Chapter 10

RESOURCE SCHEDULING WITH PERMUTATION BASED REPRESENTATIONS: THREE APPLICATIONS

Darrell Whitley[1], Andrew Sutton[1], Adele Howe[1] and Laura Barbulescu[2]
[1]*Computer Science Department, Colorado State University, Fort Collins, CO 80523 USA*
[2]*Robotics Institute Carnegie Mellon University Pittsburgh, PA 15213 USA*

Abstract Resource based scheduling using permutation based representations is reviewed. Permutation based representations are used in conjunction with genetic algorithms and local search algorithms for solving three very different scheduling problems. First, the Coors warehouse scheduling problem involves finding a permutation of customer orders that minimizes the average time that customers' orders spend at the loading docks while at the same time minimizing the running average inventory. Second, scheduling the Air Force Satellite Control Network (AFSCN) involves scheduling customer requests for contact time with a satellite via a ground station, where slot times on a ground station is the limited resource. The third application is scheduling the tracking of objects in space using ground based radar systems. Both satellites and debris in space must be tracked on regular basis to maintain knowledge about the location and orbit of the object. The ground based radar system is the limited resource, but unlike AFSCN scheduling, this application involves significant uncertainty.

Keywords: resource scheduling, genetic algorithms, local search, permutations, representation

1. INTRODUCTION

The goal of resource scheduling is to allocate limited resources to requests during some period of time. A schedule may attempt to maximize the total number of requests that are filled, or the summed value of requests filled, or to optimize some other metric of resource utilization. Each request needs to be assigned a time window on some appropriate resource with suitable capabilities and capacity. Given the often large numbers of possible combinations of time slots and resources, a simple strategy is a greedy scheduler: allocate resources on a first-come-first-served basis by placing each request taken in some order on its best available resource at its best available time.

The problem with the simple greedy strategy is that requests are not independent – when one request is assigned a slot on a resource, that slot is not longer available. Thus, placing a single request at its optimal position may preclude the optimal placement of multiple other requests. A significant improvement then is to explore the space of possible permutations of requests where the permutation defines a priority ordering of the requests to be placed in the schedule.

Representing a schedule as a permutation offers several advantages over simply using a greedy scheduler alone or searching the schedule space directly. Permutations support a strong separation between the problem representation and the actual details of the particular scheduling application. This allows the use of relatively generic search operators that act on permutations and which are independent of the application. A more direct representation of the scheduling problem would require search operators that are customized to the application. When using a permutation based representation, this customization is hidden inside a greedy schedule builder.

Permutation representations do incur costs. First, the permutation is an indirect representation of the problem. Thus, a separate schedule builder must be constructed to map the permutation into an actual schedule for evaluation. How well the schedule builder exploits critical features of the problem may play a key role in how well the overall scheduling system works. Second, the permutation representation often introduces redundancies in the search space. Two different permutations may map to the same schedule. Assume request B is before request A in one permutation, but request A is before B in another permutation; otherwise the permutations are similar. If A and B do not compete for resources and otherwise do not interaction, the two permutations may map to exactly the same schedule. This would also mean that the two permutations also have the same evaluation. This redundancy can also contribute to the existence of plateaus, or connected regions with flat evaluation.

Permutation based representations were popularized because they support the application of genetic algorithms to resource scheduling problems. Whitley et al. (Whitley et al., 1989) first used a strict permutation based representation in conjunction with genetic algorithms for real world applications. However, Davis (Davis, 1985b) had previously used "an intermediary, encoded representation of schedules that is amenable to crossover operations, while employing a decoder that always yields legal solutions to the problem." This is also a strategy later adopted by Syswerda (Syswerda, 1991; Syswerda and Palmucci, 1991), which he enhanced by refining the set of available recombination operators.

The purpose of this chapter is to review how permutation representations have been used on three applications. The first example involves warehouse scheduling. The other two examples are from the related domains of satellite communication scheduling and radar tracking of objects (both satellites and debris) in space. In these last two examples, we examine the question of how well “heuristic search” works compared to optimal solutions obtained using exact methods. The problem with exact methods is that they can be costly. To make them work, one must sometimes decompose the problem. However, as will be shown, “heuristic” methods can yield superior results compared to “optimal exact methods.”

For the applications, we will examine several factors related to the permutation representation. We will show how well the permutation representation performs in the applications. We will indicate how the domain information has been separated from the representation and embedded in the schedule builder. In some cases, we will show evidence of effects of the representation on the search space.

1.1 The Genitor Genetic Algorithm

The experiments discussed in this paper use the Genitor (Whitley, 1989) “steady-state” genetic algorithm (Davis, 1991). In the Genitor algorithm, two parents mate and produce a single child. The child then replaces the worst, or least fit, member of the population. Using a population of size P, the best P-1 individuals found during the search are retained in the population.

In addition, Genitor allocates reproduction opportunities based on the rank of the individuals in the population. A linear bias is used such that individuals that are above the median fitness have a rank-fitness greater than one and those below the median fitness have a rank-fitness of less than one. Assuming the population is sorted from best to worst, we will say that an individual has *rank* i if it is the i^{th} individual in the sorted population. The selective pressure at i will be denoted $S(i)$ and corresponds to the number of times that the individual at rank i will be sampled in expectation over P samples under sampling with replacement. We will represent the overall selective pressure by $S = S(1)$, the selective pressure for the best individual in the population. For example, standard tournament selection has a linear selective pressure of $S = 2.0$ which implies the best individual in a population is sampled 2 times in expectation over P samples, while the median individual in the population is sampled 1 time, and the worst individual is sampled 0 times. Linear selective pressure can be implemented by a biased random number generator (see (Whitley, 1989)) or by using stochastic tournament selection. Stochastic tournament selection is implemented by comparing two individuals, but selecting the best with a probability greater than 0.5 but less than 1.0; if p_s is the probability of keeping the best individual, the selective pressure is then $2p_s$ (Goldberg, 1991).

1.1.1 Genetic Algorithm and Permutation Codings. Typically, simple genetic algorithms encode solutions using bit-strings, which enable the use of "standard" crossover operators such as one-point and two-point crossover (Goldberg, 1989). Some genetic algorithms also use real-valued representations (Davis, 1991).

When solutions for scheduling problems are encoded as permutations, a special crossover operator is required to ensure that the recombination of two parent permutations results in a child that (1) inherits good characteristics of both parents and (2) is still a permutation of the N task requests. Numerous crossover operators have been proposed for permutations representing scheduling problems.

