Two heuristic stochastic algorithms applied to a temporally constrained scheduling program
by Scott Alan Furois

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science
Montana State University

Abstract:
The temporally constrained scheduling problem requires that given a list of tasks, each having a window of availability and minimum and maximum run times, an optimal schedule (schedule having the most tasks, subject to priority concerns) be generated. Previous attempts have used deterministic enumerative algorithms and deterministic heuristic algorithms. This paper applies two heuristic stochastic algorithms to this problem.

The simulated annealing algorithm starts with a state and randomly generates changes that reduce a "potential" in the problem so that the states generated will approach an optimum. The addition of a "temperature" allows the states generated to occasionally go "uphill" to avoid falling into local minima.

The genetic algorithm maintains a large number of potential solutions and uses "crossovers" to combine good features of two schedules. "Mutations" are also allowed to increase diversity of solutions and to allow avoidance of local minima.

Both simulated annealing and genetic algorithms are heuristic stochastic algorithms that have show promise on NP-complete problems like the temporally constrained scheduling problem. For this instance, the genetic algorithm outperformed all the previous attempts at scheduling, while the simulated annealing algorithm performed as well as many of the previous algorithms applied to the problem. This was determined by considering the results of a large data set with 3227 tasks. This result suggests that the solution space of the temporally constrained scheduling problem is more amenable to global search algorithms than local ones.

TWO HEURISTIC STOCHASTIC ALGORITHMS APPLIED TO A TEMPORALLY CONSTRAINED SCHEDULING PROBLEM

By

Scott Alan Furois

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Masters of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2001

APPROVAL

Of a thesis submitted by

Scott Alan Furois

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Dr. John Paxton _____ 5/8/01
(Signature) Date

Approved for the Department of Computer Science

Dr. J. Denbigh Starkey _____ 5/8/01
(Signature) Date

Approved for the College of Graduate Studies

Dr. Bruce McLeod _____ 5-9-01
(Signature) Date

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in part may be granted only by the copyright holder.

Signature Scott Z

Date 5/9/01

ACKNOWLEDGEMENT

Many thanks to Raytheon Corporation, which provided the problem and funding that led to this thesis. In addition, thanks to Dr. John Paxton for suggesting to Raytheon that I would be a good person to investigate this problem.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES - CONTINUED

Figure Page

ABSTRACT

The temporally constrained scheduling problem requires that given a list of tasks, each having a window of availability and minimum and maximum run times, an optimal schedule (schedule having the most tasks, subject to priority concerns) be generated. Previous attempts have used deterministic enumerative algorithms and deterministic heuristic algorithms. This paper applies two heuristic stochastic algorithms to this problem.

The simulated annealing algorithm starts with a state and randomly generates changes that reduce a "potential" in the problem so that the states generated will approach an optimum. The addition of a "temperature" allows the states generated to occasionally go "uphill" to avoid falling into local minima.

The genetic algorithm maintains a large number of potential solutions and uses "crossovers" to combine good features of two schedules. "Mutations" are also allowed to increase diversity of solutions and to allow avoidance of local minima.

Both simulated annealing and genetic algorithms are heuristic stochastic algorithms that have show promise on NP-complete problems like the temporally constrained scheduling problem. For this instance, the genetic algorithm outperformed all the previous attempts at scheduling, while the simulated annealing algorithm performed as well as many of the previous algorithms applied to the problem. This was determined by considering the results of a large data set with 3227 tasks. This result suggests that the solution space of the temporally constrained scheduling problem is more amenable to global search algorithms than local ones.

