Genetic Algorithms for Complex, Real-Time Scheduling

David Montana, Marshall Brinn, Sean Moore, Garrett Bidwell
BBN Technologies / GTE Internetworking
10 Fawcett Street, Cambridge, MA 02138
{dmontana,mbrinn,smoore,gbidwell}@bbn.com

ABSTRACT

Real-time scheduling of large-scale problems in complex domains presents a number of difficulties for search and optimization techniques, including: (i) large and complex search spaces, (ii) dynamically changing problems, and (iii) a variety of problem-dependent constraints and preferences. Genetic algorithms are well suited to such problems due to their adaptability and their effectiveness at searching large spaces. We have used genetic algorithms to solve real-world problems in areas such as field service scheduling, air crew scheduling, and transportation scheduling. In this paper, we discuss key aspects of our approach including: (i) domain-specific chromosome representation and genetic operators, (ii) multi-objective evaluation function, (iii) heuristic initialization of the population, (iii) dynamic rescheduling, and (iv) cooperative interaction with human operators.

1. REAL-TIME SCHEDULING ISSUES

Most large-scale, real-time optimal scheduling problems cannot in practice be solved by traditional operations research (OR) techniques such as branch-and-bound and integer programming. Some of the issues that make real-time scheduling difficult are:

Large and complex search spaces: A generic scheduling problem involves assigning each of N tasks to one of M resources with a particular ordering of tasks at each resource. For this problem, the number of possible schedules is $\left(\begin{array}{c} N+M-1 \\ M-1 \end{array} \right) N!$, which implies superexponential growth as a function of the number of tasks and resources. In addition to its large size, this search space has a more complex topology than a Euclidean space or, when $M > 1$, a space of permutations.

Dynamically changing problems: By real-time scheduling we mean that the schedule is continually updated in response to events that change the problem definition. For example, a new high-priority task may arrive that needs to be scheduled for immediate execution and hence throws off much of the current schedule. Another example is that resources may malfunction, not only causing delays but also changing the timing of events, and hence necessitating a large-scale change to the schedule. We have introduced yet another source of change by allowing the human operators to override the automatic scheduler; the automatic scheduler then needs to work around what the human operator has mandated. Most automatic schedulers are not designed to react in real time to changes in the problem.

A variety of constraints: There are two types of constraints in scheduling problems. Hard constraints are those which must be satisfied for the schedule to be considered legal, while soft constraints are essentially preferences. An example of a hard constraint is when a certain task can only be performed by a particular resource. An example of a soft constraint is when it would be best but not essential to perform a certain task some time within the next two hours. The hard constraints essentially act to limit the search space, while the soft constraints help define the evaluation function. These constraints vary greatly across problem domains, and any approach that is general enough to handle the full range of different scheduling problems with their different constraints must be highly adaptable.

2. GENETIC ALGORITHMS

Genetic algorithms are an approach to optimization and learning based loosely on principles of biological evolution. Genetic algorithms maintain a population of possible solutions to a problem, encoded as chromosomes based on a particular representation scheme. After generating an initial population, new individuals for this population are generated via the process of reproduction. Parents are randomly selected from the current population for reproduction with the better ones (according to the evaluation criteria) more likely to be selected. The genetic operators of mutation and crossover generate children (i.e., new individuals) by random changes to a single parent or combining the information from two parents respectively.

Genetic algorithms have been applied to scheduling

problems in a wide variety of domains including (with a more complete set of references given in [9])

- jobshop/flowshop scheduling [1,2]
- urban transit systems
- supply chain management
- exam timetabling [3]
- scheduling computing tasks [5]
- scheduling laboratory equipment
- crew scheduling
- maintenance/rehabilitation scheduling
- talent/project scheduling

The reason for genetic algorithms' success at a wide and ever growing range of scheduling problems is a combination of power and flexibility. The power derives from the empirically proven ability of evolutionary algorithms to efficiently find globally competetive optima in large and complex search spaces. The favorable scaling of evolutionary algorithms as a function of the dimension of the search space makes them particularly effective in comparison with other search algorithms for the large search spaces typical of real-world scheduling.

The flexibility of genetic algorithms has multiple facets. Even the "standard" genetic algorithm (i.e., bit string representation with traditional crossover and mutation operators) can effectively handle problems that many traditional optimization algorithms cannot including: (i) discrete spaces, (ii) nonlinear, discontinuous evaluation functions, and (iii) nonlinear, discontinuous constraints. The use of "non-standard" genetic algorithms and the tailoring of representation, operators, initialization method, etc. to fit the problem/domain greatly increases the range of problems to which genetic algorithms can be effectively applied.

