Efficient Solutions to a Class of Generalized Time-Dependent Combinatorial Optimization Problems

Todd W. Kolb and Lixin Tao
School of Computer Science and Information Systems
Pace University
1 Martine Ave., White Plains, NY 10606
USA

Michael W. Tao
Thomas Jefferson High School for Science and Technology
6560 Braddock Road, Alexandria, VA 22312
USA

Abstract: A class of generalized intractable time-dependent problems is identified and abstracted into a mathematical model. Simulated annealing is adopted as the base of the solution strategy. Comprehensive experiments are conducted to study the sensitivity of the algorithm to the values of its multiple parameters. Extensive performance evaluation shows that the proposed algorithm significantly outperforms the best published alternative algorithms for the same problem class.

Key-Words: Time-dependent combinatorial optimization problems, meta-heuristics, simulated annealing.

1 Introduction

As a branch of operations research, combinatorial optimization plays an important role in obtaining efficient solutions to NP-hard problems common in science, engineering, and management domains. In 2001, Michael L. Gargano and William Edelson [3] described a model of time-dependent combinatorial optimization problems, and outlined several important applications of the model in terms of the genetic algorithm meta-heuristic. The model assumes a set of unit-time tasks to be completed sequentially, a set of workers bidding on completing the tasks with time-dependent costs, and seeks an optimized ordering for completing the tasks and an optimized assignment of the tasks to the workers for minimizing the total cost. The model can be subdivided into sub-models depending on whether a worker can bid on multiple tasks, or whether a worker can be assigned to completing more than one task.

In 2002, Joseph DeCicco [1] used genetic algorithm to solve a sub-model of the above problems in which a worker can only bid on one task and be assigned at most one task. In 2004, Rigoberto Diaz [2] established the mathematical model for the same sub-model of problems that Joseph DeCicco worked on, simplified the problems in the sub-model with a novel problem transformation algorithm, and claimed significantly reduced solution costs and algorithm time-complexity with his algorithm based on simulated annealing meta-heuristic. In 2004, Maheswara Kasinadhuni [7] proposed a new genetic algorithm heuristic to solve the same sub-model of problems with multiple genome coding.

This research focuses on the efficient solutions to the most difficult sub-model of the above time-dependent combinatorial optimization problems in which a worker can bid on multiple tasks but can be assigned to at most one task. Since the task assignment for a worker now depends on all previous such assignments, Diaz's problem transformation cannot be applied to simplify the problems. In addition to its own value in solving more difficult real-world problems, this sub-model of problems also provides a platform for more objective comparison of solution quality and time-complexity of different solution approaches for the timed-dependent combinatorial optimization problems since problem transformation cannot be applied any more.

Efficient solution algorithms based on exhausted search, repeated random solutions, genetic algorithm, and simulated annealing were designed and implemented in Java. Except the exhaust search algorithm that can find optimal solutions to problem instances of size less than 10, others are all heuristic algorithms that can provide efficient solutions for problem instances of size larger than 200. Extensive experimental study shows that the simulated annealing algorithm outperforms all of the other approaches.

2 Problem Formulation

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of n workers. Let $S = \{1, 2, \ldots, s\}$, $1 < s \le n$, be a set of s tasks, where each task takes a unit of time to complete, and the tasks must be completed sequentially in consecutive time units. Let $\{X_1, X_2, \ldots, X_s\}$ be a set of non-empty subsets of X, where for each $1 \le i \le s$, X_i contains the workers of X who can

work on task *i*. Let $\wp_s$ be the set of all permutations on set $S = \{1, 2, \ldots, s\}$ representing all the orderings of the tasks to be completed, $\Re^+$ the set of all nonnegative real numbers, and $c : X \times S \times S \to \Re^+$ a given cost function where for each $x \in X$ and $i, j \in S$, $c(x,i,j) = \infty$ if and only if $x \notin X_i$, and $c(x,i,j)$ is the cost for worker x to complete task *i* in time unit *j* otherwise.
Find a vector
$v = (x_1, x_2, \ldots, x_s) \in X_1 \times X_2 \times \ldots \times X_s$ where each component of *v* is unique, and a permutation $\pi \in \wp_s$ such that the objective function

$$f(v, \pi) = \sum_{i=1}^{s} c(x_i, i, \pi(i))$$

is minimized.