Syswerda's (Syswerda, 1991) order crossover and position crossover differ from other permutation crossover operators such as Goldberg's PMX operator (Goldberg, 1985) or Davis' order crossover (Davis, 1985a) in that no contiguous block is directly passed to the offspring. Instead, several elements are randomly selected by absolute position. These operators are largely used for scheduling applications (e.g., (Syswerda, 1991; Watson et al., 1999; Syswerda and Palmucci, 1991) for Syswerda's operator) and are distinct from the permutation recombination operators that have been developed for the Traveling Salesman Problem (Nagata and Kobayashi, 1997; Whitley et al., 1989). Operators that work well for scheduling applications do not work well for the Traveling Salesman Problem, and operators that work well for the Traveling Salesman Problem do not work well for scheduling. Operators such as PMX and Cycle crossover represent early attempts to construct a general purpose permutation crossover operator; these have not been found to be well suited to scheduling applications.

Syswerda's order crossover operator can be seen as a generalization of Davis' order crossover (Davis, 1991) that also borrows from the concept of uniform crossover for bit strings. Syswerda's order crossover operator starts by selecting K uniform-random positions in Parent 2. The corresponding elements from Parent 2 are then located in Parent 1 and reordered so that they appear in the same relative order as they appear in Parent 2. Elements in Parent 1 that do not correspond to selected elements in Parent 2 are passed directly to the offspring.

```
           Parent 1:  "A B C D E F G"
           Parent 2:  "C F E B A D G"
  Selected Elements:     *   * *
```

For example, the selected elements in Parent 2 are F B and A in that order. A remapping operator reorders the relevant elements in Parent 1 in the same order found in Parent 2.

```
"A B _ _ _ F _"  remaps to  "F B _ _ _ A _"
```

The other elements in Parent 1 are untouched, thus yielding

```
"F B C D E A G"
```

Syswerda also defined a "position crossover". Whitley and Nam (Whitley and Yoo, 1995) prove that Syswerda's order crossover and position crossover are identical in expectation when order crossover selects K positions and position crossover selects L-K positions over permutations of length L. In effect, order crossover inherits by order first, then fills the remaining slots by position. Position crossover inherits by position first, then fills the remaining slots by their relative order.

Syswerda's contribution (Syswerda, 1991) was to emphasize that permutation-based recombination operators can preserve either the position, relative order or adjacency of the elements in the parents when extracting information from the parents to construct a child. But operators cannot do all three things well. In various applications, we have found Syswerda's order crossover operator to be robust across a wide range of resource scheduling applications. This makes intuitive sense, given that the relative order in which requests are filled affects the availability of resources for later requests.

2. THE COORS WAREHOUSE SCHEDULING PROBLEM

The Coors production facility (circa 1990) consists of 16 production lines, a number of loading docks, and a warehouse for product inventory. At the time this research was originally carried out (Starkweather et al., 1991), each production line could manufacture approximately 500 distinct products; use of different packaging constitutes a different product. The plant contained 39 truck and 19 rail-car docks for loading customer orders. Orders could be filled directly from the production lines or from inventory.

A solution is a priority ordering of customer orders, given the mix of products that make up that order. A weekly production line schedule already exists.

Orders are separated into truck and rail-car orders before scheduling begins. A customer order remains at a dock until it is completely filled, at which point the dock becomes empty and available for another order. Only orders using an equivalent transport compete for dock space; however, all orders compete for product from either the production line or inventory.

In the schedule builder, orders are taken from the permutation of customer orders; in effect, the permutation queues up the customer orders which then wait for a vacant loading dock. When a dock becomes free, an order is removed from the queue and assigned to the dock. Orders remain at a dock until they are completely filled with product either drawn from inventory or directly from one of the production lines. Product comes from the production lines organized into pallets. An order specifies a method of transport (a truck or rail-car) as well as a certain combination of product pallets. During a typical 24 hour period, approximately 150 to 200 orders are filled.

Simulation is used to evaluate the quality of a particular schedule in this domain. We used both a fast *internal* simulator and a much slower high-resolution *external* simulator. The internal simulator executes in less than 0.01 second and provides evaluation information used by the objective function. The external simulator requires several minutes to execute. Thus, the search is done using the internal simulator.

The external simulator models warehouse events in much greater detail and was used to validate and confirm the quality of the schedules found by the search algorithms.

While the differences between the internal simulator and external simulator are largely a matter of detail, the differences can be important. For example, when product is moved from inventory to a loading dock in the internal simulator, the amount of time needed to move the product is a constant based on the average time required to move product to a particular dock. However, in the external simulator, individual fork lifts are modeled. The external simulator determines when the fork lift becomes available, where it is located and the route it needs to take to move the product. Over time, small differences in the two simulations can accumulate. However, to the degree that good schedules found using the internal simulator tend to be good schedules under the external simulator, search using the internal simulator can be effective. (We have also looked at other ways to combine and exploit both simulators during search (Watson et al., 1999).)

Figure 10-1 illustrates how a permutation is mapped to a schedule. Customer orders are assigned a dock based on the order in which they appear in the permutation; the permutation in effect acts as a customer priority queue. In the right-hand side of the illustration note that initially customer orders **A** to **I** get first access to the docks (in a left to right order). **C** finishes first, and the next order, **J**, replaces **C** at the dock. Order **A** finishes next and is replaced by **K**. **G** finishes next and is replaced by **L**.

If two customer orders need the same product, the customer order that has been at dock the longest gets the product first. Product is drawn from inventory or production on-demand, but with a bias toward product from the production line if it is available within a time horizon that does not impact total time at dock.

All results in this paper are based on an actual manufacturing plant configuration provided to us by Coors. The production line schedule and initial inventory were provided, as well as 525 actual customer orders filled over a three day period.

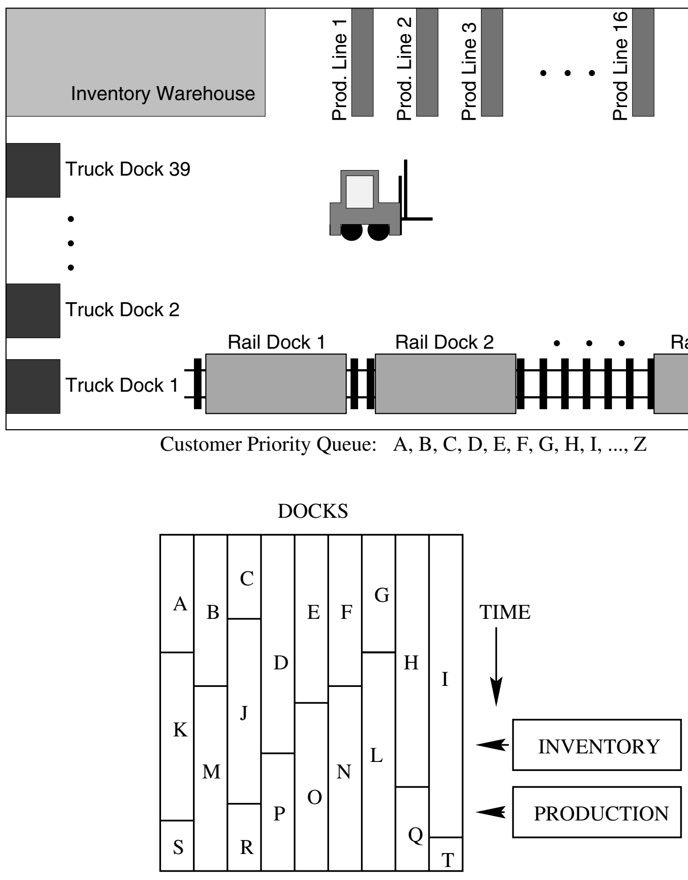

Figure 10-1. The warehouse model includes production lines, inventory, truck docks and rail docks. The columns in the schedule represent different docks. Customer orders are assigned to a dock left to right and product is drawn from inventory and the production lines.