CHAPTER 1

INTRODUCTION

NP-complete problems such as the scheduling problem are a major focus of computer science research, both in theory and in practical arenas. NP-complete problems have interesting theoretical significance, and also are common in many realistic areas of industrial problems. There are three main types of approaches to "solving" practical examples of these NP-complete problems. Enumerative (or partially enumerative) algorithms attempt to generate all (or a significant subset) of the solutions to a problem to find the optimal solution. The most commonly used enumerative algorithm is the branch-and-bound algorithm. These are generally not useful for NP-complete problems except for very small or restricted cases, because they can require too much time or space to be practical. Approximation algorithms use "performance guarantees" to find solutions within a specified "distance" of the optimal solution. Different NP-complete problems are more or less amenable to these "performance guarantees" [17]. Heuristic algorithms use "rules of thumb" to generate good solutions more quickly, but a solution generated by a heuristic algorithm doesn't give any information about how close it is to the optimal solution.

Heuristic algorithms are divided into two main groups. Deterministic algorithms use only "rules of thumb" to generate a single solution for each data set. Many "rules of thumb" have been discovered for scheduling problems, but each rule can be shown to lead to suboptimal solutions in some cases. Several examples of these "rules of thumb"

are "scheduling shortest tasks first", "scheduling tasks with the least 'slack' time first", and "scheduling tasks that require the least resources first". Stochastic algorithms incorporate random choices as well, in order to expand the space they can search and reduce the chance of being "stuck" at a suboptimal solution. Both of the algorithms described in this paper are stochastic.

The Scheduling Problem

The scheduling problem can be formulated in the following manner. We are given a set of tasks T with associated constraints or resources required, and a partial ordering over the set of tasks that specify any tasks that require other tasks to have been done previously. We wish to maximize the number of tasks scheduled without violating any constraints.

The four main additional constraints found in the scheduling problem are physical, availability, precedence, and temporal constraints. Physical constraints are constraints inherent in the physical world, such as requiring that a task must take place at a certain location. Availability constraints stop tasks from being performed when resources are not present. For instance, it is not good to attempt to build a hull plate for a ship when the steel necessary is not yet present. Precedence constraints require some tasks to be performed before other tasks. For example, a hull plate must be built before it can be welded into place on a ship hull. Temporal constraints control when tasks can be performed. If a task involves a satellite transmitting data to a ground station, this task must be performed while the satellite is above the horizon with respect to the ground

station. Most variations of the scheduling problem are NP-hard, including the temporally constrained single processor scheduling problem, based on a transformation from 3-PARTITION [5]. This problem is listed in [5] as "Sequencing with release times and deadlines."

Scheduling Application Studied

The Air Force Satellite Control Network is a network of ground antennae that are used to gather information from satellite systems that do not have their own ground antenna stations or require additional support. The ground network consists of eight stations, each with one or more antennae. In an effort to better schedule the tasks from the incoming satellites, several approaches have been examined. Previous attempts to create schedules for this network have used techniques such as linear programming or dynamic programming [Raytheon Corp. communications] . The purpose of this research is to see if stochastic heuristics could generate better solutions for the problem, especially on larger data sets.

Tables 1.1 shows some sample data from the satellite file. This file shows the name of each satellite and its priority (1, 2, or 3, with 1 being the highest). Each task request (see Table 1.2) represents a connection between a satellite and a specified ground antenna. The tasks have limited time windows of availability. The tasks also have specified minimum and maximum times of connection. The tasks are divided into three different priorities, where a single high-priority task is more important than any number of lower-priority tasks. The object of the search is to find a valid schedule that

maximizes the number of high priority tasks. Maximizing the lower-priority tasks is secondary. The test data for this experiment consists of two files containing data requests and satellite data. The first data set consists of 1071 tasks, the second of 3229 tasks.

Table 1.1: Sample Data File

Satellite	Priority
APEX-1	2
COBE	3
DMSP_2-03	3

Table 1.2: Sample Task File

	Task Requests			Min Time	Max Time				
Task number	Satellite	Station	Antenna	(mi)	(mi)	Start of window		End of window	
1	APEX-1	BOSS	B	8	13	12/1/02	18:57:00	12/1/02	19:11:00
2	APEX-1	BOSS	B	9	14	12/2/02	19:40:00	12/2/02	19:54:00

This is an example of off-line scheduling, since the times of all the tasks are provided at the start. There are physical constraints [18] that limit which antenna can process a specific task. The availability windows represent temporal constraints, as do the requirements of minimum and maximum times of connection. The priorities represent precedence constraints. The tasks cannot be split apart, and a task must run for its minimum time allotment before another task can start.

