Comparing Heuristic, Evolutionary and Local Search Approaches to Scheduling *

Soraya Rana Adele E. Howe L. Darrell Whitley Keith Mathias
Computer Science Department
Colorado State University
Fort Collins, CO 80523

email: {rana,howe,whitley,mathiask}@cs.colostate.edu
URL: http://www.cs.colostate.edu/~{rana, howe, whitley}
tele: 1-970-491-7589
fax: 1-970-491-2466
keywords: scheduling, evaluation

Abstract

The choice of search algorithm can play a vital role in the success of a scheduling application. In this paper, we investigate the contribution of search algorithms in solving a real-world warehouse scheduling problem. We compare performance of three types of scheduling algorithms: heuristic, genetic algorithms and local search. Additionally, we assess the influence of heuristics on search performance and check for bias induced by using a fast objective function to evaluate intermediate search results.

*This research was supported by NSF grant IRI-9503366, by NSF Research Initiation Award #RIA IRI-930857, by Coors Brewing Company and by the Colorado Advanced Software Institute (CASI). CASI is sponsored in part by the Colorado Advanced Technology Institute for purposes of economic development. Soraya Rana was supported by a National Physical Science Consortium Fellowship.

1 Introduction

Approaches to scheduling have been varied and creative. Many points on the spectrum from domain independent to knowledge intensive have been explored. Examples of the diverse underlying scheduling techniques are local search (e.g., [2]), simulated annealing (e.g., [12]), constraint satisfaction (e.g., [4]), and transformational (e.g., [7]), to name a few. The problem is choosing the appropriate technique for a specific type of scheduling application.

One method for determining the appropriate method is to investigate the generality of particular scheduling algorithms. Studies of single approaches demonstrate the utility and capabilities of a particular system without saying which approach is "best". This method is most appropriate for knowledge intensive approaches which require significant effort for re-tooling to new applications and which include a range of support capabilities beyond simply producing a schedule.

Another method is to compare alternative approaches against each other on the same set of scheduling problems (e.g., [1,6,5]). This expedites determining which approach is best, while acknowledging that some support capabilities, such as user interfaces and development environments, will not be evaluated.

We conducted a comparative study of several knowledge poor search techniques for scheduling a real-world manufacturing line. The domain involves scheduling orders to be filled at the Coors brewery plant with a fixed set of production lines. The purpose of the study is to measure the contribution of search in this scheduling application and determine which technique is best.

Three key issues are examined. First, the selection of search algorithm significantly affects the performance of scheduling. Most other studies have focused on the effect of the representation while using the same search strategy for all cases. Second, it can be shown that combining general search algorithms with a few domain based heuristics improves performance for this application. Third, the use of a fast objective function by a search algorithm can significantly bias the results, suggesting the need for careful calibration between the fast objective function and the full simulator and ultimately the real warehouse. We have found that while our fast objective function executes three orders of magnitude faster than the detailed simulator, our data indicates that the best results with respect to the fast objective function are not always the best results when evaluated using a detailed simulator.

2 Background

Scheduling is typically characterized by multiple jobs contending for a limited number of resources. Scheduling applications are often posed as constraint satisfaction problems (CSP) with the ultimate goal of completing all jobs as quickly as possible while meeting

as many of the constraints as possible. For the developer, most of the difficulty lies in organizing the constraints and successfully modelling the environment in terms of those constraints. For the scheduling system, the difficulty is in maneuvering through the space of possible schedules to locate one or more feasible solutions.

