Comparing Heuristic Search Methods and Genetic Algorithms for Warehouse Scheduling

L. D. Whitley, A. E. Howe, S. Rana, J. P. Watson, L. Barbulescu
Computer Science Department
Colorado State University
Fort Collins, CO 80523
e-mail: {whitley, howe, rana, watsonj, laura}@cs.colostate.edu

Abstract—

We compare several techniques for scheduling shipment of customer orders for the Coors Brewing warehouse and production line. The goal is to minimize time at dock for trucks and railcars while also minimizing inventory. The techniques include a genetic algorithm, local search operators, heuristic rules, systematic search and hybrid approaches. Initial results show a hybrid genetic algorithm to be superior to the other methods. The evaluation function is a fast approximate form of a warehouse simulation. We also assess the sensitivity of the search algorithms to noise in an approximate evaluation function using a more detailed (and costly) simulation.

I. Introduction

Warehouse scheduling is the problem of sequencing requests for products (i.e., customer orders) so as to minimize the average time required to fill an order and the amount of product left in inventory. Product can be taken from the warehouse or directly from the production line. The time required to fill an order depends on whether product comes from the production line or whether product must be move out of inventory. Moving from inventory is slower, but only a small fraction of all possible product types are being produced at any point in time.

The problem is complex due to several factors. First, the search space is quite large: all possible sequences of all orders. Several hundred possible orders are considered for each schedule. Optimization involves several distinct performance measures, which can be inversely related. Evaluating a schedule is costly; a simulation is used to determine when enough of the needed product is available and how long it takes to move product from its starting location (warehouse or production line) to its transport (rail car or truck).

Search algorithms can be easily applied to warehouse scheduling. However, given the expense of schedule evaluation and the number of possible schedules, knowledge-poor search algorithms may be too costly. Heuristic-based methods provide a promising alternative, as they explicitly leverage domain knowledge to directly construct candidate schedules.

To assess the trade-offs between search and heuristic methods, we compared a variety of standard optimization techniques. These algorithms fall into three basic categories: stochastic search, domain-based heuristics, and hybrid systems. We investigated three stochastic search techniques: a genetic algorithm and two local search operators. We implemented our own order sequencing heuristic and used it in two different contexts for building schedules. First, it is used directly as a greedy construction procedure. Second, it is used as the heuristic component of Limited Discrepancy Search (LDS), a systematic AI search method that has performed well on synthetic test problems in the job shop scheduling domain. Hybrid approaches combine the stochastic search methods (i.e., the genetic algorithm and the local search methods search techniques) with the heuristics: heuristically generated solutions are used to initialize the stochastic search methods. We describe these approaches in more detail in Section III.

We compared the scheduling methods using three performance metrics: schedule makespan, average time to fill an order, and average total inventory (see Section II). The values for the performance metrics were obtained from a coarse-grain simulator we developed. We also developed a more detailed, fine grain simulator. The coarse-grain simulator was used as the objective function in our experiments because of the expense of the more detailed simulation. While run-time was substantially reduced, the loss in granularity necessarily produced some inaccuracy in the performance measurements. We refer to the coarse-grained simulation as the "internal simulator" (since it is internal to the optimization procedure) and the detailed simulator as the "external simulator."

We found that methods combining stochastic search with heuristic initialization significantly outperform heuristic-based methods, given equal amounts of computation. Section IV provides detailed results. Furthermore, the globally motivated search of the genetic algorithm outperforms the local search algorithms. However, heuristic-based methods are not without merit; hybridization of the stochastic search algorithms with heuristic-based initialization further improves performance.

A key features of the Limited Discrepancy Search (LDS) is the use of backtracking to undo some number of heuristic decisions. We also find that the backtracking of LDS

fails to overcome the deficiencies of the greedy heuristic construction procedure for this particular domain.