This problem formulation contains *s* tasks and *n* workers, where $0 < s < n$. Each worker is capable of working on one or more tasks, and each task can be completed by any of its eligible workers in a unit of time. All tasks must be completed sequentially in consecutive time units. The cost for a worker to complete a task depends on the time slot in which the task in conducted. This problem tries to find one worker for each task (vector *v*) and an ordering of the tasks (π) in which they will be completed such that the total cost would be minimal.

As an example application of this problem formulation, let us consider the *Highway Minimum Bidding Problem*. There is a need to build *s* highway segments to connect some cities. There are *n* ($n \geq s$) companies bidding to build the highway *segments*. The segments will be built in successive months, one in each month. Each bidding company can bid on the construction of multiple highway segments, be assigned to the construction of no more than one highway segment, and the cost of the bids vary with the month in which construction occurs. Assign the bidding companies to build the highway segments in a particular order so that the total cost is minimized.

As a particular problem instance, there are four highway segments involved. Due to budget limits, the four segments must be built in four consecutive months, one month for each segment, but the order of their construction is not important. There are nine companies bidding on the construction of the four segments, each can bid on multiple segments. Let us assign unique identification numbers 1, 2, …, 9 to these companies. Table 1 shows the costs for the bidding companies to build highway segments in particular months. For example, for company 2 to build segment 1 in month 3, the bidding cost is 12. A feasible solution (not optimal) is to let company 5 build segment 2 in month 1, company 2 build segment 1 in month 2, company 6 build segment 3 in month 3, and company 4 build segment 4 in month 4. The resulting total cost is 65 + 36 + 65 + 12 = 178. We notice that company 4 could build segment 2 in month 1 with a lower cost of 57, instead of the current 65. But since each company can only be assigned to no more than one segment, the fact that company 4 is assigned to segment 4 prevents it from being assigned to segment 2.

Table 1 Highway Bidding Costs

highway	company	cost (in millions of dollars)			
		M1	M2	M3	M4
1	1	21	47	18	85
	2	66	36	12	28
	3	43	59	89	18
	7	42	42	30	29
2	4	57	41	85	52
	5	65	23	62	13
3	6	80	11	65	17
	7	25	72	31	24
	8	21	75	86	22
	1	40	52	50	33
4	9	63	79	32	78
	1	21	30	62	39
	4	20	85	79	12

3 Problem Solution

In this section we describe the design and implementation of our simulated annealing algorithm for time-dependent problems based on the simulated annealing meta-heuristic. We also design experiments to conduct sensitivity analysis of the heuristic to its various parameter values.

The simulated annealing meta-heuristic described by David S. Johnson [6] will be used as the base of our solution to the class of time-dependent combinatorial optimization problems that can be modeled with our problem formulation. Based on our problem formulation, a feasible solution to our problems is of form $\theta = (v, \pi)$ where $v = (x_1, x_2, \ldots, x_s) \in X_1 \times X_2 \times \ldots \times X_s$ represents the assignment of the workers to the tasks, and π is a permutation of numbers in {1, 2, …, s} representing the order in which the tasks will be conducted. Given a current solution $\theta = (v, \pi)$, we define a *move* on it to be either randomly swapping two values in π, or randomly changing the worker assignment for one randomly chosen task. The algorithm is described in

the pseudo-code below.

```
Get a random initial solution θ as the current solution.
Let temperature t = t0, the initial temperature.
While there are improvements of the best cost in the last k
      iterations do
   While there are improvements of the best cost in the
      last l iterations do
      Perform the following loop l times.
         Let θ' be a random neighbor of θ.
         Let Δ = f(θ') – f(θ).
         If Δ ≤ 0 (downhill move), set θ = θ'.
         If Δ > 0 (uphill move),
            set θ = θ' with probability e^(–Δ/t).
   End While.
   Set t = r · t (reduce temperature).
End While.
```