A variety of approaches have been adopted for search in scheduling. The ISIS system [4] searches through a space of world states to locate a position for an order which does not violate the hard constraints of the order. ISIS uses beam search to schedule backwards from a due date (a primary constraint) to find a good location for the order. Should this prove to be impossible, the due date constraint might be relaxed and the order rescheduled in a forward direction. In a reactive system such as OPIS [8], several types of local search are performed to revise schedules. OPIS uses beam search to reschedule a set of operations pertaining to a single order. Another search technique is applied when operations (from multiple orders) conflict on a single resource; a forward-dispatch search reschedules specific operations on an alternate resource. The CERES system developed by Drummond et al. [2] defines the search space as a projection graph of feasible solutions. Given a start state, the system applies all appropriate schedules and calculates all end states. From each of those end states, it generates the next level of possible groups to execute. This process results in all feasible schedules which can be searched by a variety of algorithms to optimize some objective function [1]. Zweben [12] uses simulated annealing to perform iterative repair of schedules in the GERRY system.

Constraint based schedulers reduce their search spaces through their hard constraints: at any given point in time, only a subset of possible schedules will meet the hard constraints. Consequently, they have emphasized efficient and expressive representations of constraints over effective search methods. For the Coors scheduling application, enough product is available in inventory and from the production lines to fill the set of orders which are to be scheduled; any permutation of the set of orders will yield a valid schedule. Consequently, any order can be placed in any position in the schedule (even if it is unwise to do so), making the search space over n orders $n!$. At the same time, we can impose soft constraints on the schedules that are produced. In general, we would like the wait time for orders at the loading docks to be low. For example, some orders require product that must come from production and inventory; for these orders, we can compute time windows of activity with respect to the production line and inventory. Soft constraints can then be used with the time window information to intelligently restrict the set of possible schedules. For the work presented here, we translate these soft constraints into a small set of heuristic rules for scheduling. These rules can be used in conjunction with any permutation of orders, where the permutation serves as a queue when allocating resources to orders.

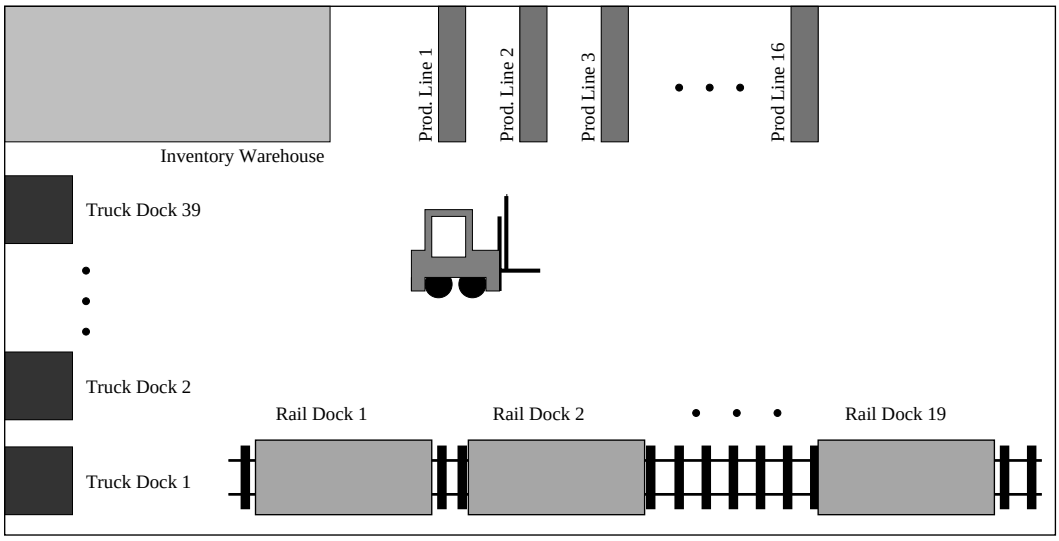

Figure 1: A simplistic view of the Coors scheduling environment.

3 Coors Order Scheduling

The warehouse scheduling problem involves sequencing orders which must be filled using a predefined production schedule, a limited number of loading docks and a variable mix of products in any given order (as shown in Figure 1). During a 24 hour period, approximately 150 to 200 orders are filled. For the data set presented here, 525 orders were actually filled from the Coors warehouse and production lines over approximately three days.