The total number of evaluations is limited to 100,000 in our experiments. Thus, our conclusions may not generalize to situations where the number of total evaluations is different. We limited total evaluations for two reasons. First, this was done in order to limit total search time to a reasonable amount of time. Second, the largest improvements in the objective function appear to occur within the first 100,000 evaluations. Improvement is asymptotic after this point. Given that the fast internal simulator is weakly correlated with the detailed external simulator, it is unclear that further optimization with respect to the internal simulator actually translates into further improvement in the external simulator. We are currently investigating this question further.

II. Coors Warehouse Scheduling

During a 24 hour period, Coors fills from 150 to 200 orders where each order may include different products. The manufacturing plant produces 500 different products. Orders are filled from either the production line or inventory. For our simulations, we are given a fixed, predetermined production schedule by Coors; the test data is for 525 orders over approximately a three day period.

Orders are shipped on either truck or rail. The manufacturing plant has 39 truck docks and 19 rail docks. Orders are separated into truck and rail orders before scheduling begins. However, both truck and rail orders compete for product. Once loading of an order begins, the dock assigned to that order is unavailable until the order has been completely filled. Thus, the order sequence is sensitive to the fixed production schedule, the contents of inventory, and the availability of the docks.

Schedules are evaluated by simulating the processing of orders from the manufacturing plant. We use two different simulators. The external simulator is detailed, but slow. In contrast, the internal simulator is a coarse-grained, fast simulator that saves time by substituting fixed time estimates for portions of the simulation and modeling at a larger time step. These differences lead to dramatically different execution times: 50 milliseconds for the internal simulator versus 90 seconds for the external simulator. Therefore, we use the internal simulator as the evaluation function and the external simulator to verify the quality of the solutions obtained using the internal simulator with various search methods. As discussed in Section IV, we were only able to achieve weak correlation between the two simulators.

The simulators compute two metrics for use in the objective function: mean-time-at-dock and average-inventory. The objective function normalizes these metrics by their means and standard deviations over a set of solutions. The formula, developed using methods reported in [1], is 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. Another commonly used performance metric is the makespan. However, the problem with minimizing the makespan is that the production schedule imposes a limit on the minimum makespan (4311.15 minutes) due to a product bottleneck. Thus, many schedules that are "optimal" in terms of makespan are very different in terms of mean time at dock and average inventory. In practice, shipping never stops, and therefore minimizing the mean-time-at-dock is more effective than minimizing the makespan. The goal is to have as many orders as possible filled and off the docks as soon as possible, even if the last order cannot be filled before the 4311 minute mark.

III. Approaches

The techniques we compared were: a genetic algorithm, local search operators, heuristic rules, systematic search, hybrid approaches, and random sampling. Genetic algorithms provide a globally motivated search. Local search closely fits the model of schedule construction in which schedules are gradually improved. Heuristics incorporate domain knowledge about constraints and therefore are used to reduce the search space. As a systematic AI search method, we chose Limited Discrepancy Search which has been shown to perform well on benchmark scheduling test problems. Random sampling works well for search spaces with a high density of good solutions and, in general, can be used to characterize worst case behavior. The size of the search space and the time constraint of rapid scheduling precluded the use of meta-heuristic methods such as TABU search or simulated annealing.

A. Genetic Algorithm

GENITOR [2] is what has come to be called a "steady-state" genetic algorithm [3](GA). Steady-state GAs, unlike traditional GAs, produce a "one-at-a-time" replacement, where only one pair of chromosomes reproduces during a generation, and only one offspring is generated. The offspring will replace the worst (least fit) chromosome in the population, and therefore the best string found so far will always remain in the population. The reproduction opportunities are allocated using the rank of the parents 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.

A special operator is needed to recombine two "parent" permutations and generate a new permutation combining relevant features of the two parents structures. Syswerda [4] notes that operators can attempt to preserve either the position, relative order or adjacency of the elements in the

Parent1	a	b	c	d	e	f	g
Select list	*	*				*	*
Parent2	c	f	a	d	b	g	e
Crossover	f	a	c	d	e	b	g

Fig. 1. Syswerda's order crossover operator

parents. Previous experiments [5] suggest that Syswerda's order crossover operator (which is designed to preserve relative order) performs the best for the warehouse scheduling application. This makes sense, given that the relative order in which orders are filled affects the availability of product for later customers.