3. OUR APPROACH

We have developed a general approach to scheduling using genetic algorithms that allows us to relatively easily and quickly produce solutions for different domains. We now discuss the major components of this approach.

Domain-Specific Representation and Operators: Which representation (i.e. encoding of schedules as chromosomes) and genetic operators are best depends greatly on the requirements and constraints of the problem to be solved. For example, if all the tasks are assigned to a single resource and the times required for each task are known once the order of the tasks is known, then an order-based chromosome (i.e., chromosome that is a permutation of the integers 1 through N) is the natural one [4,6]. However, if the ordering of the tasks is predetermined and the data to optimize are the time separations or assignments to resources, then a string-based chromosome is the natural one. When both the ordering and additional information, such as the assignments, need to be optimized, more complicated representations are required, such as the one described in Section 4. In addition, genetic operators need to be defined that operate on the particular representation and, if possible, are matched to the problem domain.

Multi-Objective Evaluation Function: There generally are multiple evaluation criteria. We therefore utilize an evaluation function that is a linear combination of the different individual criteria, with the weights on the criteria being adjustable to allow different tradeoffs. The criteria can be of any form and are often nonlinear and discontinuous.

Heuristic Initialization: Grefenstette [7] has shown that using domain knowledge to initialize the population with better than random individuals can potentially greatly increase the performance of the genetic algorithm. Burke, Newall and Weare [3] have shown that key to getting optimal performance from the genetic algorithm is making the individuals in the initial population as good as possible while still maintaining maximal diversity. Since scheduling is a computationally difficult problem, it has been necessary for us to utilize heuristic initialization that maintains diversity to get good performance.

Dynamic Rescheduling: Dynamic rescheduling occurs by continuously cycling through two phases. In the execution phase, the scheduler is working on creating an optimal schedule for a snapshot of the data as it appeared at the beginning of the phase. During this phase, the data can change due to changes in the environment and due to inputs from the human operators (see below). In the reconciliation phase, the data is locked to temporarily prevent changes from the environment or human operators. The scheduler changes its schedule to reflect the changes to the data that occurred during the execution phase (using a heuristic algorithm and possibly a short run of the genetic algorithm to reoptimize) and commits the new schedule. The transition from the reconciliation phase to the execution phase includes a reinitialization of the population keeping only the best individual from the previous cycle and using heuristic initialization to generate the rest [5,2]. A big benefit to this process is that periodically diversity is reintroduced into the genetic algorithm. As a genetic algorithm runs, diversity decreases and with it the capability of the genetic algorithm to perform global search. Continually reintroducing diversity enables the genetic algorithm to get beyond any local optima into which it may settle.

Shared Control: Very few organizations would be

willing to grant full control of their daily operations to a computer. There need to be humans in the loop, overriding the computer's decisions where necessary, usually due to the computer lacking full information. This mode of operation where humans and computers both contribute to decisions is known as *shared control* [10]. Shared control raises a variety of issues including (i) potential contention between the scheduler and human operators and (ii) communication between the scheduler and human operators. Dynamic rescheduling deals with the issue of contention; the human operators have the ability to manipulate the schedule any time except during the reconciliation procedure, at which point the automated scheduler considers all changes made by the operators before writing its latest schedule. Communication from the human to the scheduler occurs via the process of freezing an assignment; humans have the ability to lock a task to a certain resource or at a certain time without the scheduler being able to override these decisions. Communication from the scheduler to the human occurs via alerts; the scheduler generates an alert for each event causing a sufficiently large penalty in the evaluation function, and a human operator must examine each alert and either choose to ignore it or handle it.

We illustrate our approach using two of the problem domains to which we have applied our genetic scheduling technology. We concentrate on the field service domain, which is the only one for which we have developed an operational system, as opposed to demonstrations.

4. PROBLEM 1: FIELD SERVICE SCHEDULING

Field service scheduling is essentially the problem of assigning service calls (i.e., tasks) to field engineers (i.e., resources) in a particular order or at particular times. Traditional approaches to automated field service scheduling are based on simple heuristic dispatch rules [8]. Our approach is superior due to the ability of the genetic algorithm to find competetive global optima (rather than the greedy approach of the dispatch rules) and due to the use of a more complicated and representative evaluation function.