The manufacturing plant includes 16 production lines, which can produce 500 different types of product after considering different packaging and labeling options. Different packaging options for a given *brand* are different products from a shipping point of view. In filling any particular order, some product may have to come from inventory. We accept the production line schedules as fixed; the production line is generated over a different time scale (e.g., weekly production) and is constrained in ways unrelated to the orders. For example, it is preferable to run the same brand of product as a batch, since a significant cost may be incurred by changing a production line from one brand to another.

Product is organized into *pallets*; each pallet contains a given number of product units (e.g., cases). Orders are a single load of multiple products and are separated onto different docks according to the mode of transportation required to ship the product: either truck or rail. The manufacturing plant has 39 truck docks and 19 rail docks. A truck carries approximately 18 pallets while a rail car typically carries approximately 60 pallets. Orders are separated into truck and rail orders before scheduling begins; however, truck and rail orders simultaneously compete for product.

3.1 Internal and External simulators

This application employs two different simulations of the Coors warehouse. The *internal* simulator is a coarse grained, fast simulator used by the search algorithms to quickly compute the objective function. The *external* simulator is a detailed and relatively slow simulation of the warehouse used to verify the quality of the solutions.

Both simulators use event lists to model vehicles moving in and out of truck or rail docks along with production line events. In both simulators, orders are placed on a waiting queue for loading docks where one order corresponds to one vehicle at dock. When a dock is free, the first order is removed from the queue and placed on the dock. Orders remain at dock until they are filled with product drawn either from inventory or from the production lines.

The external simulator models forklifts moving individual pallets of product. The internal simulator saves time by simply moving all available product from inventory to the dock in one block as needed and adding a time penalty to account for the forklift wait time. The internal simulator is also more coarse in that the production line events take place at 15 pallets per event. These differences in granularity lead to dramatically different execution times; the internal simulator runs in less than one tenth of a second, while the external simulator requires three minutes to evaluate the same schedule. Despite the differing execution times, we attempted to correlate the output of the two simulators by varying the parameter settings of the internal simulator. The internal simulator has a parameter to change the temporal granularity of the simulation (i.e., amount of time treated as a single event) and two other parameters, minimum time at dock and amount of time to transfer each pallet, which determine the wait time due to inventory retrieval.

To tune the parameters of the internal simulator, we tested parameter settings (18 combinations) over 30 random schedules to determine the values that yielded the strongest correlations between the internal and external simulators for our objective function. Additional tests were run on the parameter settings to determine how coarse the production line event list could be without degrading the correlation results. The resulting correlation of 0.463 indicates that the two simulators are weakly correlated.

4 Scheduling Approaches

The search space in this application is quite large ($n!$). Thus, the suitability of a particular search algorithm to the problem dictates the quality of the schedule that can be found in reasonable time. We compared three search algorithms to baselines of randomly generated schedules, a simple heuristic scheduler, and the actual Coors schedule. Local search closely fits the iterative model of schedule construction in which schedules are gradually improved. Genetic algorithms provide a promising alternative for a more globally motivated search. These approaches were varied by the operator employed (for local search) and also by using heuristics to provide good initial points