Specific Scheduling Problem

For this problem, let $T = \{t_i\}$ be the set of tasks allocated to a single antenna. Each task t_i has a minimum length min_i, a maximum length max_i, a release time $start_i$ and a deadline end_i. A valid schedule is a schedule where all tasks scheduled are scheduled

for a time between the respective min and max time, every task is wholly scheduled within its availability window ($start_i$, end_i), and no task's scheduled time overlaps with another task's scheduled time. The problem is to find the valid schedule that maximizes the number of high-priority tasks scheduled, with secondary concern to maximizing lower priority tasks.

CHAPTER 2

SIMULATED ANNEALING

Description

The simulated annealing (SA) algorithm is well studied. It has been applied successfully to a wide variety of problem domains including k-partitioning and TSP (the traveling salesperson problem). It was first described in 1983[11].

The name "simulated annealing" comes from the analogy of the algorithm to the physical action of molecules in a solid (such as steel) as the solid is cooled from a liquid state. The molecules in general change configuration into lower-energy states, but can occasionally make leaps to higher-energy states depending on the temperature that the solid is at. The annealing process cools this solid slowly to minimize the number of defects that occur in the solid when it is fully cooled. A quick cooling introduces defects that can make the solid less useful. [12]

The SA algorithm as used on combinatorial optimization problems contains some of these concepts. There is a start state, a set of transitions, a measure of the energy of the system, and a temperature. The SA algorithm is a gradient descent method, with the simple addition of the ability to go "uphill" occasionally, with a probability to go "uphill" dependent upon the "temperature" of the solution generated. [7]

A description of the generic SA algorithm follows[16].

Figure 2.1: Simulated Annealing Algorithm

Given a problem with solution space S and objective function f

1. Select an initial state s_0.

2. Select an initial temperature $t_0>0$
3. Select a temperature reduction function α
4. Repeat
    a. Repeat
        i. Randomly select s from the neighborhood of s_0
        ii. $\delta = f(s) - f(s_0)$
        iii. if $\delta < 0$
        iv. update solution s_0 to s
        v. else
        vi. generate random x in the range of (0,1);
        vii. if ($x < \exp(-\delta/t)$ then update solution s_0 to s
    b. Until number of iterations == chosen value
    c. Set $t = \alpha(t)$
5. Until (stopping condition is true)
6. s_0 is the approximation of the optimal solution

Implementation

In the satellite scheduling problem, the initial state $\underline{s_0}$ consists of all the available tasks for the antenna being examined, each scheduled at some random point within its availability window for the minimum time required. The objective function $\underline{f}$ is the total amount of overlap in the schedule. The temperature $\underline{t}$ is a run-time parameter. The cooling function $\alpha(t) = \underline{a} * \underline{t}$, where $\underline{a}$ is a run-time parameter between 0 and 1. Typical values of $\underline{a}$ range from 0.8 to 0.99[2]. Values smaller than 0.8 generally converge too quickly to provide the minima-avoiding effect that is desired. If $\underline{t}$ is set to 0, the algorithm becomes a simple randomized hill-climbing algorithm

To select the next state from the neighborhood of the current state s_0, the task currently scheduled that has the largest amount of overlap (subject to a penalty term) is found. Then this task is randomly shifted within its availability window. The number of times this shifting is done is a controllable parameter. In this problem domain, maximizing the number of high-priority tasks is more important than maximizing the

total number of tasks, so the total overlap is multiplied by the priority of the given task before being compared to the "largest amount of overlap". Without the penalty term, the number of tasks scheduled remains roughly the same, but the number of high priority tasks scheduled decreases from the range of 510 to 540 to a new range of 370 to 390, a very significant decrease. The number of priority 3 tasks that are scheduled increases slightly more than this decrease (from 300 to 500).