There are four parameters that we need to configure:

- Initial temperature t_0. A too large value for t_0 will lead to wasted random walk in the solution space at the beginning of the algorithm execution thus prolong the algorithm's running time without the benefit of improving solution quality. A too small value for t_0 will let the algorithm get stuck in a local optimum.
- Temperature reduction ratio r. Ratio r should be a real number between 0.0 and 1.0. If it is too large, the temperature will be reduced very slowly, leading to prolonged algorithm execution. On the other hand, if r is too small, the temperature will be reduced too fast and the current solution can get stuck in a local optimum.
- Number l of consecutive non-improvement iterations before the temperature is reduced. If l is too large, the algorithm may waste execution time in a prolonged non-aggressive solution search. If l is too small, then the current solution may not have a chance to settle down to a stable good solution.
- Number k of consecutive non-improvement iterations before the algorithm is terminated. If k is too large, execution may be extended without quality benefit. If k is too small, then the algorithm may terminate too soon before better solutions could be obtained.

These four parameters are not independent. As a matter of fact, they have close inter-dependence. It is a big challenge to find optimized values for them so that the resulting algorithm can perform well on a large set of potential problem instances [5].

We run the algorithm on seven training problem instances with value of s ranging from 4 to 200. After experimenting with various parameter combinations, we decided the effective ranges for the four parameters are as follows:

Table 2. Chosen parameter values for simulated annealing.

t_0	r	l	k
13	0.995	1400	160

4 Performance Comparisons

Since heuristics for solving combinatorial optimization problems are not based on theoretical analysis, the only objective way to evaluate their performance is by conducting comparative study based on a large enough set of benchmark problem instances.

In this section, we first define the experimental environment and problem instances. Then a thorough performance evaluation will be conducted to compare both the solution quality and running time for three different heuristics for solving the same time-dependent problems.

4.1 Experiment Design

All experiments were conducted on a Hewlett-Packard xw4200 workstation with an Intel Pentium 4: 3.4 GHz with EM64T CPU, 2 GB RAM, and running the Sun Microsystems Solaris 10 (x86) operating system in 64-bit mode.

Seventy problem instances, generated with a random number generator, are used for performance evaluations. These problem instances have task numbers 4, 10, 20, 30, 50, 100, and 200; worker numbers vary from 6 to 425; and bid numbers vary from 9 to 637. There are ten problem instances for each of the task numbers.

4.2 Simulated Annealing vs. Genetic Algorithm

For this experiment, we compare the performances of simulated annealing and genetic algorithm. For each of the 70 benchmark problem instances, we use each of the above two algorithms to run ten times and report the best cost, worst cost, and average cost for the problem instance, as well as the best running time, worst running time, and average running time.

Table 2 and Figure 1 compare the solution quality and running time between simulated annealing and genetic algorithm. We observe that, except for the trivial cases where $s = 4$, the cost values of simulated annealing significantly outperform those for genetic

algorithm. The larger the problem instances, the more simulated annealing outperformed genetic algorithm. When s = 200, the simulated annealing improved the average costs of the genetic algorithm's by 72%. The running times of simulated annealing are on the same order as those of the genetic algorithms.

Table 2 Solution Quality comparison between SA and GA

File Name and Algorithm	Best Cost	Average Cost	Best Time	Average Time
Simulated Annealing (SA)				
h4s04c7b10.txt	133	133	202	207
h10s10c20b30.txt	162	162	292	362
h20s01c44b66.txt	235	242	602	952
h30s08c60b90.txt	343	360	1354	1575
h50s09c91b136.txt	556	582	2248	2728
h100s03c203b304.txt	1105	1129	6584	8291
h200s06c403b604.txt	2179	2205	19056	22417
Genetic Algorithm (GA)				
h4s04c7b10.txt	133	133	12	15
h10s10c20b30.txt	174	183	24	43
h20s01c44b66.txt	371	413	68	201
h30s08c60b90.txt	626	683	218	439
h50s09c91b136.txt	1251	1415	563	1259
h100s03c203b304.txt	3022	3458	997	4879
h200s06c403b604.txt	7379	7965	4575	23593

