The Impact of Approximate Evaluation on the Performance of Search Algorithms for Warehouse Scheduling

J. P. Watson, S. Rana, L. D. Whitley, A. E. Howe
Computer Science Department
Colorado State University
Fort Collins, CO 80523 USA
E-mail: {watsonj, rana, whitley, howe}@cs.colostate.edu
Telephone: 01 (970) 491-5373, Fax: 01 (970) 491-2466

Please direct correspondence to L. D. Whitley.

The Impact of Approximate Evaluation on the Performance of Search Algorithms for Warehouse Scheduling

Keywords: warehouse scheduling; approximate evaluation; heuristics; genetic algorithms; local search

Abstract

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. Search based solutions require fast objective functions. Thus, a fast low-resolution simulation is used as an objective function. A slower high-resolution simulation is used to validate solutions. We compare the performance of a constructive scheduling algorithm to a genetic algorithm and local search approach. The constructive algorithm is based on a heuristic built specifically for this application. We also tested a hybrid of the genetic algorithm and local search approaches by initializing the search using the domain-specific heuristic. This hybrid genetic algorithm was able to find the best solutions when evaluated by the high-resolution simulation. Finally, we consider the effect of using the high-resolution simulation to filter a set of solutions found by the different approaches.

1 Introduction

Domain independent search algorithms exploit fast evaluation to maneuver through solution gradients. Thus, they require little domain knowledge a priori, acquiring what they need as they search. Their generality makes search the paradigm of choice for poorly understood and/or highly complex problems for which close to optimal solutions are desirable. However, as the evaluation time increases and dominates the computation time, the likely success of this family of algorithms wanes.

Search based methods have been highly successful in solving scheduling problems. In flowshop and jobshop scheduling, fast evaluation methods have been developed to expedite search. However, real world scheduling problems with resource limitations may require simulation to determine the value of a proposed schedule. Simulations, unfortunately, tend to be costly, which undermines the value of search based solutions.

We built a scheduler for one such application: warehouse scheduling at a Coors brewery. The Coors warehouse scheduling problem requires sequencing shipment of customer product orders. Good solutions should minimize both the average time required to fill an order and the running average of the warehouse inventory level. To accommodate changes in orders and production lines going down, the scheduler needs to produce a schedule quickly (e.g., a new 24 hour schedule within minutes). This paper focuses on how different scheduling algorithms perform given these criteria.

The problem is complex and difficult to optimize for several reasons. The search space is quite large. Each schedule typically consists of several hundred customer orders. Schedule evaluation is costly. These characteristics distinguish this problem from commonly studied scheduling problems such as flowshop and jobshop scheduling, which generally minimize the makespan of the schedule [25] [18] [19].

Like flowshop problems, however, the Coors warehouse scheduling problem is posed as a search through a permutation space: all sequences of customer orders. This formulation facilitates solution by domain independent search algorithms. However, with a costly evaluation function and a huge search space, domain-specific constructive algorithms that rely less on search provide a compelling alternative. Such methods use knowledge to reduce the number of solutions that must be evaluated. The central component of our domain-specific methods is a heuristic developed specifically for the Coors warehouse scheduling problem.

We studied the performance of both heuristic based and domain independent search algorithms on our problem. We chose two methods that directly use the domain-specific heuristic: Iterative One-Samp [16] [15] and Limited Discrepancy Search [15]. One-Samp refers to the generation of one solution (or sample) using a constructive heuristic method. Iterative One-Samp uses iterative random restarts and thus applies the construction heuristic multiple times. Limited Discrepancy search is a restart method that views the construction heuristic as building a path through a decision tree. A path through the tree with 1-discrepancy ignores the heuristic choice at only one decision point along that path. Thus, a decision tree of depth n will have n paths in the decision tree with 1 discrepancy.

We considered four domain-independent search algorithms: random sampling, the *shift* local search operator, the *exchange* local search operator and a genetic algorithm. Random sampling serves as a performance baseline; it provides statistical information on the mean and variance of solutions in the search space. The local search algorithm is implemented as a next-descent local search. The shift operator selects an order and moves it to a new position in the permutation; the exchange operator selects two customers from the permutation and swaps their positions. Genetic and evolutionary algorithms have been applied to a variety of scheduling problems, including flowshop [23] [4] and jobshop [7] [1] [9] [14] problems, timetabling problems [11] [6], as well as to a variety of dynamic and real-world applications (e.g., [28], [24] [10]). A survey of this literature is given by Hart and Corne [13].

Finally, *hybrid algorithms* seek to achieve the benefits of both types of algorithms by combining the search algorithms with the domain-specific knowledge. Our results suggest that this hybrid approach provides the best results for this problem. It leverages the ability of the domain-specific heuristic to quickly focus the search on competitive solutions while also exploiting the domain independent search algorithm's ability to discover solutions which may be off the path suggested by the heuristic.

The research described in this paper extends our previous studies of this problem [20] [21] [26]. We finalize this work by more thoroughly exploring the effect of approximate simulation as an evaluation function–with particular emphasis on fast low-resolution simulation versus slower high-resolution simulation. In many complex real world scheduling problems and especially in dynamic environments, simple mathematical models may not be adequate as an objective function.

The central question is: how can we trade-off accuracy against run time in the design of the algorithms that use the evaluation function? To answer it, we compared the performance of a wide variety of algorithms as measured by both simulators (see Section 4). The best solutions found when using the low resolution simulator are good, but are rarely the best when tested on the high resolution simulator. Based on these results, in section 5, we propose a strategy for compensating for inaccurate fast evaluation toward the end of search.

2 The Coors Warehouse: Schedule Simulation and Evaluation

The Coors manufacturing operation and warehouse consists of 16 production lines, a number of loading docks, and a warehouse for product inventory. Each production line can manufacture 500 distinct products; different packaging and labeling options for a single brand are considered different products from the shipping perspective. The plant contains 39 truck and 19 rail-car docks for the loading of various customer orders. Orders are loaded using product obtained from either the production lines or from inventory reserve.

The production line schedule is generated over a different time scale (i.e., weekly) than that of filling of customer orders. Furthermore, the production line schedule is constrained in ways unrelated to the customer orders. For example, it is preferable to simultaneously produce products associated with a single brand, since a significant cost may be incurred in changing the production line from one brand to another.

Orders are separated into truck and rail-car orders before scheduling begins. Orders are then assigned to empty docks of the corresponding type. An order is filled from pallets either coming off the production lines or from warehouse inventory. An order remains at a dock until it is completely filled, at which point the dock becomes empty and available for another order. While only orders using an equivalent transport (i.e., truck or rail-car) compete for dock space, all orders simultaneously compete for product, whether from the production line or inventory.