The following toy sample problem will serve as the basis for the examples in this section. The field engineers (i.e., resources) are:

	Avaliable Work Hours
FE1	0800-1700
FE2	1300-1700
FE3	0800-1400

The calls (i.e., tasks) to be scheduled are:

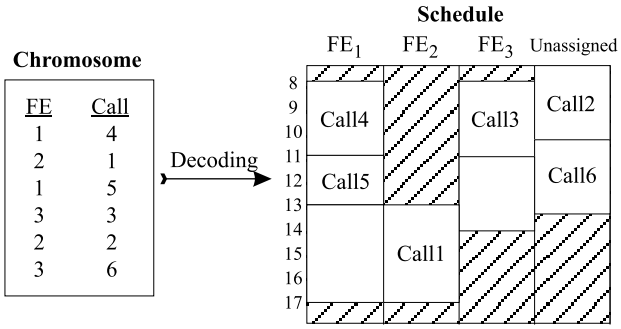

Figure 1: Decoding a chromosome into a schedule.

	Target Time	Duration	Location
Call1	1700	4	Loc1
Call2	1700	3	Loc1
Call3	1300	3	Loc1
Call4	1700	3	Loc2
Call5	0800	2	Loc2
Call6	1700	3	Loc2

The travel time between locations Loc1 and Loc2 is one hour, and travel time is zero between calls at the same location. Unlike in our actual system, the toy problem ignores issues of parts ordering, skills, lunch, customer availability, and call urgency.

Representation

We use what we call an "ordered-paired" representation, which was introduced by Bagchi et al. [1]. As we will see, it is a natural representation for this problem because it minimally captures the essential information, the assignment of calls to field engineers and ordering of the calls. Chromosomes are structured as follows:

Given a list of n field engineers $\{FE_1, ..., FE_n\}$ and m calls $\{C_1, ..., C_m\}$, a chromosome is an ordered list of pairs of indices into these lists: $\{\{f_1, c_1\}, \{f_2, c_2\}, ..., \{f_m, c_m\}\}$ where $0 \leq f_i < n$ and $1 \leq c_i < m$, where 0 indicates an association with no field engineer (i.e., an unscheduled call), and where each c_i is unique (and hence each call is represented exactly once).

Turning a chromosome into a schedule (i.e., decoding) is done as follows. For each $\{f_i, c_i\}$ taken in sequence from the chromosome, assign call C_{C_i} to field engineer FE_{f_i} at the earliest feasible time. To compute this time, one must consider the expected duration of the field engineer's previous call, the travel time from the previous call to this call, the availabilities of the field engineer and the customer, and the need for the field engineer to eat lunch at some reasonable time. If no such time exists within a specified scheduling window, put the call in the unscheduled bin. Figure 1

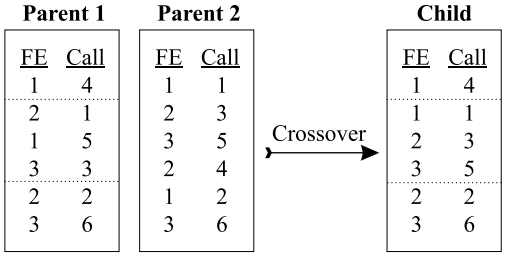

Figure 2: Two Point Crossover.

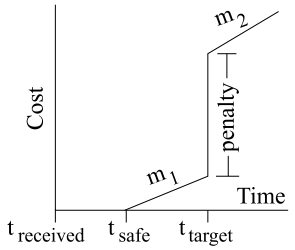

Figure 3: Cost for a call as a function of time.

illustrates this for the sample problem.

Turning a schedule into a chromosome (i.e., encoding) is performed as follows. Generate a FE/Call index pair for each assigned call. For each unassigned call generate a $\{0, Call\}$ entry. Sort all entries by actual scheduled time with unscheduled calls sorting to the end.

With this representation there are generally multiple chromosome representations for each "tightly packed" schedule. For example, the chromosome $\{\{1,4\},\{2,1\},\{1,5\},\{3,3\},\{2,2\},\{3,6\}\}$ represents the same schedule as $\{\{2,1\},\{1,4\},\{3,3\},\{1,5\},\{3,6\},\{2,2\}\}$. Every time we generate a new chromosome, before inserting it into the population we decode it into a schedule and then encode back into a potentially different chromosome. This process allows us to check the uniqueness of the chromosomes and not allow multiple representations of the same schedule in the population. It also enables better performance by the operators with the tasks always ordered by time.

Operators

We use the following mix of genetic operators:

Two Point Crossover (40%) picks two random points from the ordering of the first parent. All tasks between these two points are reordered according to the order of the second parent and receive their resource assignments from the second parent. This is illustrated in Figure 2.

Even Crossover (20%) is a special case of Two Point Crossover, with the constraint that exactly half the chromosome is taken from each parent.

Assignment Mutation (20%) takes a random task and assigns it to a new resource selected randomly from those capable of performing the task.