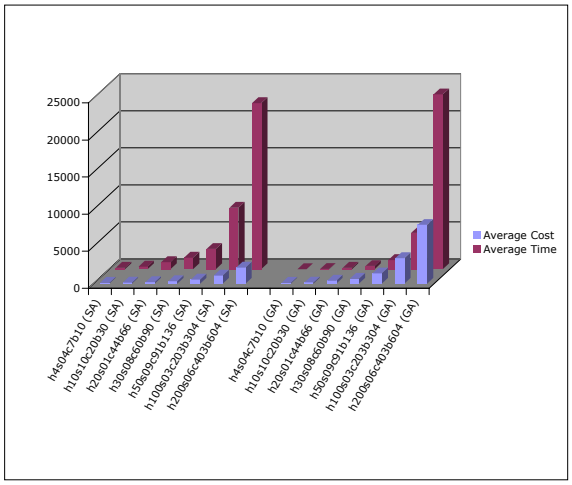

Figure 1 SA vs. GA Solution Comparisons

4.3 Comparisons between Simulated Annealing and Repeated Random Solutions

For a heuristic to prove its value in combinatorial optimization, it must show that it can produce better solutions than repeatedly generated random solutions in the same amount of running time. In this and the following two sections, we conduct this type of performance evaluation for both simulated annealing and genetic algorithm.

Table 3 and Figure 2 show the comparison of solution quality of simulated annealing with that of the Repeated Random heuristic. In the same amount of running time, simulated annealing greatly improved the costs of repeated random by up to 74%.

Table 3 SA vs. RR Solution Comparisons

File Name and Algorithm	Best Cost	Average Cost	Best Time	Average Time
Simulated Annealing (SA)				
h4s04c7b10.txt	133	133	202	207
h10s10c20b30.txt	162	162	292	362
h20s01c44b66.txt	235	242	602	952
h30s08c60b90.txt	343	360	1354	1575
h50s09c91b136.txt	556	582	2248	2728
h100s03c203b304.txt	1105	1129	6584	8291
h200s06c403b604.txt	2179	2205	19056	22417
Repeat Random for SA				
h4s04c7b10.txt	133	133	207	207
h10s10c20b30.txt	212	225	362	362
h20s01c44b66.txt	476	541	952	952
h30s08c60b90.txt	872	948	1575	1575
h50s09c91b136.txt	1772	1815	2728	2728
h100s03c203b304.txt	3872	3933	8291	8291
h200s06c403b604.txt	8404	8550	22417	22417

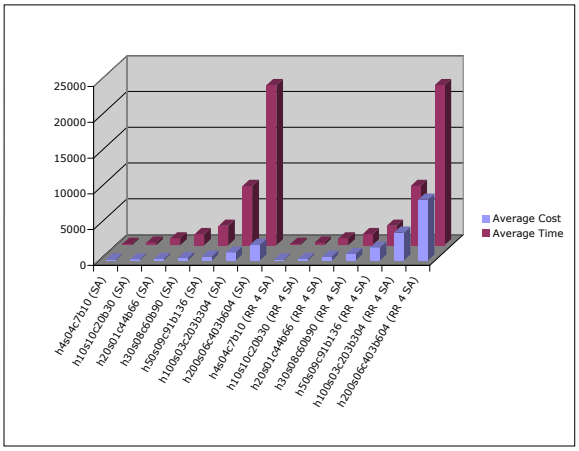

Figure 2 SA vs. RR Solution Comparisons

4.4 Comparisons between Genetic Algorithm and Repeated Random Solutions

Table 4 and Figure 3 compare the solution quality between the genetic algorithm and repeated random solutions. The repeated random heuristic was

executed for as long as the average running time of the genetic algorithm for each particular problem instance. In the same amount of running time, the genetic algorithm greatly worsened the average costs of repeated random by up to 71%.