Simulation is the only method that can be used to evaluate the quality of a particular schedule in this domain. The *internal* simulator is a fast, low-resolution simulator, which is integral to the search algorithms we used. In contrast, the *external* simulator is a slow, high-resolution simulator, which is external to the search algorithms and is used to verify solution quality.

In both simulators, orders are placed in a queue to wait for a vacant loading dock. When a dock is 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. Both the production line schedule and the initial inventory are predetermined. Product comes from the production lines organized into pallets; each pallet contains some number of product units (e.g., cases). An order specifies method of transport (a truck or rail-car) as well as a certain combination of product pallets. A truck can hold approximately 18 pallets, while a rail-car carries approximately 60 pallets. During a typical 24 hour period, approximately 150 to 200 orders are filled.

The external simulator models a fixed number of fork-lifts moving individual pallets of product. In contrast, the internal simulator moves the product required by an order from inventory to a dock in a single step with the time required being represented by a constant. The internal simulator also lumps production line events into blocks with 15 pallets per event rather than 1 pallet per event. Such simplifications result in a dramatic reduction in the simulation cost for a schedule. On a Sun SPARC 20, the internal simulator requires an average of 50 milliseconds to simulate a schedule, while the external simulator averages 5 seconds. Obviously, this 100 fold reduction[1] in run-time results in some loss of simulation accuracy.

Only orders that have been allocated to some dock can compete for product. For purposes of allocating empty docks, the permutation of customer orders implicitly forms two priority queues–one for truck and one for rail-car orders. Truck docks are distinct from rail docks. While only orders using the same type of transport compete for dock space, all orders simultaneously compete for product. We therefore include all truck and rail orders in a single permutation. This permutation represents the customer's priority for obtaining needed product: the first customer in the queue that needs a particular available product is the one that gets that product–*assuming that the customer has already been assigned to a dock.* Thus, a rail-order may be first

[1]We had previously reported a 3 order of magnitude different in the internal and external simulators [21]. The difference has been reduced to 2 orders of magnitude by using machines with considerably more memory.

in the overall combined permutation, but a truck-order which occurs later in the combined permutation may be allocated to a truck dock before the rail-order. An order at dock then has priority over an order not yet assigned to a dock.

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

2.1 Schedule Evaluation

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 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

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

Events do not occur at the same granularity in the two simulators. While the time units are the same in both simulators, the low-resolution simulation uses larger time steps *between* events than the high-resolution simulator. As a result, there will be differences in the running average inventory, I, as reported by the two simulators.

Although I may change for different schedules, the final inventory, $\mathcal{J}_{\mathcal{F}}$, will always be the same: the amount left in inventory must be the total amount produced minus the total amount shipped. In reality, shipping never stops, and the perpetuation of a smaller running average inventory translates in general into a smaller inventory.

Starkweather and Whitley [26] report for this same Coors warehouse problem that attempting to minimize either of the mean time at dock or average inventory metrics independently can have a detrimental effect on the other metric. We have also observed the same trade-off.

Multi-objective problems such as this are often transformed into single-objective problems by forming a linear combination of the individual objectives. The single objective function used, derived from Bresina (1995) [3], is given as follows:

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

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. The solution set and the objective function parameters are updated several times during the run of each algorithm; this results in a non-stationary (time-varying) objective function. Normalization by the standard deviations has the effect of standardizing the units and the weighting each of the metrics equally.

3 The Scheduling Algorithms

We investigate a variety of methods for generating solutions to the Coors warehouse scheduling problem. The algorithms were chosen based on both our prior experience with the Coors warehouse scheduling problem [20] [21] [26] and successes reported in the literature [15] [2].

These algorithms fall into three general categories:

1) Constructive methods that use explicit knowledge of the simulator state.

2) Search or sampling methods that do not use simulator state information.

3) Hybrid (combination) methods.

For our constructive methods, a special version of the internal simulator was constructed which integrated computations of the heuristic. The domain independent methods do not use state information to decide how to schedule orders. Instead, these methods simply pass in a fixed permutation of orders to the internal simulator and get back a single real value to indicate the quality of that permutation. The hybrid approaches use a constructive method to build initial schedules and then apply search/sampling methods to refine those schedules.

In the Coors environment, we must be able to reschedule within a reasonable period of time to deal with changes in production (e.g., if a production line goes down). For our problem instance of 525 orders, approximately 90 minutes of CPU time on a Sun SPARC 20 is required to perform 100,000 schedule evaluations on the internal simulator. For all of the domain-independent search methods we examine, run-time is dominated by use of the internal simulator for schedule evaluation. Due to the negligible algorithm overhead, fair comparisons of these search algorithms can be made by allocating an equivalent number of schedule evaluations to each trial.

The constructive heuristic we employ requires approximately 10 times the execution time of a normal evaluation on the internal simulator. Thus, those algorithms that repeatedly use the construction heuristics are restricted to sampling 10,000 potential solutions.

3.1 The Constructive Algorithm and Domain Specific Heuristic

The domain-specific heuristic function, E, is defined using the following symbols:

Π : A random permutation of unscheduled customer orders.

i : A customer order.

t : A decision point for placing a new order on a dock (dock event).

π_t : The partial schedule up to dock event t.

ζ_t : The state of the simulation at dock event t.

The state of the simulation is modeled by the set of orders on the docks, the mixture and number of products in inventory and the remainder of the production schedule.

Define the following constants:

U_i: The number of units of product needed to fill order i.

T_i: The number of product types needed to fill order i.

C : A constant representing the approximate lower bound on order size that is set to 2500 product units for rail orders and 300 units for truck orders. The upper bound is 2000 units (18 pallets) for trucks and 6500 units

(60 pallets) for rails.

Define the following functions:

$F_I(i, \zeta_t)$: Estimated finish time of i due to inventory given ζ_t.

$F_P(i, \zeta_t)$: Estimated finish time of i due to production lines given ζ_t.

We next compute $E(i, \zeta_t)$ as a heuristic; it attempts to identify jobs that can be quickly finished while also exploiting overlap in product drawn from both inventory and production.

$$E(i, \zeta_t) = \begin{cases} \frac{U_i}{C} & \text{if } F_P(i, \zeta_t) = 0 \\ \frac{F_P(i,\zeta_t) - F_I(i,\zeta_t)}{T_i} & \text{if } F_P(i, \zeta_t) > F_I(i, \zeta_t) \\ \frac{1}{T_i} & \text{if } F_P(i, \zeta_t) <= F_I(i, \zeta_t) \end{cases}$$

The constructive method begins with a fixed partial schedule that is the actual initial configuration of the warehouse for our data set. We will refer to the initial partial schedule as π_0 and the initial state of the simulation as ζ_0. The schedules are constructed as follows:

At dock event t, the domain-specific heuristic computes

$$z = min(\forall_{x \in \Pi} E(x, \zeta_t)).$$

Ties between orders are resolved by choosing the first customer order encountered in Π. If Π is chosen randomly, ties are resolved randomly. The customer order represented by z is then added to the partial schedule π_t and removed from the permutation of unscheduled orders. Ties are common due to competition between orders for popular products.