Reverse Order Mutation (20%) takes two random points on a chromosome and reverses the ordering of all tasks between the two positions that are assigned to the same resource as the task in the first position. (The purpose is to "uncross" routes.)

Evaluation Function

Field service providers want to give good service to their customers while minimizing their costs. We reflect these competing concerns in the following criteria, which are the individual components of our evaluation criterion:

- **Missed Target Cost** is the sum over all calls of the cost of scheduling a call after its target time or dangerously close to its target time. Figure 3 shows this cost as a function of time.
- **Travel Cost** is the sum over all field engineers of the time they spend traveling.
- **Slack Cost** is the sum over all field engineers of the time spent idle within a certain window of time. Outside of this window, slack time is not penalized because of the high probability that new calls will arrive to fill this void.
- **Return Home Cost** is the sum over all field engineers of their travel costs from their last job to their "home" location. This is penalized separately from other travel costs as it is not on company time but is rather a job satisfaction issue.
- **Parts Order Cost** is the sum of the extra cost of ordering parts for a call when parts are available in a different field engineer's on-hand inventory.
- **Unscheduled Cost** is a sum of a fixed penalty for each call that is not scheduled.
- **Skills Mismatch Cost** is the sum over calls of the penalty of assigning a field engineer with an adequate but not perfect skills match.

Initialization

As discussed in Section 3, we initialize the population with the outputs of a greedy optimization algorithm to improve performance. To maintain the diversity of the initial population, we have used a set of different greedy optimization algorithms.

Task Greedy Seeding - This algorithm orders all calls (tasks) by target time and allocates a resource to each task in order according to minimal cost. This algorithm is used to produce a single chromosome in

the initial population.

Fair Resource Greedy Seeding - Given an ordering of the resources, this algorithm sequences through the resources and allows each to pick its lowest cost next task, proceeding in a round-robin fashion. This algorithm is used to produce m chromosomes, where m is the number of resources, generated by m different orderings of the resources.

Unfair Resource Greedy Seeding - Given an ordering of the resources, this algorithms allows each resource to completely fill its schedule (up to a certain limit of tasks per resource) before allowing the next resource to grab tasks. This algorithm is used to produce m chromosomes.

Shuffled Task Greedy Seeding - This algorithm takes the seed produced by Task Greedy Seeding and randomly takes some number of tasks from the chromosome and places them back in random positions in the chromosome. This algorithm is used to generate all remaining required chromosomes left to fill out the initial population.

Dynamic Rescheduling

In addition to the new calls that continuously arrive, feedback from the field engineers on the progress of their calls causes a constant reassessment of the expected duration of these calls. The dynamic rescheduling cycle described in Section 3 is executed every ten minutes and adjusts the schedule based on the changes to the environment.

Shared Control

The different types of alerts that the scheduler generates are:

- Unscheduled: A call has been left unscheduled by the scheduler.
- Missed Target: A call has been scheduled so as to miss its contractual target time.
- Inadequate Parts : A call has been scheduled at a time when parts will not be available (usually due to a delayed shipment)
- Inadequate Data : A call record has come in with inadequate information to allow for scheduling.

5. PROBLEM 2: MILITARY LAND MOVE SCHEDULING

A second scheduling problem that we have solved using our approach is that of scheduling a military land move from a fort to a seaport. Items are loaded onto trucks to be driven to the port. (Some items are also transported by trains, but we have concentrated primarily on trucks.) The trucks are formed into convoys and travel together via civilian roads.

Hard constraints include not only weight and volume capacities for the trucks but also minimum and maximum sizes for the convoys. The time that each item is scheduled by the port to be loaded onto the ship is provided, and each item must be at the port by that time. The goals, or soft constraints, are (i) to minimize the amount of time that items sit at the port waiting to be loaded (i.e., minimize staging) and (ii) to minimize the disturbance to civilian traffic on the roads by not sending too many military vehicles on a given road within a given time span.

The four basic pieces of information we need to determine are: (i) how trucks are packed, (ii) how trucks are formed into convoys, (iii) what time each convoy leaves the fort, and (iv) what route each convoy travels. The packing of the trucks is done by a heuristic that knows that an item should be packed with other items that need to arrive at around the same time. The departure time of each convoy, once the convoy is formed, is determined by a simple heuristic that selects the latest possible time that still ensures that all items transported by the convoy arrive on time. Hence, only the grouping of trucks into convoys and the selection of routes is done by the genetic algorithm.

For convoy formation there is a natural time ordering of the trucks based on the earliest required arrival time of the items in that truck. Convoys should consist of trucks that are consecutive in this ordering, with the key information being where to draw the boundaries between convoys in this ordering.