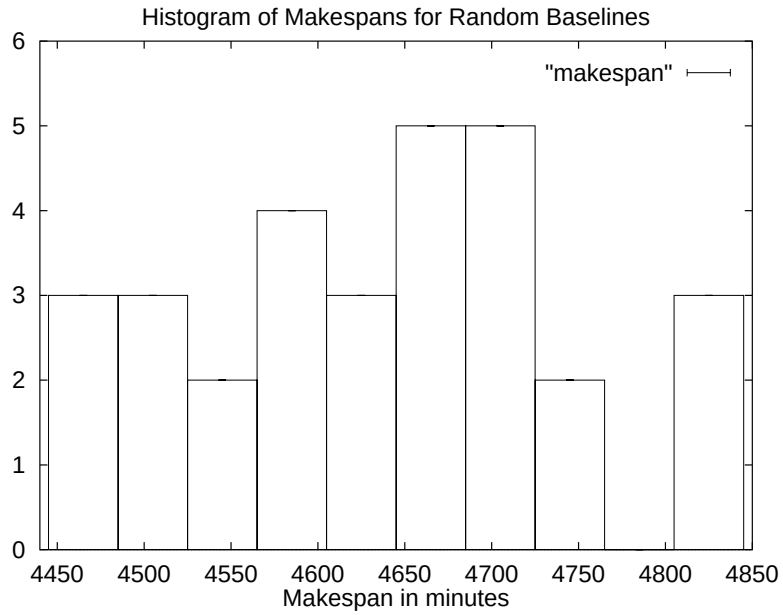

Figure 2: Histogram of the makespans of 30 random schedules.

to begin the search. Random schedules characterize worst case behavior (i.e., what can we expect if we just guess at a solution), while heuristic scheduling gauges the effect of even a little domain knowledge. All of these approaches represent the schedules as a permutation of orders.

4.1 Baselines

4.1.1 Random Schedules and the Coors Solution

Evaluating a set of random permutations estimates the density of good solutions and the variance in the search space of solutions. For this data, we also have the actual customer order sequence developed and used by Coors personnel to fill customers orders: the time required for the order sequence to complete is 4421 minutes. A schedule will be considered *competitive* if the amount of time required to fill the orders is less than or equal to the Coors solution. Figure 2 illustrates the wide range of makespans in a sample of 30 random schedules. From this we can see that these random permutations did not produce competitive schedules and the solutions varied over a 400 minute span.

4.1.2 Heuristically Generated Schedules

As noted earlier, constraint based scheduling approaches result in smaller search spaces by exploiting domain knowledge about constraints. Given that this application includes no hard constraints, a simple alternative is to develop heuristics based on soft

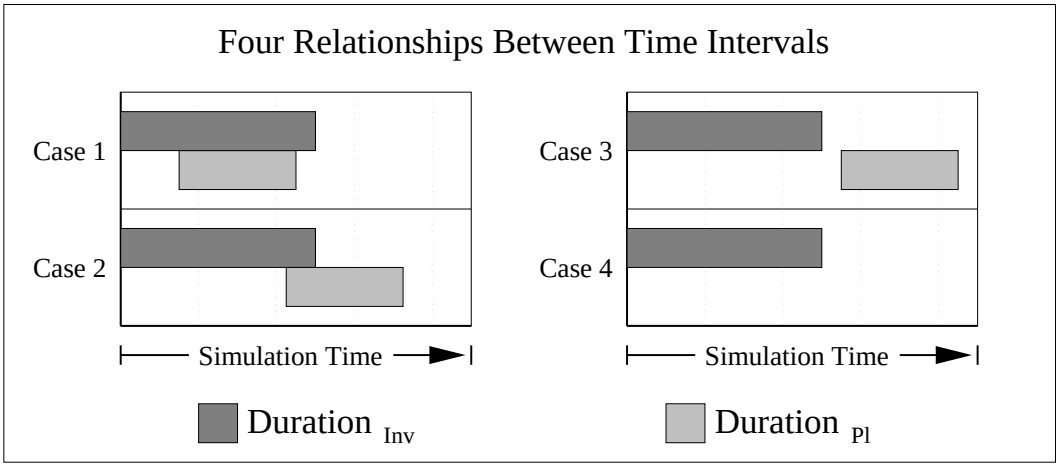

Figure 3: Four Relationships for the two time intervals $duration_{Inv}$ and $duration_{Pl}$.

constraints specific to this application. These heuristics can be used on their own or in conjunction with the search techniques to exploit domain specific knowledge to constrain search.