The minimum value and maximum value for the range of $E(i, \zeta_t)$ depends on the set of orders being processed; however, the minimum value is always greater than 0. If all product can be drawn from inventory (i.e., $F_P(i, \zeta_t) = 0$), small orders are preferred (minimize $\frac{U_i}{C}$). If the required product from the production line can be obtained faster than the required product from inventory, then more diverse orders are preferred (minimize $\frac{1}{T_i}$). If the required product from the production line takes longer to obtain than the required product from inventory, then more diverse orders are preferred that also have short waiting times (minimize $\frac{F_P(i,\zeta_t) - F_I(i,\zeta_t)}{T_i}$). The bias in the heuristic function attempts to maximize the overlap between wait times. When this is not possible, choose the smallest order possible that can be filled solely from inventory. The heuristic is based on strategies of human schedulers at Coors.

The domain-specific heuristic is embedded within the low resolution simulator. Thus, using this heuristic to reorder and evaluate a schedule requires 10 times the run-time of a normal evaluation.

3.1.1 Iterative One-Samp

Because of the tie breaking strategy of the heuristic, different initial permutations, Π, produce different final schedules. The term *One-Samp* refers to a single solution generated using a heuristic constructive method. Thus, *Iterative One-Samp* in this context refers to the generation of multiple heuristically generated solutions using different initial random permutations [15] [16] [5].

3.1.2 Limited Discrepancy Search

In contrast with the random sampling, Limited Discrepancy Search (LDS) is a systematic search technique that exploits heuristic information in tree search problems [15]. LDS has been shown to be effective when a good heuristic is available. A *discrepancy* is a decision point at which the heuristic is not followed (the chosen decision is not the one selected as "best" by the heuristic). LDS iteratively searches the space with a limit on the number of discrepancies allowed on any path to a leaf node. In the first iteration, no discrepancies are allowed (i.e., the heuristic is always followed). In general, during the i^{th} iteration, at most $i - 1$ discrepancies are allowed along any path.

One factor to consider when applying LDS to the Coors warehouse scheduling problem is the large branching factor. When only one discrepancy is allowed, there are N possible choices for first element in the schedule, where N is the number of orders to be scheduled. When the heuristic function A is not followed, there are $N-1$ possible choices for what to schedule instead.

In a binary tree, there are exactly $N-1$ leaf nodes representing solutions that have 1 discrepancy. However, in our tree, if we consider all possible alternative choices for each discrepancy, then there will be $\sum_{i=1}^{N-1} i = \frac{n(n-1)}{2}$ possible 1-discrepancy paths through the search tree, each corresponding to a possible schedule. For this application, the search tree would have over 100,000 1-discrepancy paths, which would require an evaluation time of approximately 15 hours. Instead of considering all possible alternatives at each decision point in the tree, we limited the discrepancies to the 30 best choices as ordered by the domain-specific heuristic. This would seem to be a quite reasonable compromise. However, LDS never found a solution that improved on the initial One-Samp solution corresponding to the use of the heuristic with no discrepancies. This shows that it is difficult to find good solutions that are only 1 discrepancy away from the heuristically generated solutions. Therefore, we subsequently only report the results for Iterative One-Samp.

3.2 The Domain Independent Methods

3.2.1 Random Sampling

Random sampling produces schedules by generating random permutations of customer orders. Randomly sampling a large number of schedules provides information about the distribution of solutions in the search space, as well as a baseline measure which suggests the difficulty of the problem.

3.2.2 The GENITOR Genetic Algorithm

GENITOR [30] is a "steady-state" genetic algorithm [8]. In GENITOR, a single pair of chromosomes mates and produces a single child. The child then replaces the worst, or least fit, chromosome in the population. The result is a form of elitism, in which the best individual produced during the search is always maintained in the population. The population size for the genetic algorithm was fixed at 500 elements.

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. The linear selection bias was set to 1.1 [30].

The typical genetic algorithm encodes solutions as bit strings, enabling the use of standard crossover operators such one-point and two-point crossover

[12]. In contrast, we encode solutions as permutations. Therefore, a special crossover operator is required to ensure that recombination of two parent permutations results in a child that is a permutation that inherits relevant characteristics of the two parents.

Syswerda [27] notes that permutation-based recombination operators can attempt to preserve either the position, relative order or adjacency of the elements in the parents when extracting information from the parents to construct a child. In previous experiments [26], we found that the best recombination operator for the Coors warehouse scheduling problem is Syswerda's (relative) order crossover operator. This makes intuitive sense, given that the relative order in which customer orders are filled affects the availability of product for later orders.

Figure 1 illustrates the operation of Syswerda's order crossover operator. A subset of elements is selected in Parent 1; the same elements are then located in Parent 2. The selected subset of elements in Parent 1 are then re-ordered so that they appear in the same relative order as observed in Parent 2. Elements that are not selected in Parent 1 are directly passed to the offspring in the same absolute positions. A more detailed description and analysis of the operator is given in [29].

PLACE FIGURE 1 HERE

3.2.3 Random Exchange and Shift Local Search

We previously explored both the use of 2-opt and *exchange* for Coors warehouse scheduling [21]. The *2-opt* operator is classically associated with the Traveling Salesrep Problem or TSP [17]. The *exchange* operator [23] picks two elements in the permutation and then exchanges them. Not surprisingly, the exchange operator is better suited to resource allocation problems than 2-opt, since it manipulates the relative order of the customers in the permutation.

We also tested another local search operator, *shift*, for this application. When an order is moved to its new position, orders following the new position are all shifted to a position one higher to make room for the moved order. We chose *shift* because its movement of a single order to another location seemed well suited to this application where a single order (or even single product within an order) may be the bottleneck.

Both operators start from an initial permutation and apply the operator iteratively to generate new permutations, which are evaluated as schedules. For this application, when applied systematically, there are ((525/524)/2) = 137,550 pairs of possible exchanges or 525*524 = 275,100 possible shifts. Steepest descent is impractical given that we limit the number of evaluations to 100,000, which is less than the total number of possible operator applications for either operator. We instead use a next-descent hill-climbing strategy where each improving move is immediately accepted. In addition, we obtained the best results by applying the exchange operator to random pairs of customer orders within the permutation rather than by systematically examining all pairs in some fixed order. The *shift* operator systematically selects each job in the permutation, but randomly selects its new location. We refer to these operators as the *random exchange* and *random shift* operator, respectively.