For route selection, we have a set of predetermined routes cooptimize offline by a genetic algorithm attempting to maximize joint capacity. The genetic algorithm needs only select one of these routes for each convoy.

Representation

We use a string-based chromosome consisting of two portions. The first part encodes the mapping of trucks to convoys and is of length *numtrucks*. Each slot has an integer between 1 and *maxconvoys* indicating in which convoy that truck is. The second part encodes the mapping of convoys to routes and is of length *maxconvoys*. Each slot has a number between 1 and *numroutes* indicating which route that convoy travels.

For example, if there are nine trucks, three the maximum number of convoys, and four possible routes, then $(1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 4, 1)$ would indicate two convoys with five trucks in the first convoy, four trucks in the second convoy, the first convoy traveling route 2, and the second convoy traveling route 4 (and the route for the third potential convoy ignored because

only two convoys are created).

Operators

We use the following set of operators which respect the structure of the chromosome:

- **Convoy-Route Crossover (30%)** takes the convoys from the first parent and the routes from the second parent.
- **Convoy Mutation (30%)** loops over the boundaries between convoys and with a certain mutation probability moves the boundary a random amount in one direction or the other.
- **Route Mutation (20%):** loops over the route assignments and with a certain mutation probability chooses a new random route number.
- **Combined Mutation (20%):** performs both a convoy mutation and a route mutation.

Evaluation Function

There are two criteria in our evaluation function:

- **Staging Cost** is the sum over each item of the square of how long before its loading time the item arrives.
- **Link Overuse Cost** is the sum over each hour of each link in the routes of the square of the excess capacity utilized.

Initialization

While our initialization procedure does respect the structure of the chromosome, it does not currently utilize greedy heuristics to select initial members. Instead, for each new chromosome, the initialization procedure selects random sizes for the convoys between the minimum and maximum sizes and selects random routes for each convoy.

We have not yet done any work on dynamic rescheduling or shared control in this domain.

6. CONCLUSION

Our approach uses the flexibility and power of genetic algorithms to produce practical solutions to complex scheduling problems in a generally applicable manner. Our field service scheduler is currently operational at a commercial company scheduling field engineers, and we are working on incorporating dynamic rescheduling and shared control into the land move scheduler to move it from a demonstration to a prototype system. We are also building a prototype system for a new domain of military air crew scheduling.

We are currently doing experiments to document the performance of the genetic algorithm scheduling in a controlled setting. So far, the success of the system has been primarily documented in the field service domain by the large reduction in calls serviced late and the large increase in the number of calls closed per day by a field engineer, the two main metrics of the industry.

7. REFERENCES

[1] Bagchi, S., Uckun, S., Miyabe, Y., & Kawamura, K. (1991). Exploring Problem-Specific Recombination Operators for Job-Shop Scheduling. *Proc. of the Fourth Intl. Conf. on Genetic Algorithms,* pp. 10–17.

[2] Bierwirth, C., Kropfer, H., Mattfeld, D. & Rixen, I. (1995). Genetic Algorithm Based Scheduling in a Dynamic Manufacturing Environment. *IEEE Conf. on Evolutionary Computation,* pp. 439–443.

[3] Burke, E., Newall, J. & Weare, R. (1998). Initialisation Strategies and Diversity in Evolutionary Timetabling. To appear in *Evolutionary Computation,* 6(2).

[4] Goldberg, D. E. & Lingle, Jr., R. (1985). Alleles, Loci, and the Traveling Salesman Problem. *Proc. of the First Intl. Conf. on Genetic Algorithms,* pp. 154–159.

[5] Gonzalez, C., & Wainwright, R. (1994). Dynamic Scheduling of Computer Tasks Using Genetic Algorithms, *Proc. of the First Intl. Conf. on Evolutionary Computation*, pp. 829–833.

[6] Grefenstette, J. J., Gopal, R., Rosmaita, B. J. & van Gucht, D. (1985). Genetic Algorithms for the Traveling Salesman Problem. *Proc. of the First Intl. Conf. on Genetic Algorithms,* pp. 160–165.

[7] Grefenstette, J. J. (1987). Incorporating Problem Specific Knowledge in Genetic Algorithms. In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing,* pp. 42–60. Los Altos, CA: Morgan Kaufmann.

[8] Hill, A. (1993). An Experimental Comparison of Dispatching Rule for Field Service Support. *Decision Sciences*, 23(1), 235–249.

[9] Montana, D. J. (1998). Introduction to the Special Issue: Evolutionary Algorithms for Scheduling. To appear in *Evolutionary Computation,* 6(2).

[10] Sheridan, T. (1992). *Telerobotics, Automation, and Human Supervisory Control.* Cambridge, MA: MIT Press.