A subset 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 such 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. Figure 1 illustrates this recombination process. A more detailed description and analysis of the operator is given by Whitley and Yoo [6].

B. Local Search

The local search algorithms apply 2-opt or random swap operators to iteratively improve the schedule until no further changes can be made. The 2-opt local search operator is classically associated with the Traveling Salesrep Problem (TSP). For the TSP, the permutation actually represents a Hamiltonian Circuit. 2-opt uses two positions in the permutation and reverses the substring between the two positions, with the goal of eliminating crossings in the circuit represented by the permutation. The operator is minimal for the TSP, in the sense that it only alters two edges in the graph and one cannot change only one edge and still have a circuit.

However, if relative order is more important than adjacency, then 2-opt is not a minimal operator with respect to the warehouse scheduling problem. The relative order for all the elements in the reversed segment is changed by 2-opt. A less disruptive operation is to simply swap two elements in the permutation. We will refer to this as the *swap* operator. All the elements, except for the two swapped ones, remain unchanged.

The number of 2-opt moves or swaps over all pairs of n elements is

$$\frac{n(n-1)}{2}.$$

Given the limit on the total number of evaluations, a steepest descent is impractical (the $\mathcal{O}(n^2)$ cost is more than the allotted 100,000 evaluations for our problems); we employ a next-descent strategy where each improving move is accepted instead. Because the size of the local search neighborhood is large, the operators are applied to random pairs of positions within the permutation rather than systematically examining all pairs.

C. Heuristic Rules

Constraint-based scheduling approaches yield smaller search spaces by exploiting domain knowledge about constraints. Given that this application does not include hard constraints, a simple alternative is to develop heuristics based on soft constraints. The heuristics were derived from analyzing the actual schedule used by Coors to process the set of orders.

For any particular order, two time intervals can be generated based on the amount of product needed from inventory and from the production line. The two time intervals are denoted by $duration_{Inv}$ and $duration_{Pl}$. These time intervals represent the wait time for the product to be loaded from inventory ($duration_{Inv}$) and the wait time for the product to be produced and loaded from the production line ($duration_{Pl}$). The internal simulator will always consider the $duration_{Inv}$ to start at the current time of simulation for the processed order. Moreover, due to a minimum time required to move products to the dock assigned to the processed order, the $duration_{Inv}$ is always non-zero. In order to reduce computation time in the internal simulator, when loading an order, product will be drawn from inventory first. Only four relationships between the two intervals are possible (see Figure 2):

- All needed product can be drawn from the inventory ($duration_{Pl} = 0$)
- The two time intervals do not overlap
- The two time intervals partially overlap
- The two time intervals completely overlap

Three functions were defined to compute the rank for a given order; the function to be used is determined by the relationship between the two intervals for that order. An order will be filled only from inventory if it is small (< 2500 units for rail orders and < 300 units for truck orders). If the intervals of an order do not overlap or partially overlap, the order is ranked based on the number of products required multiplied by the difference in interval completion times. Preference is given to large orders that can be filled both from inventory and production line, in parallel. If the intervals of an order completely overlap, the rank is computed based on the number of products required to fill the order, giving preference to large orders.

The heuristics were designed to favor large orders for which the two time intervals overlap. For each unscheduled order, the rank is computed; the best order to be processed will be the one with the lowest rank. Ties are common. Thus, a random permutation is generated to use in conjunction with the heuristics to break ties.