3.3 The Hybrid Algorithms

We consider two initialization methods for the genetic algorithm and local search algorithms. The most common method of initialization simply uses randomly generated schedules for the initial population of the genetic algorithm and the initial solutions used by the local search algorithm. An alternative is to use the constructive algorithm described in Section 3.1 in conjunction with a random permutation to create the initial schedules. In effect, the elements of the random permutations are re-ordered to create a schedule that is consistent with the set of heuristic order sequencing rules. We subsequently refer to the methods as *random initialization* and *domain-specific initialization*, respectively.

4 Relative Algorithm Performance

The performance of the various scheduling methods were tested over 30 independent trials each. For the genetic algorithm and exchange local search, both random and domain-specific initialization were tested. For each trial, the best schedule obtained by the algorithm, in terms of the minimal objective function value under the internal simulator, was recorded. This schedule was then evaluated on both the internal and external simulators for the metrics: order mean time at dock and average inventory.

The objective function parameters μ_I and μ_M were initialized to 0, while σ_I and σ_M were initialized to 1. These parameters were updated after every 500 evaluations.

Due to the computation requirements of the heuristic, we allocated 10,000 schedule evaluations to LDS and Iterative One-Samp and 100,000 to the domain independent and hybrid methods. For our problem instance of 525 customer orders, approximately 90 minutes of CPU time on a Sun SPARC 20 is required to perform 100,000 schedule evaluations on the internal simulator; approximately the same amount of time is required for 10,000 evaluations of LDS or Iterative One-Samp using the contruction heuristic. For the hybrid methods, the domain-specific initialization is only applied to a small number of permutations and turned out to be an insignificant fraction of the total run-time.

4.1 Empirical Results

Tables 1 and 2 compare the performance of the search algorithms when using the internal and external simulators for solution evaluation, respectively.

PLACE TABLE 1 HERE

PLACE TABLE 2 HERE

For our test data, we have the actual customer order sequence developed and used by Coors personnel to fill customer orders. This solution produced an average inventory of 549817.25 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 2 illustrates that (mean) solutions obtained by random sampling are *not* competitive with the Coors solution.

The data in Table 2 demonstrates that each of the search algorithms can produce solutions that are competitive with the Coors solution (we report only mean performance in Table 2). This improvement is often dramatic. Statistical comparison of the competitive search algorithms indicates that each performs better than random sampling, as verified by a series of two-tailed t-tests ($p < 0.0001$).

To determine whether any of the algorithms significantly outperformed the others, we ran one-way ANOVA's with algorithm as the independent variable and performance metric (either mean time at dock or average inventory) as the dependent variable. These tests indicated a significant main effect ($p < 0.0001$) in the choice of search algorithm on each of the performance metrics. In addition, one-way ANOVA's with initialization method as the independent variable (either random or domain-specific) and performance metric (either mean time at dock or average inventory) as the dependent variable were performed. Similarly, these tests indicated a significant main effect ($p < 0.0001$) in the choice of initialization method for the domain independent algorithms.

A series of two-tailed t-tests ($p < 0.0001$) indicated that both local search operators perform significantly worse than Iterative One-Samp. Further t-tests indicated no significant difference between the randomly initialized genetic algorithm and Iterative One-Samp. However, the hybrid genetic algorithm does significantly outperform Iterative One-Samp ($p < 0.0001$ for mean time at dock, $p < 0.0107$ for average inventory). A series of two-tailed t-tests showed that the genetic algorithm produces significantly ($p < 0.0001$) better solutions than both local search operators, given identical forms of initialization.

Domain-specific initialization significantly outperformed random initialization for both local search operators ($p < 0.0001$). For the genetic algorithm, domain-specific initialization more dramatically improves performance for the metric mean time at dock ($p < 0.0001$) than for average inventory

($p < 0.046$). Thus the hybrid outperforms other methods, which supports the contention that knowledge helps in large search spaces.

5 The Impact of Approximate Evaluation

The output of the low-resolution simulator serves as the objective function for all the search algorithms tested. The high-resolution simulator verifies the quality of the final solutions. While dramatically reducing execution costs, a low-resolution simulator necessarily comes with a reduction in accuracy. During the construction of the fast low-resolution simulator for Coors, we used correlation as a measure of how well the output of the fast simulator compared to the results generated by the high-resolution simulator. By tuning the parameters of the fast simulation, we were able to obtain an overall correlation of 0.463 on *randomly* generated solutions. This correlation seems low. So how do the differences between the two simulators impact our results? Clearly, the absolute results differ significantly between the simulators as shown in Tables 1 and 2. To address this question, we plot the mean time at dock and average inventory as measured by the internal simulator against the corresponding values as measured by the external simulator.

We first show the best solution found using the internal simulator on each of the 30 runs for:

1) random sampling (RBASE)

2) the Heuristic using Iterative One-Samp (HBASE)

3) the genetic algorithm (RGA)

4) the hybrid genetic algorithm (HGA)

5) exchange local search (REXCH)

6) hybrid exchange local search (HEXCH)

7) shift local search (RSHIFT)

8) hybrid shift local search (HSHIFT)

PLACE FIGURE 2 HERE

Figure 2 shows the mean time at dock. The top graph includes all of the results; the bottom graph focuses on the most competitive solutions: HBASE, RGA and HGA. Relative to random sampling, all algorithms except randomly initialized local search produce much better results on both simulators; however, the heuristic local search algorithms do not improve the solutions as much as the other three. Second, note that the Iterative One-Samp using the heuristic function produces highly correlated results: improvement on the internal simulator directly translates into improvement on the external simulator among this set of solutions. This could be a strong argument in favor of Iterative One-Samp for this problem.

For local search, while the results are better than Iterative One-Samp on the internal simulator, the results are dramatically worse on the external simulator – in fact, local search is only marginally better than random search when evaluated on the external simulator. Initializing both forms of local search with the domain-specific heuristic improves overall performance some on both the internal and external simulators. Finally the shift operator appears to do slightly better than exchange for random initialization; the difference is fairly small for heuristic initialization.

Finally, the genetic algorithm with random initialization is actually the best overall algorithm according to the internal simulator. In fact, all solutions generated by the genetic algorithm have virtually the same evalution on the internal simulator. While some of the solutions are extremely good, overall the set of solutions exhibits significant variation on the external simulator. Initializing the hybrid genetic algorithm population results in slightly poorer performance according to the internal simulator, but a much tighter distribution and more predictably good solutions on the external simulator. The solutions found by the hybrid are biased strongly by the initial population and so retain some of the robustness of the heuristic solutions.