2.1 The Coors Evaluation Function

For the Coors warehouse scheduling problem, we are interested in producing schedules that simultaneously achieve two goals. One of these is to minimize the mean time that customer orders remain at dock. Let N be the number of customer orders. Let M_i be the time that the truck or rail car holding customer order i spends at dock. Mean time at dock, $\mathcal{M}$, is then given by

$$\mathcal{M} = \frac{1}{N} \sum_{i=0}^{N} M_i.$$

The other goal is to minimize the running average inventory. Let F be the makespan of the schedule. Let $\mathcal{J}_t$ be inventory at time t. The running average inventory, I, is given by

$$I = \frac{1}{F}\sum_{t=0}^{F} \mathcal{J}_t.$$

Attempting to minimize either the mean time at dock or average inventory metrics independently can have a detrimental effect on the other metric. In this case, this multi-objective problem was transformed into a single-objective problem using a linear combination of the individual objectives proposed by Bresina (Bresina et al., 1995):

$$obj = \frac{(\mathcal{M} - \mu_{\mathcal{M}})}{\sigma_{\mathcal{M}}} + \frac{(I - \mu_I)}{\sigma_I} \tag{1}$$

where I represents running average inventory, $\mathcal{M}$ represents order mean time at dock, while μ and σ represent the respective means and standard deviations over a set of solutions.

2.2 Comparing Algorithm Performance

We have compared various algorithms to the results produced by the genetic algorithm for this problem (Watson et al., 1999). Here, we only report results for a stochastic hill-climber. The best results were given by an "exchange operator" (also referred to as a "swap operator"). This operator selects two random customers and then swaps their position in the permutation. We also report results using random sampling in conjunction with the greedy schedule builder. All of the algorithms reported here used 100,000 function evaluations. The Genitor algorithm used a population size of 500 and a selective pressure of 1.1.

For our test data, we have an actual customer order sequence developed and used by Coors personnel to fill customer orders. This solution produced an average inventory of 549817.25 product units and an order mean time at dock of 437.55 minutes. We consider a schedule *competitive* if these measures are less than or equal to those of the Coors solution. The first column of Table 10-1 illustrates that (mean) solutions obtained by random sampling are *not* competitive with the Coors solution. Random sampling in this situation does not imply random solutions; instead it involves the iterative application of the greedy schedule builder using 100,000 randomly generated permutations.

Results are presented in Table 10-1; we report mean performance and standard deviations over 30 runs. Statistical comparison of the competitive search algorithms indicates that both algorithms perform better than random sampling for both reducing inventory and time at dock, as verified by a series of two-tailed t-tests $(p < 0.0001)$.

	Random Sampling		Genetic Algorithm		Exchange Hill Climber		Coors
	Internal	External	Internal	External	Internal	External	Solution
			Mean Time-At-Dock				
μ	456.88	447.11	392.49	397.48	400.14	439.43	437.55
σ	3.2149	8.0304	0.2746	7.3680	4.7493	4.0479	n.a.
			Average Inventory				
μ	597123	621148	364080	389605	370458	433241	549817
σ	10041	28217	1715	9137	20674	20201	n.a.

Table 10-1. Performance Results on the Internal and External Simulator. The Coors Solution indicates a human generated solution used by Coors.

The final solution found using the internal simulator on each of 30 runs is also evaluated using the external simulator. Of course, the external simulator is expected to be a better predictor of actual performance. Comparing the actual events that occurred at Coors against the detailed external simulator, we find that Genitor was able to improve the mean time at dock by approximately 9 percent. It was the only scheduler that produced an improvement over what the human schedulers at Coors had done as measured by the external simulator. The big change however is in average inventory. Both Genitor and the local search methods show a marked reduction in average inventory. In the end, all of the schedules shipped exactly the same product; if two schedules finish shipping at the same time then the same amount of product must be left over. However, average inventory levels can be lower due to the combined effects of pulling strategic product from inventory early on and filling directly off of the production line. When time at dock is lower however, there is also an impact on inventory, since a longer overall schedule translates into a longer production period with the additional product going into inventory.

Overall, it is notable that the genetic algorithm produced results using the internal simulator that hold up rather well under the external simulator. The exchange hill climber did not fare as well under the external simulator. The genetic algorithm is the only scheduler with solutions for mean time at dock on the external simulator that improved on the human generated solution used by Coors.

One interesting question that could not be explored in this work was related to the long terms effects of less time at dock and lower average inventory. Are these gains sustainable over weeks and months as opposed to just 1 or 2 or 3 days? Could the warehouse be operated long-term with less overall inventory? These kinds of questions are not well addressed in the scheduling literature, and arise again in a later application in this paper.

Watson et al. (Watson et al., 1999) provide a more detailed discussion of the Coor's warehouse scheduling problem.

3. SCHEDULING THE AIR FORCE SATELLITE CONTROL NETWORK

Scheduling the Air Force Satellite Control Network (AFSCN) involves coordinating communications via ground stations to more than 100 satellites. Space-ground communications are performed using 16 antennas located at nine ground stations around the globe. Customers submit requests to reserve an antenna at a ground station for a specified time period based on the visibility of the target satellites. We will separate the satellites into two types. The low altitude satellites have short (15 minutes) visibility windows that basically only allow one communication per pass over a ground station. High altitude satellites have longer windows of visibility that may allow for multiple tasks to be scheduled during a single pass over the ground station.

A problem instance consists of n task requests. A task request T_i, $1 \leq i \leq n$, specifies both a required processing duration T_i^{Dur} and a time window T_i^{Win} within which the duration must be allocated; we denote the lower and upper bounds of the time window by $T_i^{Win}(LB)$ and $T_i^{Win}(UB)$, respectively. Tasks cannot be preempted once processing is initiated. Each task request T_i specifies a resource (antenna) $R_i \in [1..m]$, where m is the total number of resources available. The tasks do not include priorities.

