A Genetic Algorithm for Linear Ordering Problem Using an Approximate Fitness Evaluation

Jinhyun Kim
School of Computer Science & Engineering
Seoul National University
1 Gwanak-ro, Gwanak-gu
Seoul, 151-744 Korea
jh@soar.snu.ac.kr

Byung-Ro Moon
School of Computer Science & Engineering
Seoul National University
1 Gwanak-ro, Gwanak-gu
Seoul, 151-744 Korea
moon@snu.ac.kr

ABSTRACT

Genetic algorithms are widely used to solve combinatorial optimization problems, but they often take a long time. Usually, generating and evaluating a large number of different solutions spend most of the running time. We propose a genetic algorithm for the linear ordering problem which uses an approximate fitness evaluation. We use a part of the edges to compute the fitness function value, and the number of the edges for this is gradually increased during the evolutionary process. We present experimental results on the benchmark library LOLIB. The approximation scheme reduced the running time without loss of solution quality in general.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Global optimization*; G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*

General Terms

Algorithms

Keywords

Genetic algorithm, fitness approximation, linear ordering problem

1. INTRODUCTION

Let $G = (V, E)$ be a directed graph with N vertices and M edges, and $w(u, v)$ be the weight of an edge from vertex u to v. The linear ordering problem(LOP) is to find a linear ordering of vertices $\langle v_1, v_2, \cdots, v_N \rangle$ which maximizes the fitness function

$$f(\langle v_1, v_2, \cdots, v_N \rangle) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} w(v_i, v_j).$$

GECCO'14, July 12–16, 2014, Vancouver, BC, Canada.
ACM 978-1-4503-2881-4/14/07.
http://dx.doi.org/10.1145/2598394.2602282.

If the graph does not contain a cycle, then a topological ordering is an optimal solution; if not, the problem is equivalent to feedback arc set problem(FASP) which tries to remove minimal edges from the graph in order to make the graph acyclic.

LOP, FASP, and several other related problems have been studied for a long time [1]. And these problems have a number of applications in various fields [3]. The problem is known to be NP-hard and many metaheuristic approaches, including evolutionary approaches have been proposed [2]. It is known that genetic algorithms are suitable for solving combinatorial optimization problems. But usually they are slow since a large number of candidate solutions are generated and evaluated during the evolutionary process.

In this paper, we propose a genetic algorithm for LOP which evaluates an approximate fitness value. In the early generation of the GA, only a portion of the edges are used to evaluate the fitness. We increase the number of used edges from 0 to M over time. The proposed algorithm was tested on well-known instances. Experimental results show that the proposed approach reduces the running time without losing the quality of the solutions.

2. THE PROPOSED METHOD

We use a typical GA to solve the problem.

- **Population**: The population consists of 100 permutations, each representing a linear ordering of the vertices. We generate 20 offspring in each generation. Among the 120 solutions, the best 100 solutions form the population of the next generation.
- **Selection**: We randomly pick two solutions, and return the better one with probability 80%, and the worse one with probability 20%.
- **Crossover**: We use order based crossover [3]. First, each gene is copied from one of the two parents. Then we pick some genes and reorder them with respect to their order in the other parent. The probability for each gene to be picked is 50%.
- **Mutation**: Each gene has 1% chance of mutation, and the selected genes are randomly shuffled.
- **Stopping Criterion**: The GA runs for a fixed number of generations and stops. The number of generations K depends on the characteristic of the instance.

To compute the fitness directly, all N^2 pairs of vertices has be considered. By considering the edges of the graph, the fitness can be rewritten as

$$\sum_{(u,v)\in E} w(u,v)R(u,v).$$

R is a binary relation on V and $R(u,v)$ is defined to be 1 if u comes before v in the ordering, and 0 otherwise.

To approximate the fitness, we use the subset $E' \subseteq E$ which has M' edges, and compute the summation only for $(u,v) \in E'$. The accuracy of the approximation is controlled by the size of E'; it is more accurate if the size is close to M. We set the size to be 0 in the first generation, and to be M in the half-way point. During the first half of the GA, the size is gradually and linearly increased as the generation progresses. In the second half, the GA is as same as the one using exact fitness evaluation. The GA travels the search space almost randomly in the earlier generations, and it gradually finds the right direction later.