PLACE FIGURE 3 HERE

The performance on average inventory is somewhat different. Figure 3 shows the average inventory. As before, the top graph includes all the algorithms; the bottom graph focuses on the three best algorithms from mean time at dock. The random sampling results are again not competitive. The results of random local search are the poorest on both the internal and external simulators. The results of the hybrid exchange local search operator are the best obtained on the internal simulator, but this improvement failed to carry over to the results on the external simulator. The Iterative One-Samp results and the randomly initialized genetic algorithm are more or less the same on both the internal and external simulation: their distributions are highly overlapped. The hybrid genetic algorithm is not as good as hybrid local search or randomly initialized genetic algorithm as measured on the internal simulator; however, on the external simulator, it holds a slight advantage.

Although the correlation between the low and high resolution simulators is not high, it does appear to be adequate to direct most search algorithms, at least until near the end of the search process. The final correlations for our most competitive search algorithms vary from higher than random schedules (.56 on the inventory metric and .97 on the time metric for Iterative One-Samp) to much worse (.15 on inventory and -.26 on time for the genetic algorithm). However, our results show that the algorithm still manages to find excellent solutions (i.e., much better than that found by a human scheduler and competitive or better than other algorithms) when relying on the low resolution simulator.

5.1 Noisy Evaluation Functions and Genetic Algorithms

In effect, the internal simultor is a noisy evaluation function compared to the external simultor. In any noisy function, because the genetic algorithm retains the best solutions as it searches the space, over time the members of the population all have similar evaluations. Using a noisy evaluation function, these solutions will not have the same evaluation as on the non-noisy evaluation function. As diversity is driven out of the population and all members of the population have similar evaluations, the correlation between the noisy evaluation and the true evaluation goes to zero.

Over the 30 runs in Figures 2 and 3, the genetic algorithm has generated "best found" solutions that are very similar in evaluation on the internal simulator, but which display significant variation on the external simulator. This is typical behavior for genetic algorithms when ran on noisy functions[22].

When sampling a noisy function, it is very common for the signal-to-noise ratio to be such that the signal is increasingly lost in the noise as one samples better solutions that have increasingly similar evaluations. We do not know the shape of the fitness landscape that best characterizes the objective function corresponding to the external simulator or what kind of distribution characterizes the difference between the internal simulation and the external simulator in our application. However, it is instructive to consider an example where the true function is parabolic. Figure 4 shows such a function with random uniform noise (generated between 0 and 1) added. As one samples closer to the global optimum, the true evaluation is lost in the noise. Also, the correlation between the true evaluation and the noise evaluation approaches zero as one samples closer to the global optimum. Table 5 give the correlations between the true evaluation and the noisy evaluation when sampling 1/2 and 1/4 and 1/6 of the space around the optimal.

In our evaluations with the Coors application, for the construction heuristic used by One-Sample, the correlation between the internal simulator and the external simulator is very high. This is due in part to the fact that the internal simulator is not being used to guide the heuristic: the decisions of the heuristic are independent of the internal simulator. Also, the principles used by the heuristics are general principles that appear to be have a similar impact under both the internal and external simulator. But this also means that the constructive heuristic is not exploiting more low level and local interaction effects between jobs which can be exploited by a more general search paradigm.

However, under genetic search correlation does go to zero as better and better solutions are found with respect to the internal simulator. In the next section, we propose a strategy that can be employed to improve the solutions found by the genetic algorithm to compensate for this phenomenon.

5.2 Leveraging the External Simulator

So far, we have treated the external simulator as strictly a validation tool representing ground truth. Since the internal simulator is an inaccurate noisy evaluation function, the external simulator can also be exploited as *part* of the optimization process. While it is too costly to use as an evaluation function in general, we could use it to choose among a small set of candidate solutions suggested by the various optimization methods. We thus compared 100 solutions in the final population of a single run of the hybrid genetic algorithm to the 100 solutions produced by Iterative One-Samp. In both cases, we picked the 50 best solutions as measured by the internal simulator. We also picked 50 random solutions from the final population generated by the genetic algorithm; likewise, the 50 best and 50 random Iterative One-Samp solutions were chosen.

PLACE TABLE 3 HERE

Table 3 shows that how the external simulator can be used to exploit the diversity of a single hybrid genetic algorithm population. The means of the 50 best solutions for a single run of the hybrid genetic algorithm are basically the same when compared to the means of the 30 independent runs as measured by the external simulator (given in Table 2). The same is true for the 50 best Iterative One-Samp solutions on a single run compared to best solutions 30 independent runs. The mean time at dock values are almost identical, and the average inventory values are very similar.

The best of the 50 solutions improves over the mean by about 1 percent in each case. For Iterative One-Samp, the best and the mean of the randomly selected solutions are worse than the best solution selected based on the internal simulator. In this case, the tight correlation between the internal and external simulation found for Iterative One-Samp is a disadvantage: the internal simulator itself is a good predictor of the external simulator. But the hybrid genetic algorithm, while driving all solutions to a similar value on the internal simulation, showed significant variation on the external simulator. The 50 randomly selected solutions included an even better solution than that found among the 50 best solutions, resulting in a 2 percent improvement.

Comparing the best external simulator solutions for Iterative One-Samp and the hybrid genetic algorithm on the 100 sample solutions, the result achieved by the hybrid genetic algorithm is approximately 3 percent better.

PLACE FIGURE 5 HERE

Finally, we examined the trajectory of this single run of the hybrid genetic algorithm. Figure 5 shows the mean time at dock and average inventory at 10, 20, 40, 60, 80 and 100 thousand evaluations. The internal simulator results are plotted against the external simulator results. One can see the improvement on the internal simulator and a trend toward improved solutions on the external simulator over the course of the run. But solutions continue to vary significantly when evaluated on the external simulator, especially late in the search. Also, most of the improvement appears to take place during the first 60,000 evaluations.

PLACE TABLE 4 HERE

We next looked at the 50 best solutions and 50 randomly selected solutions in the hybrid genetic algorithm population at 60, 70, 80 and 100 thousand evaluations using the external simulator. The results presented in Table 4 suggest the solutions found at 60 thousand evaluations are basically as good as the solutions at 100 thousand evaluations. The best solution ever seen on this problem as evaluated by the external simulator is found in the randomly selected set after 70,000 evaluations: a mean time at dock of 378.8 with an average inventory of 375,949.

6 Conclusions

We have analyzed the performance of several algorithms that use an approximate evaluation method in the search for solutions to the Coors warehouse scheduling problem. Furthermore, we have assessed the impact of the approximate evaluation method on search algorithm performance.

Our results demonstrate that domain-specific constructive algorithms equal or outperform any randomly initialized search/sampling algorithms we tested. Although the domain-specific constructive algorithm is not best by itself, when combined with domain-specific initialization, the performance of the genetic algorithm is greatly improved and outperforms any other method tested. Thus, we have affirmed the value of domain knowledge in a scheduling problem with large search space and approximate evaluation.