Table 4 GA vs. RR Solution Comparisons

File Name and Algorithm	Best Cost	Average Cost	Best Time	Average Time
Genetic Algorithm (GA)				
h4s04c7b10.txt	133	133	12	15
h10s10c20b30.txt	174	183	24	43
h20s01c44b66.txt	371	413	68	201
h30s08c60b90.txt	626	683	218	439
h50s09c91b136.txt	1251	1415	563	1259
h100s03c203b304.txt	3022	3458	997	4879
h200s06c403b604.txt	7379	7965	4575	23593
Repeat Random 4 GA				
h4s04c7b10.txt	133	133	15	17
h10s10c20b30.txt	213	247	43	43
h20s01c44b66.txt	574	598	201	201
h30s08c60b90.txt	962	1006	439	439
h50s09c91b136.txt	1730	1819	1259	1262
h100s03c203b304.txt	3858	3926	4879	4879
h200s06c403b604.txt	8459	8536	23593	23593

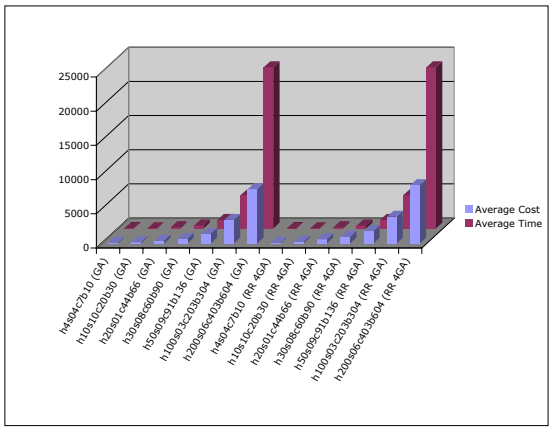

Figure 3 GA vs. RR Solution Comparisons

5 Conclusion

This paper generalized a previous problem formulation of time-dependent combinatorial optimization problems. Extensive experimental results show that the proposed simulated annealing algorithm outperforms the costs of the genetic algorithm and repeated random solutions by up to 72%.

References:

[1] Joseph DeCicco, "Sensitivity Analysis of Certain Time Dependent Matroid Base Models Solved by Genetic Algorithms," DPS dissertation, CSIS, Pace University, New York, May 11, 2002.

[2] Rigoberto Diaz, Lixin Tao, Michael Gargano, Fred Grossman, and Michael W. Tao. "Solving a class of time-dependent combinatorial optimization problems with abstraction, transformation and simulated annealing," *IADIS International Conference of Applied Computing*, March 23-26, 2004, Lisbon, Portugal. pp. I-535-I-542.

[3]Michael L. Gargano, William Edelson. "Optimal Sequenced Matroid Bases Solved by Genetic Algorihms with Feasibility Including Applications," Congressus Numerantium 150, 2001, pp. 5-14.

[4]Michael R. Garey and David S. Johnson. "Computers and Intractability: A Guide to the Theory of NP-Completeness," W. H. Freeman and Company, New York, 1979.

[5]F. Glover and G. A. Kochenberger, "Handbook of Meta-heuristics," Kluwer Academic Publishers, 2003.

[6]D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. "Optimization by Simulated Annealing: an Experimental Evaluation; Part I, Graph Partitioning," *Operations Research*, vol. 37, issue 6 (Nov.-Dec.), 1989, pp. 865-892.

[7]Maheswara Kasinadhuni. "Solving Optimization Problems Using Genetic Algorithms with Multiple Genome Coding," DPS dissertation, CSIS, Pace University, May 2004.

[8]Lixin Tao. "Research Incubator: Combinatorial Optimization," Technical Report #198 CSIS, Pace University, NY, http://csis.pace.edu/~lixin /dps (current May 2005).