each case and compares those with the minimum possible number of heats (not necessarily feasible). The second column indicates the total metal required to cast all orders and the

Table 2. Minimum possible heats and results obtained with the multi-parent strategy with seven parents and 30 population members.

Variable	Total metal	Days	Min. Heats	Multi-parent GA results	
Size	reqd. (M, kg)		H	Heats (H)	Util. (%)
100	6,475	1	10	10	99.62
5,000	282,065	44	499	499	99.83
10,000	281,992	44	498	499	99.83
50,000	3,101,505	478	5,487	5,487	99.97
100,000	5,891,505	907	10,423	10,428	99.94

third column indicates the number of days required to complete the task. The next column indicates the minimum number of heats necessary to melt M kg of metal, however in order to make a feasible overall schedule, more heats may be necessary. Since the number of orders (r_k) and weight (w_k) of each casting are arbitrarily chosen, it is difficult to establish if the minimum number of heats indicated in column 4 would be adequate for the task. However, it is true that a schedule with a smaller number of heats than what is indicated in column 4 is inadequate to make all castings. For 100, 5,000, and 50,000 variable cases, the minimum possible heats are obtained in the multi-parent GA, however in the other cases although the minimum possible heats are not achieved, close values are obtained. It is important to reiterate that the marked minimum possible heats may not be the true optimal solutions. These numbers simply indicate the minimum heats required to melt the adequate amount of metal.

We have rerun the 100,000-variable case for a maximum generation of 60 and kept all other parameters the same as before. This time we obtain 10,425 heats adequate to have a solution with 99.98% utilization and requiring an average of 77,666.3 heat-updates and 548.8 sec computational time.

Figure 4 shows how the computational time varies with the increase in number of variables from 100 to 100,000. Over this wide range, the computational time is found to vary very close to linearity. The fitted straight line on the log-log plot has a slope of 1.153 indicating a time complexity of $O(n^{1.153})$. Since the plotted computational time is the average of execution times needed for a number of runs with different heat count (H), the larger-heats runs require more time and lesser-heats run require smaller time, thereby producing a large fluctuation in the time values. But the average time indicates the overall time needed to complete one heat run. Figure 5 shows the variation of function evaluations needed to have either the optimal number of heats or a utilization efficiency of over 99.7%. A near-linear ($O(n^{0.998})$) complexity is observed in Figure 5.

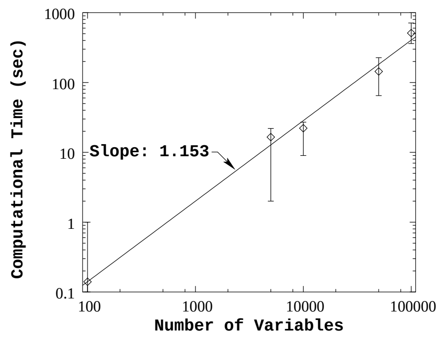

Fig. 4. Computational time complexity for the multi-parent approach.

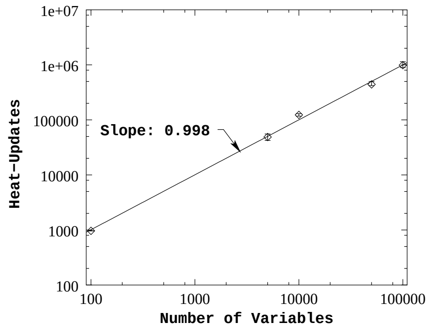

Fig. 5. Complexity of function evaluations for the multi-parent approach.

4.2 Steady-State Customized GAs

The original study (Deb, Reddy and Singh, 2003) and the above multi-parent study used a generational model of a GA. For better convergence, steady-state GAs are used in other contexts. In the next three customizations, we use the multi-parent recombinative strategy and produce one child after two mutation operators, as mentioned earlier. However, after each child is created, it is compared with the parents for its inclusion in the main population.

Approach 1: In the first approach, we compare all μ participating parents and the child and the best μ solutions are kept in the population.

Approach 2: In the second approach, the worst member of the entire current population is replaced with the child solution.

Approach 3: In the third approach, a random population member is chosen and compared with the new-born child. The better of the two solutions is kept in the population.

We compare the above steady-state approaches with multi-parent strategy and the original study (with the modified mutation operator indicated in Section 4) on the 50,000 variable problem. For each case, the best parent size and population size are found by performing an extensive parametric study and time and function evaluation results are shown in Table 3. It is clear that all multi-

Table 3. Best results of this study are compared with the original study.

Approach	N	μ	Heats	Util.	Avg. Func. Eval.	Avg. Time (sec)
Original	30	2	5,488	99.97	112,888.2	283.0
Multi-parent	30	7	5,487	99.97	48,505.1	143.7
Approach 1	25	6	5,487	99.99	23,045.4	95.8
Approach 2	25	7	5,490	99.94	22,618.8	82.1
Approach 3	25	8	5,490	99.94	22,069.8	109.2

parent GAs perform better than the original two-parent algorithm and the first steady-state approach and the generational multi-parent approach found the truly minimum solution (with 5,487 heats), whereas others require more heats involving a lesser utilization of metal. However, between these two approaches, the steady-state approach requires much less average computational time.

Figures 6 and 7 show the scalability of the first steady-state approach in terms of computational time and function evaluations for variables from 100 to 100,000. It is observed that the time complexity varies as $O(n^{1.055})$ and the function evaluations as $O(n^{0.963})$ to obtain either the optimal solution or a minimum utilization of 99.7%.

We perform similar studies for the other two steady-state approaches and similar results are obtained. We show the computational time variations for the two approaches in Figures 8 and 9, respectively. The corresponding time complexities are $O(n^{1.025})$ and $O(n^{1.026})$. The corresponding complexities on the required number of function evaluations are $O(n^{0.894})$ and $O(n^{0.928})$, respectively.