The domain specific heuristics were derived from analyzing the actual schedule used by Coors to process the set of orders. For any given order, two time intervals can be generated based on the amount of product needed from inventory and the amount of product needed from the production line. The two time intervals are represented by $duration_{Inv}$ and $duration_{Pl}$. These intervals correspond to the wait time for product to be loaded from inventory ($duration_{Inv}$) and the wait time for product to be produced and loaded from the production line ($duration_{Pl}$). The relationships between the two intervals can be one of four possible cases (illustrated in Figure 3). The ideal situation for large orders is to have the two time intervals overlap because large orders should be filled from both inventory and production. The first three cases illustrate that the intervals can completely overlap, partially overlap or not overlap at all. The fourth case occurs when all product needed to fill an order can be drawn from inventory (i.e. $duration_{Pl} = 0$). The $duration_{Inv}$ always begins at the current time and is always nonzero due to a minimum time penalty incurred when an order moves onto a dock.

Heuristic derivation of a schedule works by starting from the beginning of the schedule and adding orders one at a time until they are all included in the solution. Thus, the heuristics measure the value of placing a given order in the next position based on the relationship between its $duration_{Inv}$ and $duration_{Pl}$ variables as well as its size. For each of the four interval cases, a different function is used to compute the rank for a given order; these ranks are comparable and indicates the overall best order to process as the one with the lowest rank. The heuristics simply choose the first lowest ranked order when there are ties; therefore, the heuristic solutions differ when computed from different random permutations.

The ranking for each order is determined from three functions. If an order's intervals completely overlap, the order is ranked based on the number of products required to fill the order (favoring the higher number). If an order's intervals partially overlap or do not overlap, the order is ranked based on the number of products required by the order (favoring the higher number) multiplied by the difference in interval completion times. For an order to be filled entirely from inventory, it should be small (< 2500 *units* for rail orders and < 300 *units* for truck orders).

4.2 Search Algorithms

4.2.1 The GENITOR Genetic Algorithm

GENITOR [11] is what has come to be called a "steady state" genetic algorithm [3] (GA). Steady state GAs differ from traditional GAs in that offspring do not replace parents, but rather replace some less fit member of the population. In the GENITOR algorithm, offspring are produced one at a time and deterministically replace the worst member of the population. In addition, the GENITOR algorithm allocates reproductive opportunities based on the rank of the parents in the population. A linear bias is used so 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.

Parent 1	a	b	c	d	e	f	g	h
Cross Points		*	*	*		*		
Parent 2	h	g	f	e	d	c	b	a
Parent 1 selected								
Offspring	a	f	d	c	e	b	g	h

Figure 4: Example crossover and recombination process.

To apply a GA to a permutation problem, a special operator was needed to ensure that recombining two permutations also yields a permutation. Syswerda [10] notes that operators may attempt to preserve the position of the elements in the parents, the relative order of elements in the parents or the adjacency of elements in the parents when extracting information from parent structures to construct offspring. In previous experiments [9], we found that the best operator for the warehouse scheduling application is Syswerda's order crossover operator. This operator recombines two parent permutations by randomly selecting permutation positions for reordering. When a set of elements are selected in Parent 1, the same elements are located in Parent 2. The selected subset of elements in Parent 1 are then reordered so that they appear in the same relative order as observed in Parent 2. Elements which are not selected in Parent 1 are directly passed to

the offspring structure in the same absolute positions. Figure 4 illustrates the crossover and recombination process.

4.2.2 Local Search

The local search algorithms, 2-opt and random swap, use multiple trials of iterative improvement of the schedule until no further change can be made. The 2-opt local search operator, which is classically associated with the Traveling Salesrep Problem, was explored as the search operator. In the TSP, the permutation actually represents a Hamiltonian Circuit, so that a cycle is formed. 2-opt cuts the permutation at 2 locations and reverses the order of the segment that lies between the two cuts. For the TSP, such an operator is minimal in the sense that it affects only two edges in the graph, and no operator exists which changes a single edge. Nevertheless, if relative order is more important than adjacency (i.e., graph edges), then 2-opt is not a minimal operator with respect to the warehouse scheduling problem. In fact, 2-opt potentially changes the schedule in a significant way for all of the orders contained in the reversed segment. A less disruptive operation to permutations is to simply swap two positions. We will refer to this as the *swap* operator. All of the elements between the two swapped elements remain unchanged.