D. Limited Discrepancy Search

Limited Discrepancy Search (LDS) is a systematic search technique that exploits heuristic information in tree search problems [7]. LDS is effective if a good heuristic is available. A *discrepancy* defines a decision point where the heuristic is not followed (the chosen successor

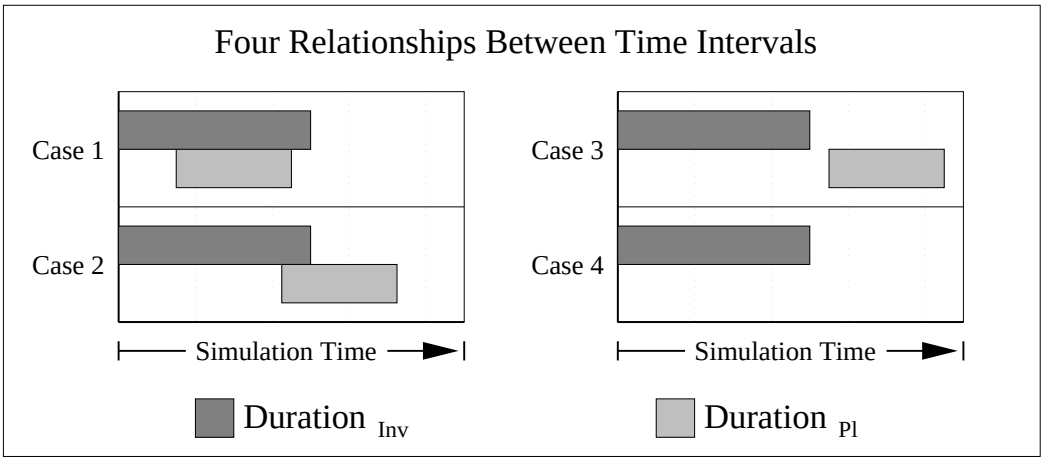

Fig. 2. Four relationships for the two intervals $duration_{Inv}$ and $duration_{Pl}$

is not the one selected as "the best" by the heuristic). LDS iteratively searches the space with a limit on the number of discrepancies allowed on any path. In the first iteration, no discrepancies are taken, i.e., the heuristic is followed. In general, in the i^{th} iteration at most $i-1$ discrepancies are taken along any path.

The problem with applying LDS to the warehouse scheduling problem is the branching factor. Indeed, allowing only one discrepancy, there are n possible choices of the first element in the schedule for the initial state (no discrepancy or $n-1$ possible choices for one discrepancy). If in the initial state the heuristic was followed, then in the next state, there are $n-1$ possible choices for the second element in the schedule, and so on. With only one discrepancy, there will be

$$\frac{n(n-1)}{2}+1$$

possible paths.

The depth of the search tree is equal to the maximum number of decisions to be made; if in each state only one order is scheduled, then the depth of the tree is equal to the total number of orders to be scheduled. For the problem we solved, it is possible to schedule more than one order in a particular state. This resulted in trees with an approximate depth of 350.

E. Hybrid Heuristic and Search

The genetic algorithm and local search operators are usually initialized with random permutations. An alternative is to use the heuristic rules in conjunction with a permutation to create a schedule. This process allows the heuristic solutions to be used to seed the initial solutions for all of the stochastic search methods (local search and genetic algorithm).

F. Random Sampling

Random sampling produces random schedules by simply generating random permutations. Evaluating a set of random permutations estimates the density of good solutions and the variance in the search space of solutions. Random sampling is included as a baseline of performance.

IV. Experiments

We pose and answer three empirical research questions regarding the performance of various algorithms as applied to the Coors warehouse scheduling problem. First, we examine algorithm performance with respect to each other, to randomly generated solutions, and to a real Coors solution. Next, we investigate whether heuristic initialization confers any advantage to the performance of the search algorithms. Finally, we assess the performance bias associated with the use of the coarse-grained internal simulator.

A. Methodology

We provide answers to these questions by measuring the performance of the five (two heuristic and three search) algorithms under various experimental conditions. We compared algorithm performance on both the internal and external simulators. External simulation runs were performed on the solutions obtained with the internal simulator (the external simulator assesses the relative accuracy of the internal simulator). For the three stochastic search algorithms, we also measured performance under both forms of initialization: random and heuristic. For each set of experimental conditions, we report the mean and standard deviation of 30 independent trials. For each trial, the mean-time-at-dock, average-inventory, and makespan of the best solution obtained are recorded.