T_i may optionally specify $j \geq 0$ additional (R_i, T_i^{Win}) pairs, each identifying a particular alternative resource (antenna) and time window for the task. While requests are made for a specific antenna, often a different antenna at the same ground station may serve as an alternate because it has the same capabilities.

There are (at least) two approaches to defining evaluation functions for optimizing the utilization of the ground stations. One approach is to minimize the number of request conflicts for AFSCN scheduling; in other words we maximize the number of requests that can be scheduled without conflict. Requests that cannot be scheduled without conflict are bumped out of the schedule. This is historically the objective function that has been used for this problem.

However, this is not what happens when humans carry out AFSCN scheduling. Satellites are valuable resources, and the AFSCN operators work to fit in every request. So an alternative is to schedule every request, but minimize the amount of overlap in the schedule. Under this approach, all of the requests are scheduled, but some requests get less than the requested amount of time. In this approach, the amount by which requests must be trimmed to fit in every request (i.e., the overlap) is minimized.

To assess the relative merits of different heuristic search techniques on permutation representations of AFSCN, we compare performance of three algorithms: Genitor, local search and random sampling.

A genetic algorithm for minimizing conflicts searches permutations of contact requests. Genitor's schedule builder considers requests in the order that they appear in the permutations. Each task request is assigned to the first available resource (from its list of alternatives) and at the earliest possible starting time. If the request cannot be scheduled on any of the alternative resources, it is dropped from the schedule (i.e., bumped). The evaluation of a schedule is then defined as the total number of requests that are scheduled (for maximization) or inversely, the number of requests bumped from the schedule (for minimization).

Local search for minimizing conflicts uses the *shift* operator because we found it to work well compared to several relatively standard alternatives. From a current solution π, a neighborhood is defined by considering all $(N-1)^2$ pairs (x, y) of task request ID positions in π, subject to the restriction that $y \neq x - 1$. The neighbor $\pi^{'}$ corresponding to the position pair (x, y) is produced by *shifting* the job at position x into the position y, while leaving the relative order of other jobs unchanged.

Given the large neighborhood size, we use the shift operator in conjunction with next-descent hill-climbing: the neighbors of the current solution are examined in a random order, and the first neighbor with either a lower or equal number of bumped tasks is accepted. Search is initiated from a random permutation and terminates when a pre-specified number of solution evaluations is exceeded.

Random sampling for minimizing conflicts produces schedules by generating a random permutation of the task request IDs and evaluating the resulting permutation using the schedule builder. Randomly sampling a large number of permutations provides information about the distribution of solutions in the search space, as well as a baseline measure of problem difficulty for heuristic algorithms.

3.1 Results for Minimizing Conflicts

Parish (Parish, 1994) first applied the Genitor algorithm to AFSCN scheduling using Syswerda's order crossover with positive results. Parish used data from 1992 when about 300 requests were being scheduled each day. In the following experiments we used five days of actual data[1] for the dates: 3/7/2002, 3/20/2002, 3/26/2003, 4/2/2003 and 5/2/2003. The number of requests received during a typical day is approximately 450 each day. Our experiments show the increased demand from 300 to 450 or more request each day results in

[1]We thank William Szary and Brian Bayless at Schriever Air Force Base for providing us with data.

		Genitor			Local Search			Random Sampling		
Day	Size	Min	Mean	S.D.	Min	Mean	S.D.	Min	Mean	S.D.
03/07/02	483	42	43.7	0.98	68	75.3	4.9	73	78.16	1.53
03/20/02	457	29	29.3	0.46	49	56.06	3.83	52	57.6	1.67
03/26/03	426	17	17.63	0.49	34	38.63	3.74	38	41.1	1.15
04/02/03	431	28	28.03	0.18	41	48.5	3.59	48	50.8	0.96
05/02/03	419	12	12.03	0.18	15	17.56	1.3	25	27.63	0.96

Table 10-2. Performance of Genitor, local search and random sampling in terms of the best and mean number of bumped requests (with standard deviation as S.D.). All statistics are taken over 30 independent runs, with 8000 evaluations per run.

significantly more difficult problems because the available time on the ground stations is increasingly oversubscribed.

In Table 10-2, we present the results obtained for these problems. Statistics were obtained over 30 runs, with 8000 evaluations per run. We have run the algorithms for 100,000 evaluations, but the results change very little, and 8000 evaluations would allow human operators to run the schedulers interactively (the schedulers take less than 10 seconds to execute.)

3.2 A Hybrid Method with Optimal Low Altitude Scheduling

In our investigations we were able to prove that it is possible to schedule contacts with low altitude satellites optimally (Barbulescu et al., 2004). This is because the window of visibility for low altitude satellites typically allows only one contact per pass. This restricts the number of ways that a schedule can be constructed. Under these conditions we constructed a proof showing that a variant of activity selection scheduling (Corman et al., 1990) on multiple resources is optimal.

Thus, an alternative approach to this problem is to schedule the low altitude requests first using an exact method, and then schedule the remaining high altitude requests using a genetic algorithm or local search. We refer to this approach as the *the split heuristic.* In effect, we break the problem into two parts, and we know that one of the parts is solved optimally.

As shown in Table 10-3 the split heuristic fails to find the best known schedules for two problems, using Genitor with 8000 evaluations. The results remain the same even when dramatically more evaluations are used per run. By examining the data, we identified situations in which the low altitude requests were blocking insertion of longer high altitude requests.

Figure 10-2 illustrates a situation for which scheduling low-altitude requests first results in suboptimal solutions. Assume there are two ground stations and

Day	Best Known	Genitor-Split		
		Min	Mean	Stdev
03/07/02	42	42	42	0
03/20/02	29	30	30	0
03/26/03	17	18	18	0
04/02/03	28	28	28	0
05/02/03	12	12	12	0

Table 10-3. Results of running Genitor with the split heuristic over 30 experiments, with 8000 evaluations per experiment.

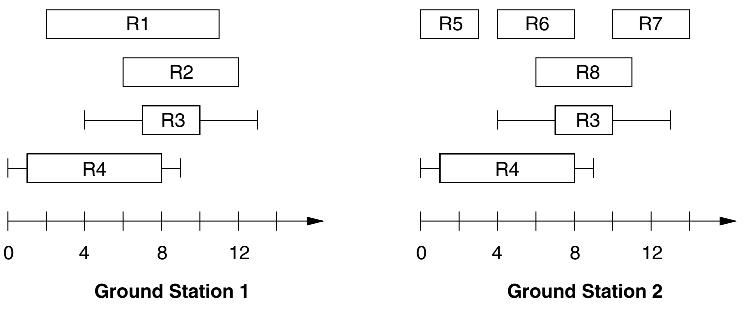

Figure 10-2. Example of a problem for which the split heuristic can not result in an optimal solution. Each ground station has two antennas; the only high-altitude requests are $R3$ and $R4$.

two resources (two antennas) at each ground station. Assume two high-altitude requests, $R3$ and $R4$, have durations three and seven, respectively. $R3$ can be scheduled between start time 4 and end time 13; $R4$ can be scheduled between 0 and 9. Both $R3$ and $R4$ can be scheduled at either of the two ground stations. The rest of the requests are low-altitude requests. $R1$ and $R2$ request the first ground station, while $R5$, $R6$, $R7$, and $R8$ request the second ground station.