To put the above results into a better perspective, the best approach of this paper finds a near-optimum solution of the 100,000-variable ILP problem within only 130 sec on a Pentium IV processor. Solving ILP problems as large as 100,000 variables in an almost linear computational time complexity is rarely demonstrated in the EA literature. By performing extensive parametric studies for different algorithms proposed in this paper, we believe to have developed very efficient population-based steady-state optimization algorithms for solving ILP problems which would be difficult to match in terms of computational time or overall function evaluations by classical point-by-point search approaches.

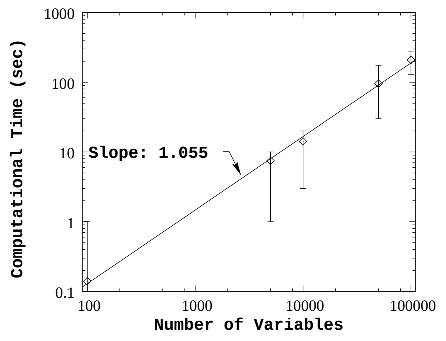

Fig. 6. Computational time complexity for the steady-state approach 1.

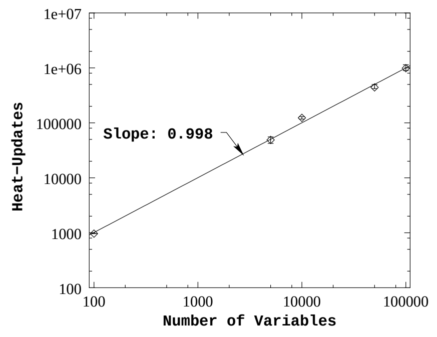

Fig. 7. Complexity of function evaluations for the steady-state approach 1.

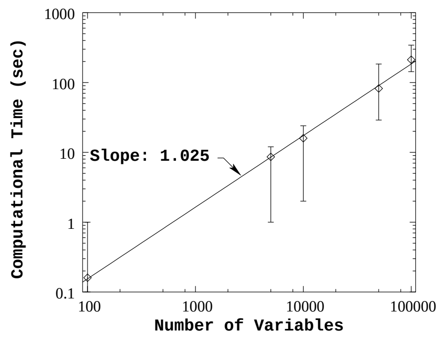

Fig. 8. Computational time complexity for the steady-state approach 2.

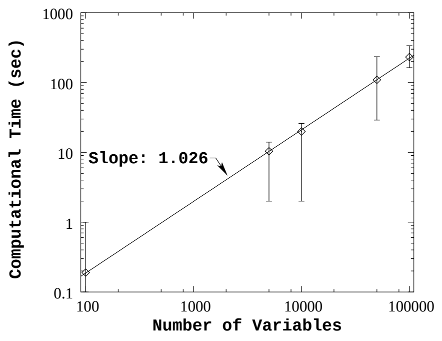

Fig. 9. Computational time complexity for the steady-state approach 3.

5 Conclusions

Despite many applications of evolutionary algorithms (EAs) to various search and optimization problems, there exists very few studies where EAs have been tested on very large sized problems, such as problems having hundreds or thousands of variables. An earlier study demonstrated the use of a customized GA for the purpose, in which initialization and genetic operators are all customized using the problem knowledge. In this paper, we have extended the idea further to solve integer linear programs often arising in scheduling and resource allocation problems. A multi-parent version and three steady-state GAs have been suggested and extensive parametric studies have been performed to develop efficient optimization algorithms. Following conclusions can be made from this study:

1. The parametric study on population size indicates an optimal population size varying between 25 to 30, indicating that a population-based approach

(not a point-by-point approach) is an optimal procedure for solving the ILP problems considered here.

2. The parametric study on parent size indicates that the usual consideration of two parent recombination is not the optimal approach. However, there lies an optimal parent size (about six to eight parents) with which the suggested algorithms work the best. This is due to the exploration and exploitation balance which must be honored in a population-based search algorithm in order to constitute a successful search process.
3. The multi-parent steady-state approaches have been found to be better than the multi-parent generational approach both in terms of computational time and amount of function evaluations.
4. The results of this study are better than those reported in the original study in which two parents were used. The best computational time complexity of this study is $O(n^{1.025})$, whereas the best reported in the original study was $O(n^{1.789})$.

This paper has demonstrated the power of a recombinative GA in solving large-sized problems to optimality, as the use of multiple good parents to construct a new solution may often provide *jumps* towards the minimum solution in the case of linear problems, a matter which is not possible to obtain with point-by-point classical approaches. Hopefully, this study will motivate the design and application of EAs to similar other large-scale real-world optimization problems. The approach used here may be directly applicable to other scheduling problems (with discrete variables) having a set of linear equality and inequality constraints, such as the commonly-used knap-sack problems, however, the idea of using a good initial population, a steady-state approach, a multi-parent recombination, and the use of knowledge-based genetic operators is important in solving very large-sized problems.

References

Deb, K. (1995). *Optimization for engineering design: Algorithms and examples.* New Delhi: Prentice-Hall.

Deb, K., Reddy, A. R., and Singh, G. (2003). Optimal scheduling of casting sequence using genetic algorithms. *Journal of Materials and Manufacturing Processes, 18*(3). 409–432.

Glover, F. and Laguna, M. (1997). *Tabu search.* Boston: Kluwer.

Goldberg, D. E. (1989). *Genetic Algorithms for Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*, 671–680.

Reklaitis, G. V., Ravindran, A. and Ragsdell, K. M. (1983). *Engineering Optimization Methods and Applications.* New York: Wiley.

Taha, H. A. (1989): *Operations Research.* New York: Macmillan.