To apply either of these operators to a large permutation will incur $O(n^2)$ cost. The number of 2-opt moves or swaps over all pairs of n elements is $\frac{(n^2-n)}{2}$. A steepest ascent strategy is thus impractical given a limit on the total number of evaluations; we employ a next-ascent strategy where each improving move is accepted instead. However, these local search operators are still at a disadvantage due to the large size of the local search neighborhood. To avoid this problem, the swap operator is applied to random pairs of positions within the permutation rather than systematically examining all pairs.

4.3 Search Initialization

Our search is initialized in two ways. The simplest and most common initialization method is to use random permutations as the initial sequences. An alternative is to use the heuristic rules in conjunction with a permutation to create a schedule; this, in effect, reorders the permutation to create a queue that is consistent with the set of heuristic scheduling rules. This property allows the heuristic solutions to be used to seed the initial solutions for all of the search methods.

5 Experiments

Our experiments consist of an eight-way comparison between the baseline schedules (random and heuristic) and the three search algorithms initialized randomly and seeded with the heuristic solutions. For each baseline method, the best schedule out of 50 is saved

as the result schedule for a single run. The genetic algorithm and random swap search method were given an equal number (100K) of evaluations on the internal simulator. The 2-opt algorithm was limited to a single pass which requires 118K evaluations. All algorithms were run 30 times with different initial random seeds.

We compared the performance of the algorithms from three perspectives. The first comparison measures relative performance of the algorithms to determine whether search improves performance, and if so, which algorithm performs best. Second, we compare heuristic initialization to random initialization to assess the effect of domain specific knowledge. Third, the fast objective function should bias the performance; the question is how robust are the algorithms in the presence of the bias.

5.1 The Objective Function

For our application, a satisfactory solution balances several goals. One goal is to maximize throughput of the orders over each day. A related goal is to minimize the *mean-time-at-dock* for an order, meaning that we are effectively minimizing the idle time of the docks and indirectly increasing the throughput of processing orders. A second goal is to maximize the amount of product that is loaded directly from the production line, thus reducing the amount of product in storage (which is similar to minimizing work-in-process). This can also be achieved by minimizing the amount of *average-inventory*. A third goal is to minimize the makespan. One problem with makespan in this application is that the production schedule dictates exactly the minimum makespan (4311.15 minutes) due to a product bottleneck. Since there are many docks, minimizing the makespan is not as effective as minimizing *mean-time-at-dock*: even if the last order cannot be filled until the 4311 minute mark, we would like to have the other orders filled and off the docks as soon as possible. While we do not include makespan in our objective function, it is reported in the results section because it is a common measure of solution quality.

Our cost objective function is built around measures of the first two goals: *average-inventory* and *mean-time-at-dock*. Starkweather et al. [9] report that attempting to minimize either of these two measures independently can have a negative impact on the other measure. We also found this to be the case in our experiments optimizing a single measure. Emphasizing the minimization of *mean-time-at-dock* in particular can dramatically drive up the *average-inventory*. Minimizing *average-inventory* has a more modest impact on *mean-time-at-dock*. Thus the objective function we employed combined *mean-time-at-dock* and *average-inventory* using methods developed by Bresina et al.[1]. The objective function combines the two values using their respective means (μ) and standard deviations (σ) over a set of solutions. The means and standard deviations for our two variables of interest are periodically computed from the population of solutions (500 permutations) in the genetic algorithm or from the last 100 solutions found during local search. The formula can be written as follows:

$$obj = \frac{(ai-\mu_{ai})}{\sigma_{ai}} + \frac{(mt-\mu_{mt})}{\sigma_{mt}}$$

where *ai* represents *average-inventory* and *mt* represents *mean-time-at-dock*.