If low-altitude requests are scheduled first, then $R1$ and $R2$ are scheduled on Ground Station 1 on the two resources, and the two high-altitude requests are bumped. Likewise, on Ground Station 2, the low-altitude requests are scheduled on the two resources, and the high-altitude requests are bumped. By scheduling low-altitude requests first, the two high-altitude requests are bumped. However, it is possible to schedule both of the high-altitude requests such that only one request ($R1$, $R2$ or $R8$) gets bumped. Therefore, an optimal solution is not possible when all of the low-altitude requests are scheduled before the high-altitude requests.

3.3 Minimizing Overlap

So far we have looked at the traditional objective of minimizing dropped requests (i.e., conflicts or bumps). Our second, more realistic objective is to minimize overlaps. The new objective function provides a richer evaluation function for the algorithm.

When minimizing the number of bumped tasks, if 500 jobs are being scheduled and at least 10 must be bumped, then the number bumped is always an integer between 10 and 500. If most of the time, the number of conflicts is between 10 and 100, then most of the time the evaluation function is an integer between 10 and 100. This means that the fitness landscape is made up of very large flat plateaus. Empirically we have verified that plateaus exist and have found that the plateaus are far too large to exhaustively enumerate. Thus the precise size of the plateaus is unknown. At an abstract level, this is very similar to the landscape induced by the classic MAXSAT optimization problems. Changing the location of two requests in the permutation sometimes (or even often) does not change the output of the evaluation function. Thus, the evaluation function is a coarse metric. And the key to finding better schedules involves finding "exits" off of the plateaus leading to a better solution.

When using overlaps as an evaluation, the evaluation function is not just related to the number of tasks that must be scheduled, but also to their durations. If the number of conflicts is between 1 and 100, but the overlaps range from 1 to 50 time units, then the evaluation function can range over 1 to 5000. The landscape is still dominated by plateaus; however, the evaluation functions provides more differentiation between alternative solutions. This translates into fewer and small plateaus in the search space.

Discussions with humans who schedule AFSCN by hand suggests that minimizing conflicts in not the correct goal. The goal is to fit in all of the requests, even if that means modifying some requests.

An example of how the sum of overlaps is computed is presented in Figure 10-3. Note that this is a solution for the example problem in Figure 10-2. R8 could either be scheduled on antenna A1 or antenna A2 at Ground Station 2. In order to minimize the overlaps, we schedule R8 on A1 (the sum of overlaps with R6 and R7 is smaller than the sum of overlaps with R3 and R4). While this is a trivial example, it illustrates the fact that instead of just reporting R8 as bumped, the new objective function results in a schedule which provides guidance about the fewest modifications needed to accommodate R8. Perhaps R6 and R7 and R8 can be trimmed slightly, or perhaps R6 and R7 can be shifted slightly outside of their request windows; this may be better than getting bumped entirely.

We designed a schedule builder for Genitor to schedule all tasks and compute the sum of the overlaps. If a request cannot be scheduled without conflict on

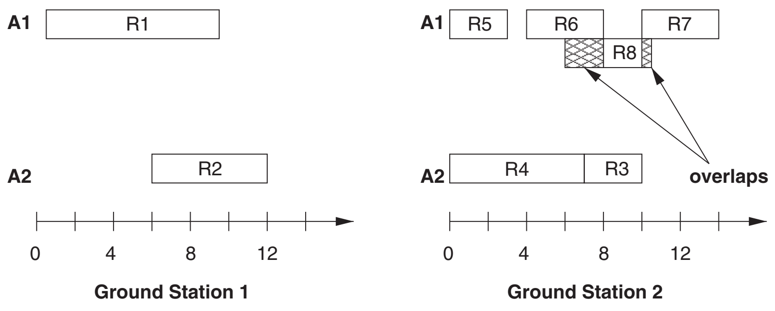

Figure 10-3. Optimizing the sum of overlaps.

any of the alternative resources, it overlaps; we assign such a request to the alternative resource on which the overlap with requests scheduled so far is minimized.

In Table 10-4, we present the results of running Genitor minimizing conflicts and Genitor minimizing overlaps for 30 runs, with 8000 evaluations per run. For "Genitor minimizing conflicts" we compute both the number of conflicts as well as the corresponding sum of overlaps (even though the schedule was not optimized to reduce overlaps) for the best schedule obtained in each run. Likewise, for "Genitor minimizing overlaps" we not only computed overlaps, we also computed the number of bumps needed to de-conflict the schedule.

The results show clearly that optimizing the number of conflicts results on average in a larger corresponding sum of overlaps than when the overlaps are optimized, and the increase can be quite significant. On the other hand, optimizing the sum of overlaps results in a number of bumps which is usually larger than when the conflicts are optimized; the increase is significant. These results also suggest that when minimizing the number of conflicts, longer tasks are bumped, thus resulting in a large sum of overlaps.

The significance of these results mainly relates to the correct choice of evaluation function. Using the wrong evaluation function can result in a scheduler that completely fails to solve the scheduling problem in a useful and appropriate manner. When conflicts are minimized, there is a strong bias toward removing large, difficult to schedule tasks. But if one must ultimately somehow put these back into the schedule it may require taking the whole schedule apart and starting over. Minimizing overlaps results in schedules that deal with all of the tasks that must be scheduled.

Barbulescu et al. (Barbulescu et al., 2004) provide a more detailed discussion of this problem. A more recent paper (Barbulescu et al., 2006) studies

	Genitor minimizing conflicts					
	Conflicts			Overlaps		
Day	Min	Mean	S.D.	Min	Mean	S.D.
03/07/02	42	43.7	0.9	1441	1650.8	76.6
03/20/02	29	29.3	0.5	803	956.23	53.9
03/26/03	17	17.6	0.5	790	849.9	35.9
04/02/03	28	28.03	0.18	1069	1182.3	75.3
05/02/03	12	12.03	0.18	199	226.97	20.33

	Genitor minimizing overlaps					
	Conflicts			Overlaps		
Day	Min	Mean	S.D.	Min	Mean	S.D.
03/07/02	55	61.4	2.9	913	987.8	40.8
03/20/02	33	39.2	1.9	519	540.7	13.3
03/26/03	24	27.4	10.8	275	292.3	10.9
04/02/03	35	38.07	1.98	738	755.43	10.26
05/02/03	12	12.1	0.4	146	146.53	1.94

Table 10-4. The results obtained for Genitor minimizing conflicts and Genitor minimizing overlaps by running 30 experiments with 8000 evaluations per experiment. The resulting schedules were then analyzed for both the number of bumps needed to de-conflict the schedule and the number of overlaps leaving everything in the schedule. This was done by using the permutation form of the solution in conjunction with the two different schedule builders.

several alternative methods for scheduling the AFSCN problem and explores why particular methods work well.