The edges in E' need to be changed at each time of the fitness evaluation. GA could easily be stuck in a local optimum, if E' has only few edges in it and it is being kept similar for a while. However, it is inefficient to randomly choose M' edges all the time. It takes time to generate a random number, and shuffling the set of edges diminishes the cache utilization. Instead, we only change some part of the elements in E'. We first randomly shuffle the list of edges, and the first M' edges form the subset E'. We exchange $0.005 \times M$ random pairs of edges at each time of evaluation; the i^{th} and the j^{th} edges(i, j are randomly picked from $\{1, 2, \cdots, M'\}$ and $\{1, 2, \cdots, M\}$).

3. EXPERIMENTAL RESULTS

The proposed algorithm was tested on the a widely used benchmark library LOLIB [2]. There are 485 instances, comprising both real-world instances and randomly generated ones. Among them, we selected the instances with sizes $N = 50, 100, 150, 200, 250$, since there exist a sufficient number of instances having those sizes. An instance having an error, `N-t65f11xx_150`, was excluded.

The number of generations K was selected in order to guarantee that the population is converged after K generations. We assume that the population is converged if no change is made to the population for 100 generations. An appropriate value of K for each instance was chosen after a sufficient number of experiments.

We conducted experiments with 4 different kinds of GAs. Two of them are the GA using exact fitness(`EXACT-K`) and the proposed GA using an approximate fitness(`APPROX-K`). The other two GAs run for $2K$ generations, and each of them uses either exact(`EXACT-2K`) or approximate fitness(`APPROX-2K`). We measured the fitness value of an optimal solution found and the running time for each run. We conducted 100 runs for each instance and each algorithm, and computed the average result.

Table 1 shows the comparison between `APPROX` algorithms and `EXACT` algorithms. For each instance, we compute the relative fitness and time. The relative fitness is the ratio of the average fitness value obtained by the algorithm with approximation to the average fitness value obtained by the algorithm without approximation. The relative time is computed similarly. Then, we averaged the relative values over the instances with the same size.

Table 1: Comparing the performance of the algorithms in terms of fitness and time. `APPROX-K` was compared to `EXACT-2K`, and `APPROX-2K` was compared to `EXACT-2K`.

N	`APPROX-K` to `EXACT-K`		`APPROX-2K` to `EXACT-2K`	
	fitness	time	fitness	time
50	0.9990	0.9819	1.0001	0.9821
100	1.0001	0.9426	1.0004	0.9446
150	0.9989	0.9595	0.9998	0.9595
200	1.0015	0.9346	1.0013	0.9342
250	0.9966	0.9905	0.9983	0.9907

By using the approximate fitness evaluation, the running time was reduced for both of the cases with K and $2K$. However, the quality of an optimal solution found did not changed significantly by using the approximation scheme. In fact, it was even better for some of the instances.

The relative performance of `APPROX-2K` was slightly better than that of `APPROX-K`. Note that the genetic algorithm was converged in K generations. This suggests that, when running a genetic algorithm with sufficiently large number of generations, the approximation scheme helps to escape from a local optimum. Using the approximate fitness evaluation gives a chance of accepting a worse solution, and the probability gradually decreases during the space search. The same idea is often used in some metaheuristics, and the representative one of them is simulated annealing.

4. CONCLUSION

In this paper, we proposed a genetic algorithm for LOP which uses the subset of edges to compute the approximate fitness value of a solution. The accuracy of the approximation was gradually increased the first half of the evolution, and exact fitness was used in the second half. The preliminary experimental results showed that the running time was decreased, but the performance was not harmed as well. The proposed scheme seems to be useful when the number of generations is large enough.

5. ACKNOWLEDGMENTS

This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Science, ICT & Future Planning(MSIP) / National Research Foundation of Korea(NRF) (Grant NRF-2008-0062609). The ICT at Seoul National University provided research facilities for this study.

6. REFERENCES

[1] M. Grötschel, M. Jünger, and G. Reinelt. A cutting plane algorithm for the linear ordering problem. *Operations Research*, 32(6):1195–1220, 1984.

[2] R. Martí, G. Reinelt, and A. Duarte. A benchmark library and a comparison of heuristic methods for the linear ordering problem. *Comput. Optim. Appl.*, 51(3):1297–1317, Apr. 2012.

[3] T. Schiavinotto and T. Stützle. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402, 2005.