In order to produce fair comparisons, we allocated to each algorithm an evaluation limit which caused the algo-

	Baseline	Heuristic		Genetic Algorithm		2-opt		Random Swap	
	Random	Greedy	LDS	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init
Mean-Time-At-Dock									
μ	459.90	407.68	406.21	392.49	395.39	423.11	398.71	400.11	399.03
σ	3.2386	1.2684	1.4523	0.2792	0.7290	5.6136	3.1036	4.8298	3.4629
Average-Inventory									
μ	627306	363160	362014	364050	354271	530446	394383	370718	350228
σ	16041	1332	1341	1739	1376	20088	12424	20987	3070
Makespan									
μ	5378.38	4318.00	4318.00	4318.00	4318.00	4762.69	4318.00	4318.69	4318.00
σ	119.644	0.000	0.000	0.000	0.000	101.368	0.000	2.792	0.000

TABLE I

PERFORMANCE RESULTS ON THE INTERNAL SIMULATOR

	Baseline	Heuristic		Genetic Algorithm		2-opt		Random Swap	
	Random	Greedy	LDS	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init	Rnd. Init	Heur. Init
Mean-Time-At-Dock									
μ	451.91	398.23	397.95	397.37	391.04	457.15	425.61	439.51	426.06
σ	7.7196	1.3550	1.2098	7.4734	2.8439	6.3953	2.0711	4.0977	1.6718
Average-Inventory									
μ	645428	389539	390081	389475	384078	539864	426707	433411	412366
σ	28497	5278	4955	9272	8229	23524	17872	20537	14252
Makespan									
μ	4631.74	4311.15	4311.15	4311.81	4311.15	4389.37	4311.15	4314.31	4311.15
σ	113.312	0.000	0.000	3.406	0.000	81.797	0.000	10.823	0.000

TABLE II

PERFORMANCE RESULTS ON THE EXTERNAL SIMULATOR

rithms to consume roughly equal amounts of computation time. Both the greedy and LDS heuristic algorithms run roughly 10 times slower than the simulation of a random order permutation. These factors are considered below in allocation of evaluation limits to the various algorithm trials.

A maximum of 100K evaluations was allowed for each genetic algorithm trial. Population size was fixed at 500 elements. The dynamic objective function parameters (μ_{ai}, σ_{ai}, μ_{mt}, and σ_{mt}) were updated after every generation using the current population, with the initial values computed from the initial population.

Similarly, we allowed a maximum of 100K evaluations for trials using the random swap algorithm. A single complete pass of the 2-opt algorithm requires 118K evaluations for the warehouse problem; we used this limit for each trial involving 2-opt. The objective function parameters μ_{ai} and μ_{mt} were initialized to 0, and σ_{ai} and σ_{mt} were initialized to 1. These parameter values were updated every 100 evaluations using the 100 most recent solutions.

For the random sampling and both heuristic (greedy and LDS) algorithms, the best solution among 100K and 10K trials, respectively, is reported. Objective function parameters are computed using the method defined for the 2-opt and random swap algorithm trials.

Because of the large branching factor at early stages in the search (> 250 child nodes), we limit the number of child nodes explored for a single discrepancy to 30; empirically, we found no performance improvement by considering child nodes beyond this point. Given a limit on the number of evaluations, we identified two design options for the LDS algorithm trials. The first option involves allocating all 10K evaluations to a single invocation of the LDS algorithm; this allows exploration of a larger number of discrepancies. The second option involves multiple invocations of the LDS algorithm, with a reduction in the limit of trials allocated to each instance. While fewer discrepancies are explored with the second option, it empirically provides the best performance. For the LDS algorithm trials, we used 100 invocations of the LDS algorithm (on different initial solutions) with a limit of 100 evaluations each.

B. Results

Tables I and II summarize the performance of the various algorithms when using the internal and external simulators for solution evaluation, respectively. The first column represents the performance of random search; we use this as a baseline performance measure. The next two

columns (greedy and LDS heuristic search) represent the performance of the heuristic-based schedule construction procedures. Finally, the remaining columns document the performance of the three stochastic search algorithms investigated.