4. SCHEDULING WHEN TO TRACK OBJECTS IN SPACE

The Space Surveillance Network (SSN) is a collection of optical and radar sensor sites situated around the globe. The mission of the SSN is to maintain a catalog of thousands of objects in orbit around the earth. This catalog serves to facilitate object avoidance or contact, to prevent potential collisions during space flight, and to predict when specific orbits will decay.

The space catalog comprises both operational artificial satellites and debris from past space missions; each object is represented by information about its orbital trajectory. Many tracking sites in the SSN are *phased array* radars: devices that contain a two dimensional array of radio antennas that utilize constructive and destructive interference to focus a radiation pattern in a particular direction. Several objects can be tracked at once by interleaving track pulses subject to energy constraints. The *duty cycle* of the radar is a limit on the amount of energy available for simultaneously interleaved tracking tasks at a given instant. This defines the maximum resource capacity of the system. Due to surveillance and calibration tasks, the *available* resource capacity may fluctuate over time.

The SpaceTrack problem is an instance of SSN mission scheduling for an AN-FPS/85 phased array radar at Eglin Air Force Base in Florida, USA. Each day, radar operators at Eglin receive a consolidated list of requests (orbital objects with tracking frequency and durations) for performing observations on objects, along with their tracking priority. A *high* priority request must be filled during the day, while lower priority tasks are of less importance, and can be bumped in favor of critical requests. For each tracking request we identify a *priority weight* denoted by the scalar $w(r)$. The objective is to find an allocation that maximizes the priority-weighted sum of successful tracking tasks (WSS) subject to resource and temporal constraints.

It is important to note that this application involves uncertainty. Not every "track" that is scheduled will actually be successful. The exact size and shape of some objects may not be known. Irregularly shaped objects can display different radar cross sections from different perspectives. Thus, there are two estimations that create uncertainty: how much energy is needed to track the object, and the exact location of the object since its size and shape impacts it orbit. It is not uncommon for less than half of the scheduled tracks to be successful. When a track fails, it must be rescheduled.

With past information about an object's orbital trajectory and average radar cross section, we can use Keplerian laws to compute its expected *range* and *position* at a given time i. These expected quantities allow us to compute the following:

1 **Visibility windows.** The object's position at a given time defines whether it is visible to the array. Tracking cannot begin before the object rises over the local horizon (its *earliest start time* $est(i)$)and must complete before the object leaves the radar's field of view (its latest finish time $lft(i)$).

2 **Energy requirement** $c_j(i)$. The amount of energy necessary to illuminate an object is a function of the object's size and shape and its range with respect to the array. Distant objects require more energy than objects in low earth orbit. The sum of the energy requirements for all objects scheduled at i must not exceed a maximum available resource constraint (duty cycle limitation for the device).

3 **Expected value** $v_j(i)$. The probability of successfully tracking an object depends on how it is situated in the array's frame of reference. We derive the probability of detection by calculating an object's signal-to-noise ratio (SNR) profile over time. The SNR profile of an object is a function of its range and angle off the boresight direction (the normal vector to the

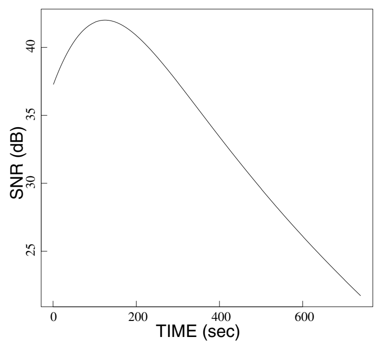

Figure 10-4. SNR profile of object with NORAD ID 27869 during a visibility window. The probability of detection (and expected weighted sum successful) is a monotonically increasing function of this quantity.

array's face plane). As the object moves through the array's field of view, this typically follows a smooth curve (see Figure 10-4). Multiplying this probability by the corresponding priority weight gives a tracking task's expected contribution to the weighted sum successful ($E[WSS]$).

Suppose system energy is infinitely available and the system has infinite capacity. In this case, the optimal solution can be calculated in polynomial time by assigning each task the tracking time that admits maximum SNR profile within a pass. However, when we constrain the available energy to realistic limits, the system becomes oversubscribed, and we can no longer feasibly schedule the entire set of tasks. Instead, we must construct a feasible subset of tasks to execute that maximizes the expected weighted sum successful.

On-peak scheduling. One approach to constructing this subset would be to pick a collection of tasks and assign them their "peak" SNR profile time. In other words, the start times for each task are fixed at the value that gives peak SNR (we call this value $x^*(i)$) and thus the highest probability of detection. We will call this approach *on-peak* scheduling.

The on-peak approach can be characterized as an integer programming problem. Let n be the number of tasks that must be scheduled and τ denote the number of time units available.

$$\begin{array}{ll} \text{maximize} & \mathbf{v}^T\mathbf{y} \\ \text{subject to} & A\mathbf{y} \leq \mathbf{c} \\ & \mathbf{y} \in \{0,1\}^n \end{array} \tag{2}$$

where $\mathbf{y}$ is a vector of n integer decision variables such that $\mathbf{y}_i = 1$ if the i^{th} on-peak task is to be included, and $\mathbf{y}_i = 0$ otherwise. Similarly, $\mathbf{v}$ is a vector of

n real values such that $\mathbf{v}_i = v_{x^*(i)}(i)$, and $\mathbf{c}$ is a resource constraint vector of τ elements such that $\mathbf{c}_j = C_j$. Finally, A is the $\tau \times n$ constraint matrix defined as follows.

$$A_{j,i} = \begin{array}{ll} c_j(i) & \text{if } x^*(i) \leq j < x^*(i) + d(i) - 1 \\ 0 & \text{otherwise} \end{array} \tag{3}$$

This is exactly an instance of the NP-hard $\{0, 1\}$ multidimensional knapsack problem (Garey and Johnson, 1979). It contains the traditional (single-dimensional) $\{0, 1\}$ knapsack problem as a special case when the constraint matrix is a $1 \times n$ row vector.