Furthermore, we have found that some search algorithms expected to perform well actually do not improve over a simple iterative sampling algorithm. A systematic search algorithm, LDS, fails to improve on the performance of Iterative One-Samp, and the hybridized exchange local search operator actually degrades the solution quality of its initial Iterative One-Samp solution.

When different algorithms are applied to the Coors scheduling problem using the approximate evaluation function, they sample the search space very differently. When final algorithm solutions are evaluated on both simulators, the results generated by Iterative One-Samp are highly correlated. On the other hand, the domain independent and hybrid methods show significant variation in the external simulator results–even when the internal simulator indicates the solutions have similar evaluations. While a strong correlation between the internal simulator and external simulator is generally good, somewhat surprisingly it was not required by the genetic algorithm to locate good solutions as evaluated by the external simulators.

We found that hybrid solutions can be robust with respect to approximate evaluation for this application. This result seems to be due to several factors. First, the heuristic biases search from the start to productive areas of the search space. Second, the correlation between approximate and full evaluations does not need to be high to direct search in some domain independent algorithms. Third, a strategy of sampling from the best solutions provides a means of using the external simulator to improve solution quality while evaluating only a small number of potential solutions.

7 Acknowledgments

This effort was sponsored by grants from the National Science Foundation under grant number IRI9503366 and the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0271. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Soraya Rana was supported by a National Physical Science Consortium fellowship awarded by NASA-SSC. The authors would also like anonymous reviewers for their comments and to thank Laura Barbulescu for her contributions to the ideas presented in this research.

References

[1] S. Bagchi, S. Uckun, Y. Miyabe, and K. Kawamura. Exploring problem-specific recombination operators for job shop scheduling. In L. Booker

and R. Belew, editors, *Proc. of the 4th Int'l. Conf. on GAs*, pages 10–17. Morgan Kaufmann, 1991.

[2] J. C. Beck, A. J. Davenport, E.M. Sitarski, and M.S. Fox. Texture-based heuristics for scheduling revisited. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-97)*, 1997.

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

[4] H. M. Cartwright and R. A. Long. Simultaneous optimization of chemical flowshop sequencing and topology using genetic algorithms. *Industrial and Engineering Chemistry Research*, 32(11):2706–2713, 1993.

[5] C. Cheng and S. F. Smith. Applying constraint satisfaction techniques to job shop scheduling. Technical report, The Robotics Institute, Carnegie Mellon University, 1995.

[6] D. Corne. Evolutionary Approaches to the Partition/Timetabling Problem. In N. Steele G. Smith and R. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms (ICANNGA-97)*, pages 281–288. Springer-Wein, 1998.

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

[8] Lawrence Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[9] U. Dorndorf and E. Pesch. Evolutionary based learning in a job shop scheduling environment. *Computers in Operations Research*, 22:25–40, 1993.

[10] Fred Easton and Nushar Mansour. A distributed ga for employee staffing and scheduling. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993.

[11] D. Elliman E.K. Burke and R. Weare. A hybrid genetic algorithm for university timetabling. In L. Eshelman, editor, *Proc. of the 6th Int'l. Conf. on GAs.* Morgan Kaufmann, 1995.

[12] David Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[13] Emma Hart and David Corne. The state of the art in evolutionary approaches to timetabling and scheduling.

http://www.dai.ed.ac.uk/daidb/people/homes/emmah/evonet.html, 1993.

[14] Emma Hart and Peter Ross. A heuristic combination method for solving job-shop scheduling problems. In *Parallel Problem Solving from Nature - PPSNV*, 1998.

[15] W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1995.

[16] P. Langley. Systematic and nonsystematic search strategies. In *Artificial Intelligence Planning Systems: Proceedings of the First International Conference*, 1992.

[17] S. Lin and B. Kernighan. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research*, 21:498–516, 1973.

[18] M. Nawaz, E. Emscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop problem. *OMEGA*, 11:91–95, 1983.

[19] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91:160–175, 1983.

[20] S. Rana, A. Howe, D. Whitley, and K. Mathias. A Genetic Algorithm Scheduler: Assessing the Contribution of Search, Heuristics and the Objective Function. *Engineering Design and Automation Journal*, 3(2):107–118, 1997.

[21] Soraya Rana, Adele Howe, Darrell Whitley, and Keith Mathais. Comparing heuristic, evolutionary and local search approaches to scheduling. In Brian Drabble, editor, *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, Menlo Park, CA., May 1996. The AAAI Press.

[22] Soraya Rana, L. Darrell Whitley, and Ronald Cogswell. Searching in the presence of noise. In H.M. Voigt, W. Ebeling, Ingo Rechenberg, and H.P. Schw efel, editors, *Parallel Problem Solving from Nature, 6*, pages 198–207, Berlin, Germany, 1996.

[23] Colin Reeves. Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem. *Journal of Evolutionary Computation*, 6(1):45–60, 1998.

[24] Kwang Ryel Ryu, Junha Hwang, Hyung Rim Choi, and Kyu Kab Cho. A genetic algorithm hybrid for hierarchical reactive scheduling. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, 1997.

[25] S. F. Smith and C. Cheng. Slack-based heuristics for constraint satisfaction scheduling. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993.

[26] Timothy Starkweather and Darrell Whitley. A Genetic Algorithm for Scheduling with Resource Consumption. In T. Gulledge G. Fanter and A. Jones, editors, *New Directions for Operations Research in Manufacturing*, pages 129–148. Springer-Verlag, New York, 1992.

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

[28] Gilbert Syswerda and Jeff Palmucci. 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, 1991.

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

[30] L. Darrell Whitley. 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, 1989.

Parent 1	a	b	c	d	e	f	g	h
Selected Elements		*	*	*		*		
Parent 2	h	g	**f**	e	**d**	**c**	**b**	a
Offspring	a	**f**	**d**	**c**	e	**b**	g	h

Figure 1: Syswerda's order crossover operator

	Baseline	Constructive	Genetic Algorithm		Random Exchange		Random Shift	
	Random	It. One-Samp	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init
Mean Time-At-Dock								
μ	456.88	407.01	392.49	395.38	400.14	399.17	406.59	396.84
σ	3.2149	1.6333	0.2746	0.7184	4.7493	3.4903	11.148	2.8455
Average Inventory								
μ	597123	362103	364080	354281	370458	350278	393364	352786
σ	10041	1620	1715	1354	20674	3026	52213	2272

Table 1: Performance Results on the Internal Simulator

	Baseline	Constructive	Genetic Algorithm		Random Exchange		Random Shift	
	Random	It. One-Samp	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init
Mean Time-At-Dock								
μ	447.11	397.43	397.48	391.20	439.43	426.05	436.35	426.73
σ	8.0304	1.6390	7.3680	2.9381	4.0479	1.6461	6.778	1.6933
Average Inventory								
μ	621148	390589	389605	384828	433241	411776	432340	399643
σ	28217	7872	9137	9038	20201	14383	38587	10669