Because the initial state s_0 is typically overscheduled, an additional step is taken. When the temperature is lowered, the task with the "largest amount of overlap" as defined above is removed from the schedule.

In this implementation, the states all represent invalid "solutions" until the objective function $f(s) = 0$, because overlapping tasks are not allowed in this scheduling problem. Once the overlap reached 0, the algorithm terminates and reports the current solution, after attempting to re-insert all the tasks that were previously removed without adding additional overlap.

CHAPTER 3

GENETIC ALGORITHM

The genetic algorithm (GA) is a very simple and powerful algorithm. Originally designed in the 1970's, it has only received notice recently with the advent of more powerful computers. The name is based on a metaphor, like the simulated annealing algorithm previously discussed. In this case, the metaphor is Darwinian evolution.

In a genetic algorithm, a population of candidate solutions is generated. Each of these candidates is assigned a fitness value based on how good the solution is. In the scheduling problem, several fitness functions are available, such as "number of tasks scheduled" or "lowest amount of idle time". Then these solutions are all allowed to undergo "crossover" and "mutation" operations to generate a new population of candidates. This process continues until some stopping criteria are met. Typical stopping criteria are lack of change over the course of several iterations or some specified number of generations being reached.

Much care needs to be taken both in the representation of the candidates and in the design of the operators. It is important that operators not generate invalid solutions, and also important to allow freedom in the crossover and mutation operators so that the space can be searched very effectively.

Specific Algorithm

The genetic algorithm used is a very simple one. Each member of the population represents a complete and valid schedule. Tasks are scheduled for the minimum time required and randomly given a starting time within the allowed window. The tasks are divided up by priority, and then inserted in a random order with the highest priority tasks being scheduled first. Tasks are only successfully inserted into the schedule if there is no overlap with currently scheduled tasks. The resulting schedules are in time order.

The fitness of each schedule is determined by the number of priority one tasks that it contains. Ties are broken by comparing the number of lower priority tasks scheduled. Due to the specifications of the problem, a schedule with a larger number of lower priority tasks scheduled is judged to be less fit than a task with a relatively small number of additional high-priority tasks. Parents for the operators are chosen by a weighted fitness metric. The following two schedules have been chosen to illustrate the operators.

Figure 3.1 Sample Schedules

Parent 1	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6	
Start & End	0	4	6	10	15	23	30	40	41	45	45	49

Parent 2	Task 1		Task 2		Task 3		Task 4		Task 6		Task 7	
Start & End	1	6	6	12	13	21	22	32	45	49	50	55

Two different versions of the simple crossover operator are used. The first is a generic simple crossover (C1), which combines the first n tasks of one parent schedule and the remaining positions (starting with position n+1) from a second parent schedule. This generates two new schedules. For the example below, crossover was performed

with n equal to four. Notice that the two children schedules are the same length as the parents. If the tasks had overlapped some, one of the child schedules would have been shorter than the parents.

Figure 3.2: Children from Operator C1

Child 1 (C1)	Task 1		Task 2		Task 3		Task 4		Task 6		Task 7	
Start & End	0	4	6	10	15	23	30	40	45	49	50	55

Child 2 (C1)	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6	
Start & End	1	6	6	12	13	21	22	32	41	45	45	49

The second crossover operation (C2) chooses a time somewhere within the bounds of the two schedules, taking the elements before this time from one parent schedule and the elements after this time from the other parent. In Figure 3.3, crossover was performed at time 45 on the original two parents. The first five tasks come from parent one and the final two tasks come from parent two. Notice that the resulting child schedule contains more tasks than either of its two parents.