Relaxed scheduling. The on-peak approach is guaranteed to give a solution in which each task is scheduled to execute during the time that maximizes the expected value. However, a solution with maximal *cumulative* expected value is not necessarily an *on-peak* solution. By shifting tasks off their peak (e.g., two tasks away from each other), other tasks may be squeezed in between. This means an individual task will be allocated a suboptimal expected value as a result of being placed away from its peak SNR profile; but more total tasks can be scheduled. We will call this approach *relaxed* scheduling.

4.1 Algorithms for Spacetrack Scheduling

Scheduling SpaceTrack is a difficult problem. Finding an optimal on-peak schedule is NP-hard, and finding an optimal relaxed schedule is APX-hard (Sutton, 2006). This makes heuristic search an attractive option for finding approximate solutions.

The current method being used to perform Spacetrack Scheduling is a simple greedy scheduler. We will look at three other ways to solve the problem that are 10 to 20 percent better than greedy.

An exact solution. Though SpaceTrack is NP-hard, we can still generate the optimal on-peak solution on small tractable instances using a branch-and-bound technique to solve the integer programming problem defined in Equation (2). This gives us a baseline with which to compare methods that use the relaxed method.

On-peak local search. Suppose $\mathfrak{S}$ is the set of all tasks under consideration. A solution to on-peak scheduling is a subset $S \subseteq \mathfrak{S}$ of tasks that are all scheduled feasibly on their peak times to maximize $E[WSS]$. The schedule builder places each task in order of the permutation as close to its peak as possible.

Starting from an initial ordering, the on-peak local search (OP-LS) produces neighboring candidate solutions using the *exchange operator* which randomly swaps the position of tasks in the permutation. Since we are maximizing, next-ascent local search is used: a reordering that results in equal or higher $E[WSS]$ (as produced by the schedule builder) is accepted as the new permutation. Since the normal exchange neighborhood is $O(N^2)$ we select candidates for the exchange operator randomly, which makes this approach a form of stochastic hill-climbing.

Evolving relaxed schedules. We also employed the genetic algorithm Genitor (Whitley, 1989) to search the relaxed schedule space. In this case, an individual is represented by a permutation, but some other mechanism is needed to place a tracking task at a particular location in the schedule. We used a variant of greedy search to place a task at its on-peak location, or if that is not possible, at the best off-peak location possible. Fitness values are then computed by finding the $E[WSS]$ of the schedule produced by the schedule builder.

4.2 The SpaceTrack Results

Since SpaceTrack belongs to the class of NP-hard problems, we generate a set of 25 small tractable instances on which the optimal on-peak solution can be found with branch-and-bound. Each small instance is comprised of 50 tasks and 50 time units. The visibility windows were created by drawing from random distributions. The calculation of signal-to-noise ratio profiles requires expensive computation, and the distribution of the resulting values is difficult to control. Therefore, on the small set, the expected value curves are generated synthetically using a Gaussian curve.

To compare the algorithms on a large realistic problem, we create an instance using real data from NORAD's space database. This instance contains 5,000 tasks that must be assigned times from a set of 86400 time units (seconds during one day). The expected value curves and visibility times are created using actual position and velocity data computed using the standard SGP (Simplified General Perturbations) model (Hoots and Roehrich, 1980). These data are partitioned into 24 one-hour scheduling periods that are solved separately. The schedule found by each algorithm in a scheduling period is executed on a simulator that assigns success based on the probability function derived from SNR profile at the assigned tracking time.

The results from the set of small problems are reported in Table 10-5. The relaxed algorithm consistently finds higher mean values than the optimal on-peak solution found by branch-and-bound. We note that the On-peak Local Search algorithms (OP-LS) also found the same optimal solution as the branch-and-bound method in every case reported here.

The real-world problem is too large to be solved to optimality using branch-and-bound. Instead, we compare the relaxed algorithm with OP-LS as a

	On-peak			**Relaxed**	
instance	OPT	OP-LS		GENITOR	
		μ	σ	μ	σ
small 1	20.696170	20.696170	0.000000	21.474456	0.016000
small 2	21.453930	21.453930	0.000000	21.796775	0.004230
small 3	21.095790	21.095790	0.000000	21.452337	0.049200
small 4	18.106710	18.106710	0.000000	18.299666	0.069200
small 5	15.539030	15.539030	0.000000	16.024361	0.044300
small 6	15.622720	15.622720	0.000000	15.803709	0.018800
small 7	17.672190	17.672190	0.000000	17.892604	0.015300
small 8	16.115000	16.115000	0.000000	16.673159	0.084400
small 9	24.944260	24.944260	0.000000	25.269401	0.070200
small 10	18.481870	18.481870	0.000000	18.858413	0.034700
small 11	15.453560	15.453560	0.000000	16.217739	0.074900
small 12	16.274500	16.274500	0.000000	17.751006	0.178000

Table 10-5. A sample of results for all algorithms on the small set. The goal is to maximize the expected yield. Column two (OPT) indicates the optimal on-peak solution found by mixed-integer programming solver.

surrogate for the optimal on-peak solution. The results from each algorithm in each scheduling period on the real-world problem appear in Table 10-6. In this table, each expected weighted sum successful value found in a scheduling period is normalized by the sum of priority weights of all requests that need to be tracked in that period. Due to the stochastic nature of the simulation, the set of requests that require tracking in any given period may vary. We therefore report the normalized value in order to provide a fairer comparison between the algorithms.

	On-peak	**Relaxed**
period	OP-LS	GENITOR
1	0.4174375	0.4755850
2	0.2740852	0.3270757
3	0.1819694	0.2287325
4	0.4684505	0.5591947
5	0.3588590	0.4447416
6	0.2001081	0.2739436
7	0.3268069	0.4129994
8	0.4448630	0.5243267
9	0.4251086	0.4824034
10	0.3749262	0.4375215
11	0.4592485	0.5370493
12	0.4739717	0.5497464

Table 10-6. Expected weighted sum successful (normalized) found by each algorithm in each scheduling period using real-world data.

Again, the algorithms that employ the relaxed approach obtain a higher expected value than that obtained using the on-peak local search (OP-LS). Higher expected values correspond to a higher yield of successful tasks. For the real-world problem we can also model the potential gain in actual tasks successfully tracked. The number of tracking tasks determined to be successful by the simulator represents the "yield" of the schedules. We report these data for each algorithm found after simulation of the entire schedule day in Table 10-7.

	On-peak	**Relaxed**
priority	OP-LS	GENITOR
1	922 (52.15%)	1078 (60.97%)
2	1073 (48.57%)	1209 (54.73%)
3	1018 (40.32%)	1257 (49.78%)
4	817 (27.92%)	1032 (35.27%)
5	606 (15.53%)	699 (17.91%)
TOTAL	4436 (33.28%)	5275 (39.57%)

Table 10-7. Yield of successfully tracked passes by priority after the completion of all schedules for an entire day. The number in parentheses represents the percentage of successfully tracked passes out of those requested.