5.2 Results

Tables 1 and 2 summarize the results found by each of the approaches evaluated on the internal and external simulators respectively. Since the external simulator provides the most accurate estimate of performance in the real warehouse, all statistical comparisons are made using the results from the external simulator. As a test of whether or not we have produced reasonable solutions, we can compare our solutions with the values generated for the Coors solution: the Coors solution produced an *average inventory* of 549817.25, a *mean-time-at-dock* of 437.55 minutes and a makespan of 4421 minutes. The data in Table 2 shows that the search techniques did locate competitive solutions. In many cases, the improvements are dramatic. Relative to the random solutions, all of the algorithms with the exception of randomly initialized 2-opt produce statistically significantly better solutions than random (verified using a one-tailed t-test). The randomly initialized 2-opt solutions are not significantly different from the random baseline solutions for *mean-time-at-dock*. Inter-algorithm comparisons can also made to determine which search algorithm is best for our application. We ran two one-way ANOVA's with algorithms as the independent variable and performance as the dependent variable. We found a significant effect of algorithm on performance ($P < 0.001$). Closer analysis using one-tailed t-tests indicates that the genetic algorithms produce statistically significantly better solutions in comparison with the other search algorithms while the local search algorithms perform similar to one another.

The next question we will address is whether heuristic initialization of the search algorithms improves the quality of the solutions. An ANOVA indicates a dependence between search initialization and performance. One tailed t-tests confirm that in all cases, the search algorithms using heuristic initialization perform significantly better than the corresponding algorithms using random initialization. We found no significant difference between the randomly initialized genetic algorithm and heuristic baseline solutions while the heuristically initialized genetic algorithm did produce significantly better solutions than the heuristic baselines.

The last question we pose is how robust are the algorithms with respect to the bias in the internal simulator. As expected, Tables 1 and 2 show a discrepancy between the two simulator's evaluations of the solutions particularly for the heuristically initialized local search methods. Since both of these algorithms use a greedy strategy, they take the first improvement as measured by the internal simulator. Many of the "improvements" are actually leading the local search algorithms astray due to the inherent bias in the objective function. Consequently, the heuristically initialized local search solutions degraded when presented with inaccurate evaluations. The genetic algorithm is the only search

	Baseline		Genetic Algorithm		2-opt		Random Swap	
	Random	Heuristic	Random	Seeded	Random	Seeded	Random	Seeded
Mean Time at Dock								
Mean	459.90	407.68	392.49	395.39	423.11	398.71	400.11	399.03
Sd	3.2386	1.2684	0.2792	0.7290	5.6136	3.1036	4.8298	3.4629
Average Inventory								
Mean	627306	363160	364050	354271	530446	394383	370718	350228
Sd	16041	1332	1739	1376	20088	12424	20987	3070
Makespan								
Mean	5378.38	4318.00	4318.00	4318.00	4762.69	4318.00	4318.69	4318.00
Sd	119.644	0.000	0.000	0.000	101.368	0.000	2.792	0.000

Table 1: Internal simulator results for *Mean time at dock* and *Average Inventory* for all 8 approaches using the *combination* objective function.

	Baseline		Genetic Algorithm		2-opt		Random Swap	
	Random	Heuristic	Random	Seeded	Random	Seeded	Random	Seeded
Mean Time at Dock								
Mean	451.91	398.23	397.37	391.04	457.15	425.61	439.51	426.06
Sd	7.7196	1.3550	7.4734	2.8439	6.3953	2.0711	4.0977	1.6718
Average Inventory								
Mean	645428	389539	389475	384078	539864	426707	433411	412366
Sd	28497	5278	9272	8229	23524	17872	20537	14252
Makespan								
Mean	4631.74	4311.15	4311.81	4311.15	4389.37	4311.15	4314.31	4311.15
Sd	113.312	0.000	3.406	0.000	81.797	0.000	10.823	0.000

Table 2: External simulator results for *Mean time at dock* and *Average Inventory* for all 8 approaches using the *combination* objective function.

technique that overcomes the bias in the internal simulator to consistently produce the best solutions.