Figure 3.3: Child from operator C2

Child 1 (C2)	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6		Task 7	
Start & End	0	4	6	10	15	23	30	40	41	45	45	49	50	55

A blend crossover is also implemented. In this crossover (C3), the two parents are "merged" by treating the schedules as a pair of queues and attempting to insert tasks in time-order from both parents. If both tasks start at the same time, the task from the second parent is inserted first, although there are many choices available for breaking this tie. This operator was implemented in the hope of removing wide areas of unused space.

This crossover generates only one child. Notice again that the resulting child schedule shown in Figure 3.4 includes more tasks than either of its parents.

Figure 3.4: Child from Blend crossover C3

Child (C3)	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6		Task 7	
Start & End	0	4	6	12	13	21	22	32	41	45	45	49	50	55

Three different mutation operators have been implemented. One operator (M1) deletes a task from a schedule. Another operator (M2) shifts a task within the schedule either forward or backward in time until it either reaches the border of its availability window or reaches the boundary of the adjacent task. M2 was chosen to act this way in order to maximize the potential benefit it might have to consolidate tasks to maximize the size of the spaces left in the schedule for future task additions. The last operator (M3) takes a schedule and attempts to add any tasks not already in the schedule via random insertion (similar to the process described above to create the initial population). When a schedule is selected to undergo mutation, either M1 or M2 is performed (with equal probability), and then the resulting schedule has M3 performed upon it.

The following schedules are based on the schedule created in Figure 3.4. In Figure 3.5, the M1 operator has deleted the fifth task.

Figure 3.5: Illustration of Deletion (M1)

Child (M1)	Task 1		Task 2		Task 3		Task 4		Task 6		Task 7	
Start & End	0	4	6	12	13	21	22	32	45	49	50	55

In Figure 3.6, the third task has been shifted to an earlier time.

Figure 3.6: Illustration of Shifting (M2)

Child (M2)	Task 1		Task 2		Task 3		Task 4		Task 5		Task 6		Task 7	
Start & End	0	4	6	12	12	20	22	32	41	45	45	49	50	55

In Figure 3.7, the result from Figure 3.5 has had a new task inserted in from the data file.

Figure 3.7: Illustration of Addition Operator (M3)

Child 2 (M3)	Task 1		Task 2		Task 3		Task 4		Task 9		Task 6		Task 7	
Start & End	0	4	6	12	12	20	22	32	33	40	45	49	50	55

The population is replaced during each generation except for a small elite portion that is kept. The proportion of the population that is kept is a controllable parameter, as are the simple-to-blend crossover ratio, the mutation rate, the size of the initial population and the number of generations. These five parameters allow much leeway to "tune" the genetic algorithm.

CHAPTER 4

RESULTS AND CONCLUSIONS

Simulated Annealing Results

For the SA algorithm, my results show that neither the cooling rate nor the initial temperature has any significant effect on the final result of the algorithm. The number of shifts performed has a positive effect that diminishes. Unless otherwise noted, the data points on the graphs are averages of five trials on the larger of the two data sets.

Figure 4.1 shows the number of priority 1 tasks scheduled, while Figure 4.2 shows the total number of tasks scheduled. The lines represent varying thresholds at two different cooling rates. Figure 4.1 shows that there appears to be a slight downward trend, but the trend is not statistically significant. In Figure 4.2, the difference is much greater, but still not significant even to the 50% confidence level. But in all these cases, the solutions with no threshold appear better than the solutions that are achieved with the varying thresholds. The data below was taken with the third parameter (number of shift operations) held constant at 100. The thresholds 100; 1,000; 10,000; and 100,000 were chosen because in the larger data set, the largest amount of overlap in the initial schedule was slightly less than 1,000,000 seconds of overlap. The cooling rates chosen are the most common cooling rates used in simulated annealing algorithms, because the threshold decreases very fast if the cooling rate goes below 0.8. At a cooling rate of 0.99, the threshold is still greater than 0 after hundreds of generations for any initial threshold of greater than 100.