The results reported here are much better than the greedy methods currently in use. We are continuing to explore different approaches for solving the Space-Track Scheduling problem. Sutton et al. (Sutton et al., 2007) is the most recent paper; this paper explores the use of dynamic local search. It also explores tradeoffs in terms of scheduling as many tracks as possible versus minimizing the mean time between tracks. Because unsuccessful tracks must be rescheduled, this problem also poses interesting issues related to the sustained long term performance of a scheduling system.

5. CONCLUSIONS

Our results on three applications demonstrate the viability of using heuristic search (local search and genetic algorithms) in combination with permutation based representations. In each case, the heuristic search methods outperformed existing approaches to the applications.

Our results also show that heuristic methods can improve on "optimal" search methods if the optimal method must be applied to a smaller decomposed problem (and hence the solutions are not optimal for the full problem) or if a restricted version of the problem must be solved in order to guarantee optimality.

Exploring the permutation space rather than directly manipulating schedules appears to be an effective search strategy. Directly manipulating schedules means that operators must be much more customized to act on a particular type of schedules. When using a permutation representation, the specifics and constraints for a specific type of schedule are hidden in the schedule builder. When changing from one application to another only the schedule builder needs to change. And our experience suggests it is much easier and intuitive to *build* a feasible schedule using the permutation as a priority queue than to *modify* a schedule so as to define a neighborhood of transformations that systemically converts one schedule into another schedule. Thus, the use of a permutation representation allows a great deal of code reuse from one application to the next.

One concern with both the Coors Scheduler and the SpaceTrack scheduler is that looking at one day's worth of data (or a few days of data) does not adequately evaluate the true value of using an improved scheduling system. In the case of the Coors problem, the improved schedules may not be totally sustainable in the long run; part of the reduction in time at dock may be due to better scheduling that exploits surplus inventory. However, with more product going directly from production line to dock there must be some reduction in inventory (and some older product in inventory will have to be discarded). When this happens some of the apparent reduction in the time at dock may be lost. However, this only happens because the overall scheduling is better; the overall operation should still be more efficient with less product going to inventory.

On the SpaceTrack problem we expect just the opposite effect: we expect additional long term gains. This is because when an object is not tracked the importance of tracking that object increases. A lower priority object that is not tracked can become a higher priority object. The relaxed approach to scheduling SpaceTrack could result in fewer high priority objects over time.

Exploring the sustained long term impact of a scheduling method is an interesting area of research that is not particularly well addressed in the literature on scheduling, and which deserves more attention.

Acknowledgments

We wish to thank Patrick O'Kane and Richard Frank at ITT industries for their generosity and assistance regarding the Eglin SpaceTrack radar model. This research was partially supported by a grant from the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grants number F49620-00-1-0144 and F49620-03-1-0233. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Funding was also provided by the Coors Brewing Company, Golden, Colorado.

References

L. Barbulescu, A.E. Howe, L.D. Whitley, and M. Roberts (2006) Understanding algorithm performance on an oversubscribed scheduling application. *Journal of Artificial Intelligence Research (JAIR)*.

L. Barbulescu, J.P. Watson, D. Whitley, and A. Howe (2004) Scheduling Space-Ground Communications for the Air Force Satellite Control Network. *Journal of Scheduling*.

John Bresina, Mark Drummond, and Keith Swanson (1995) Expected solution quality. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.

T. Cormen, C. Leiserson, and R. Rivest (1990) *Introduction to Algorithms*. McGraw Hill, New York.

Lawrence Davis (1985) Applying Adaptive Algorithms to Epistatic Domains. In *Proc. IJCAI-85*.

Lawrence Davis (1985) Job Shop Scheduling with Genetic Algorithms. In John Grefenstette, editor, *Int'l. Conf. on GAs and Their Applications*, pages 136–140.

Lawrence Davis (1991) *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

M. R. Garey and David S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

David Goldberg (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

David Goldberg and Jr. Robert Lingle (1985) Alleles, Loci, and the Traveling Salesman Problem. In John Grefenstette, editor, *Int'l. Conf. on GAs and Their Applications*, pages 154–159.

D.E. Goldberg and M. Rudnick (1991) Genetic algorithms and the variance of fitness. *Complex Systems*, 5(3):265–278.

Felix R. Hoots and Ronald L. Roehrich (1980) Spacetrack report no. 3: Models for propagation of NORAD element sets. Technical report, Peterson Air Force Base.

Yuichi Nagata and Shigenobu Kobayashi (1997) Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In T. Bäck, editor, *Proc. of the 7th Int'l. Conf. on GAs*, pages 450–457. Morgan Kaufmann.

D.A. Parish (1994) A Genetic Algorithm Approach to Automating Satellite Range Scheduling. In *Masters Thesis*. Air Force Institute of Technology.

T. Starkweather, S. McDaniel, K. Mathias, D. Whitley, and C. Whitley (1991) A Comparison of Genetic Sequencing Operators. In L. Booker and R. Belew, editors, *Proc. of the 4th Int'l. Conf. on GAs*, pages 69–76. Morgan Kaufmann.

A. Sutton, A.E. Howe, and L.D. Whitley (2007) Using adaptive priority weighting to direct search in probabilistic scheduling. In *The International Conference on Automated Planning and Scheduling*.

A.M. Sutton (2006) *A two-phase dynamic local search algorithm for maximizing expected success in on-line orbit track scheduling*. MS thesis, Colorado State University, Fort Collins, CO.

Gilbert Syswerda (1991) Schedule Optimization Using Genetic Algorithms. In Lawrence Davis, editor, *Handbook of Genetic Algorithms*, chapter 21. Van Nostrand Reinhold, New York.

Gilbert Syswerda and Jeff Palmucci (1991) The Application of Genetic Algorithms to Resource Scheduling. In L. Booker and R. Belew, editors, *Proc. of the 4th Int'l. Conf. on GAs*. Morgan Kaufmann.

J.P. Watson, S. Rana, D. Whitley, and A. Howe (1999) The Impact of Approximate Evaluation on the Performance of Search Algorithms for Warehouse Scheduling. *Journal on Scheduling*, 2(2):79–98.

Darrell Whitley, Timothy Starkweather, and D'ann Fuquay (1989) Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator. In J.D. Schaffer, editor, *Proc. of the 3rd Int'l. Conf. on GAs*. Morgan Kaufmann.

Darrell Whitley and Nam-Wook Yoo (1995) Modeling Permutation Encodings in Simple Genetic Algorithm. In D. Whitley and M. Vose, editors, *FOGA - 3*. Morgan Kaufmann.

L. Darrell Whitley (1989) The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best. In J.D. Schaffer, editor, *Proc. of the 3rd Int'l. Conf. on GAs*, pages 116–121. Morgan Kaufmann.