6 Conclusions and Future Work

Two areas for future work are studying the time/quality tradeoff in using search and using the heuristic solutions to identify biases in our objective function. Given that our heuristics produced such high quality solutions, one should naturally wonder if the time difference between generating heuristic solutions and using genetic search to improve those solutions is worthwhile in terms of the economic impact of the improvements and the time requirements. Seemingly small increments in performance can have dramatic economic impact on the warehouse in the long term. Measurements for the time/quality tradeoff [1] can be applied here using a different set of experiments which would impose a convergence criteria on the genetic algorithm instead of imposing an evaluation limit as was done for this paper. On the second issue, analysis on the internal simulator has been performed solely through generating and analyzing the internal simulator output and the external simulator output. Comparing the solutions produced by the heuristics and the search algorithms may provide information that will be helpful in reducing the bias in the internal simulator.

All of the search techniques as well as the heuristics generated better solutions than the actual schedule used by the brewery. We also showed that heuristics can significantly improve the quality of solutions found by the search algorithms. In fact, the heuristics themselves outperformed local search methods even when the local search methods were initialized with heuristic solutions. When working with a less than perfect objective function (or simulator in our case), the local search algorithms are unable to overcome the biases in the internal simulator.

The most general conclusion we can draw from our study is that the choice of search technique is not arbitrary. Different search techniques will exploit different characteristics of a search space. In our application, a globally motivated (genetic) search produced much higher quality solutions than local search despite the inherent biases in the objective measure used by the search. In fact, it was the only algorithm able to produce significantly better solutions than the heuristic search strategy.

The goal of this study has been to present our specific scheduling application and study the effectiveness of different search algorithms on our problem. The long term goals are to examine a wider variety of search algorithms as well as performing similar studies using a different domain. Since our problem is posed as a static optimization problem, a study similar to this one could be performed using schedulers that work in constraint based domains and reactive environments.

References

[1] John Bresina, Mark Drummond, and Keith Swanson. Expected solution quality. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[2] Mark Drummond, Keith Swanson, and John Bresina. Robust scheduling and execution for automatic telescopes. In Monte Zweben and Mark Fox, editors, *Intelligent Scheduling*, pages 341–370. Morgan Kaufmann, 1994.

[3] Larry Davis (ed). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[4] Mark S. Fox. Isis: A retrospective. In Monte Zweben, Mark Fox, and Fox, editors, *Intelligent Scheduling*, pages 3–28. Morgan Kaufmann, 1994.

[5] Carla O. Pedro Gomes and Julie Hsu. An assignment based algorithm for resource allocation. Technical Report unpublished, Rome Laboratories, 1995.

[6] Pat Langley. Systematic and nonsystematic search strategies. In *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, pages 145–152. Morgan Kaufmann Publishers, Inc, 1992.

[7] Douglas R. Smith and Eduardo A. Parra. Transformational approach to transportation scheduling. In Mark H. Burstein, editor, *ARPA/Rome Laboratory Knowledge-Based Planning and Scheduling Initiative Workshop Proceedings*, pages 205–216, Palo Alto, CA, February 1994. Morgan Kaufmann Publishers, Inc.

[8] Stephen F. Smith. Opis: A methodology and architecture for reactive scheduling. In Monte Zweben and Mark Fox, editors, *Intelligent Scheduling*, pages 29–66. Morgan Kaufmann, 1994.

[9] T. Starkweather, D. Whitley, K. Mathias, and S. McDaniel. Sequence scheduling with genetic algorithms. *New Directions in Operations Research*, 1992.

[10] Gilbert Syswerda. Schedule optimization using genetic algorithms. In *Handbook of Genetic Algorithms*, chapter 21, pages 322–349. Van Nostrand Reinhold, 1991.

[11] L. Darrell Whitley. The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116–121. Morgan Kauffman, 1989.

[12] Monte Zweben, Brian Daun, Eugene Davis, and Michael Deale. Scheduling and rescheduling with iterative repair. In Monte Zweben and Mark Fox, editors, *Intelligent Scheduling*, pages 241–255. Morgan Kaufmann, 1994.