We based all statistical comparisons of the algorithms on the data obtained with the external simulator, as this simulator provides the most accurate estimate of performance in the real warehouse environment. First, we examined the performance of the algorithms with respect to two baselines: random search and a solution developed by Coors personnel. 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. Note that the Coors solution is better than the best solution produced by random search.

With the exception of the randomly initialized 2-opt algorithm, the data in Table II demonstrates the ability of each of the algorithms to produce better solutions than the Coors solution. Furthermore, this improvement is often dramatic, especially when comparing *average-inventory* and *makespan*.

Next, we analyzed the performance differences between the various heuristic/search algorithms. First, with the exception of the randomly initialized 2-opt algorithm, a series of one-tailed t-tests indicated a statistically significant ($p < 0.01$) performance difference between each of the heuristic/search algorithms and random search. The randomly initialized 2-opt solutions are not significantly different from the random solutions in terms of *mean-time-at-dock*. With the noted exceptions, all of the heuristic/search algorithms outperform both solutions generated by random search and the solution generated by Coors. The low variance in the attained solution performance metrics shown in Tables I and II demonstrates the high reliability of the algorithms.

We also ran one-way ANOVA's with search algorithm as the independent variable and performance metric (either *mean-time-at-dock* or *average-inventory*) as the dependent variable. The tests indicated a significant effect ($p < 0.001$) of the choice of search algorithm on the performance metrics. Further analysis indicated, via a series of one-tailed t-tests, that the genetic algorithm produced significantly ($p < 0.01$) better solutions than the 2-opt or random swap local search algorithms. Furthermore, surprisingly the tests showed no difference between the two local search algorithms. We had expected the swap operator to be significantly better.

Table II also demonstrates the apparent benefit to heuristic initialization of the search algorithms. Two one-way ANOVA's with initialization method as the independent variable and performance metric (either mean-time-at-dock or average-inventory) as the dependent variable confirms this observation, showing a significant main effect ($p < 0.001$). One-tailed t-tests demonstrate that in all cases, heuristic initialization significantly ($p < 0.01$) improves search algorithm performance.

Inspection of Table 2 also indicates that LDS (systematic backtracking in conjunction with a heuristic) fails to yield any statistically significant improvement in performance over greedy heuristic search. As verified by a series of one-tailed t-tests, LDS is unable to overcome enough of the heuristic bias to yield solutions competitive with the search algorithms.

Finally, we assess the predictive capability of the internal simulator on the true solution quality, as dictated by the external simulator. Recall that the search algorithms operate through feedback, via the objective function, from the internal simulator. The results in Table II are then produced by running the external simulator on the obtained solutions. A comparison of Tables I and II demonstrates a clear discrepancy between the solution quality obtained by the different simulators. This result is not surprising in light of the weak correlation between the two simulators, as discussed in Section II.

V. Conclusions

The results presented here suggests that the genetic algorithm generates the best results for this problem compared to other methods we examined. These include local search operators as well as heuristic methods, including the use of Limited Discrepancy Search. Using heuristics to initialize the population used by the genetic algorithm further improved the results. The results are generated using a fast internal simulator as an evaluation function. Evaluation using a more detailed external simulator indicates that the best solutions are still those found by the genetic algorithm. One unusual factor in our experiments is the limitation to 100,000 evaluations. It is possible that using a different number of evaluations could alter the results. As noted, a critical question and a topic of on-going research is the use of additional evaluations and how the additional evaluations affect the more detailed evaluations produced by the external simulator.

VI. Acknowledgements

This effort was sponsored by 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.

References

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

[2] L. Darrell Whitley, "The genitor algorithm and selective pressure: Why rank based allocation of reproductive trials is best," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989.

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

[4] Gilbert Syswerda, "Schedule optimization using genetic algorithms," in *Handbook of Genetic Algorithms*, 1991.

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

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

[7] W. D. Harvey and M. L. Ginsberg, "Limited discrepancy search," in *Proceedings of the 14th IJCAI*, 1995.