Figure 4. 1: Priority 1 Tasks Scheduled (SA)

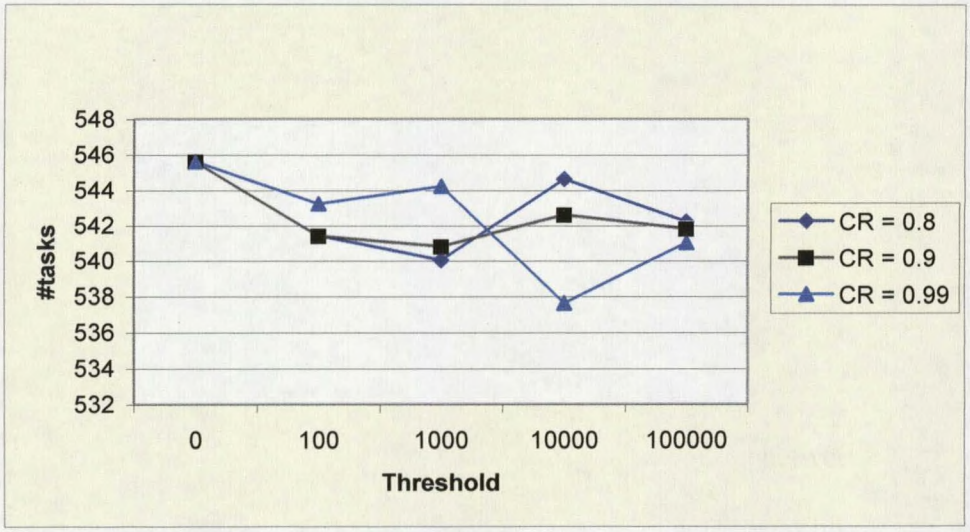
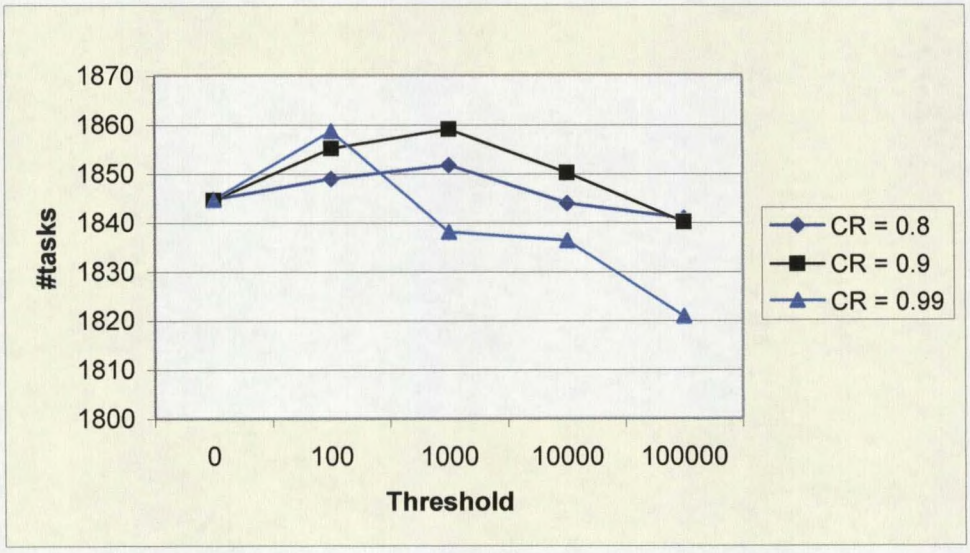


Figure 4. 2:Total Tasks Scheduled(SA)



On the other hand, the third parameter, the number of shifts between deletions, has a more noticeable effect. Figure 4.3 shows the resulting number of high-priority tasks scheduled as the number of shifts per iteration is varied from 0 to 100, with a