Table 2: Performance Results on the External Simulator

	Mean Time-at-Dock		Average Inventory	
	Best	Rnd. Selected	Best	Rnd. Selected
Best	394.1	395.8	382343	382711
Mean	397.8	398.1	393834	391891
Worst	401.6	400.7	407232	409975
The Iterative One-Samp Results				

	Mean Time-at-Dock		Average Inventory	
	Best	Rnd. Selected	Best	Rnd. Selected
Best	388.4	387.3	377891	376704
Mean	391	390.1	381550	383595
Worst	394.7	393.6	393801	394499
The Hybrid Genetic Algorithm Results				

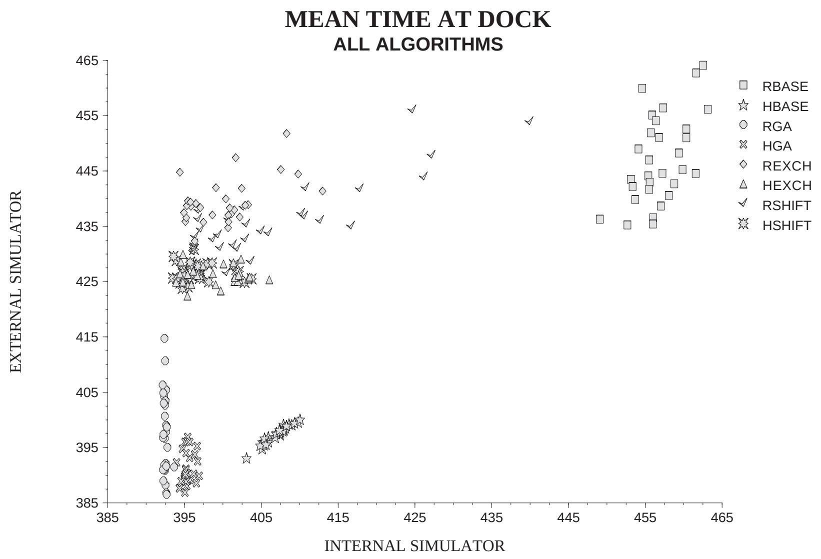
Table 3: Exploiting the External Simulator. The 50 best solutions and 50 randomly selected solution are compared for the hybrid genetic algorithm and Iterative One-Samp.

Evaluations		Mean Time-at-Dock		Average Inventory	
		Best	Rnd.	Best	Rnd. Selected
60K	Best	390.7	390.3	384119	379847
	Mean	391.8	392.1	391005	386996
	Worst	394.7	394.2	391236	329916
70K	Best	390.1	378.8	377956	375949
	Mean	390.9	389.9	388101	382507
	Worst	391.2	391.9	389773	384884
80K	Best	388.2	388.4	377167	378357
	Mean	393.3	393.5	381558	385103
	Worst	396.1	388.4	389155	393771

Table 4: Genetic Algorithm Performance on External Simulator

Proportion Sampled	Number of Samples	Uniform Noise
1.0	65536	0.9986
0.5	32768	0.9673
0.25	16384	0.8001
0.625	40960	-0.0780

Table 5: Correlation and Noise on a Parabolic function



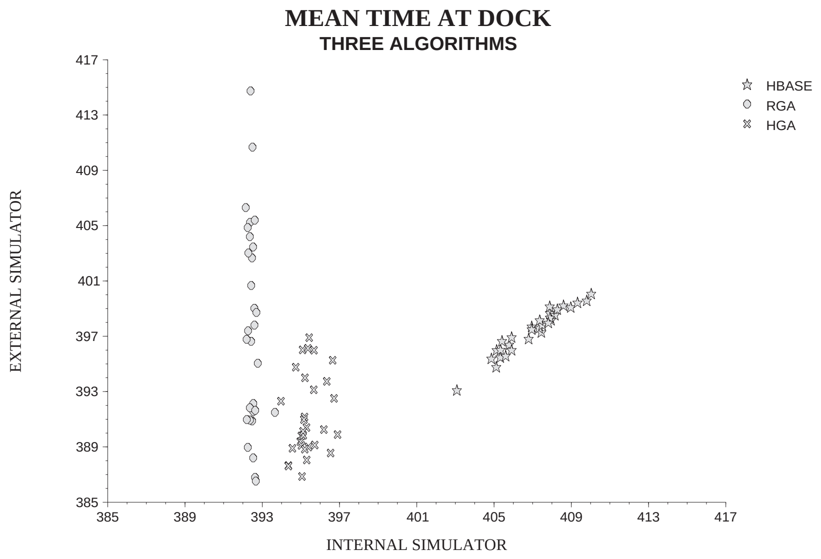
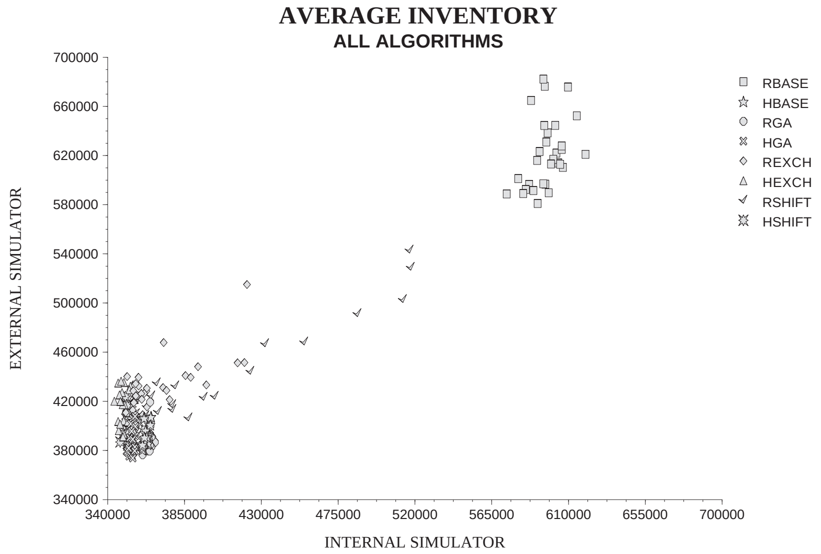


Figure 2: The external simulation values plotted against the internal simulation values for Mean time at dock.



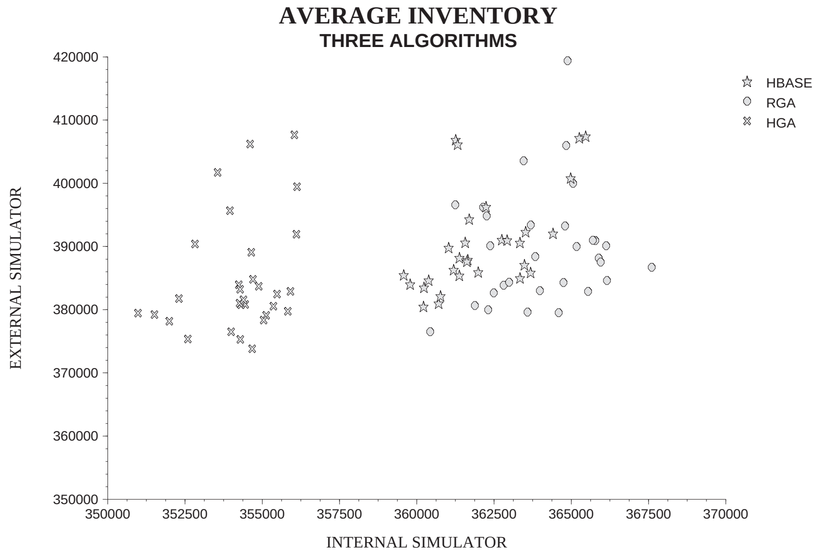


Figure 3: The external simulation values plotted against the internal simulation values for Running Average Inventory.

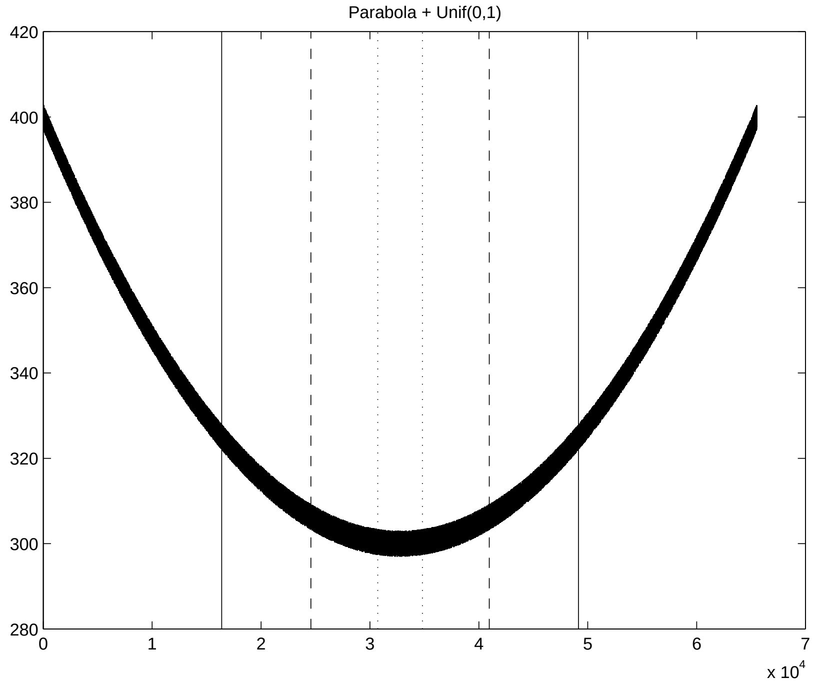
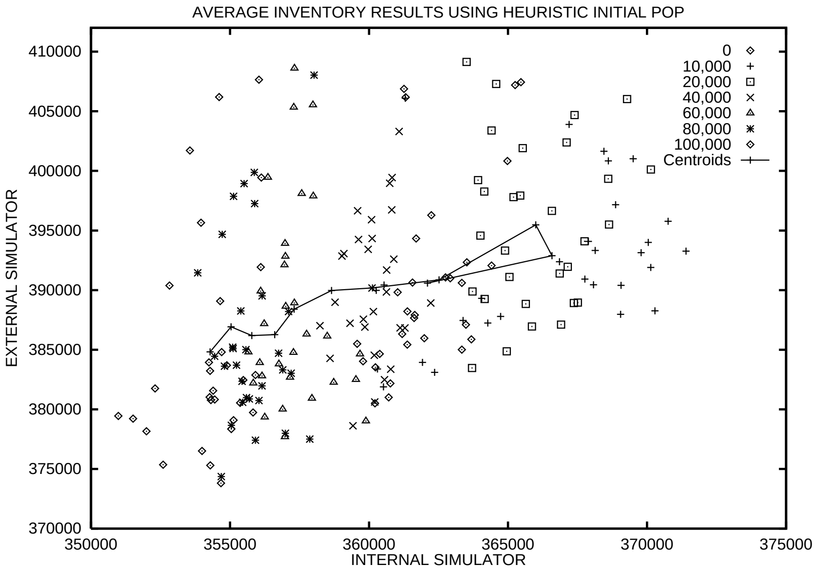


Figure 4: A parabola with uniform noise added. As one samples closer the global minima the evaluation functions (i.e., signal) is increasingly lost in the noise.



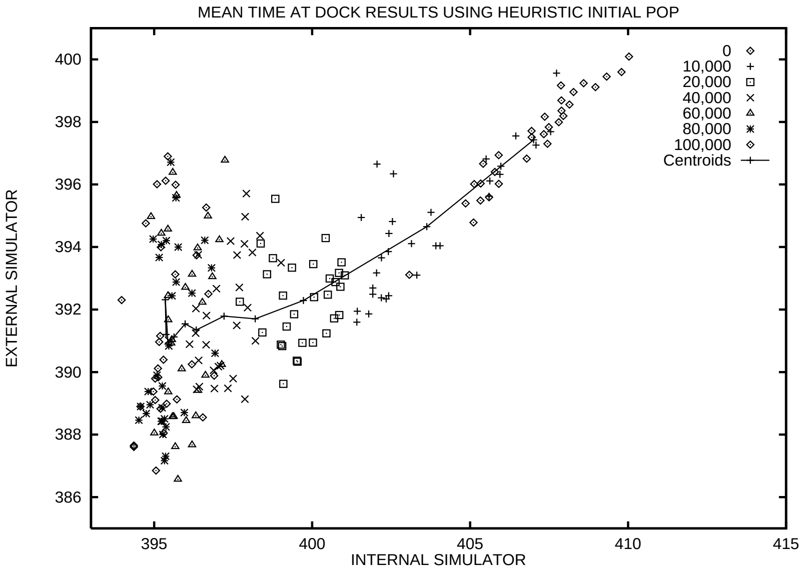


Figure 5: The external simulation values plotted against the internal simulation values for a single run of the hybrid genetic algorithm. The top graph is the average inventory and the bottom graph is the mean time at dock. The lines in the graphs correspond the the centroid of the point clouds computed for the corresponding sample intervals.

List of Figures

List of Tables