Proceedings of the 2008 Winter Simulation Conference
S. J. Mason, R. R. Hill, L. Mönch, O. Rose, T. Jefferson, J. W. Fowler eds.

THE IMPROVED SWEEP METAHEURISTIC FOR SIMULATION OPTIMIZATION AND APPLICATION TO JOB SHOP SCHEDULING

George Jiri Mejtsky

Simulation Research
314 Steeplechase Drive
Exton, PA 19341, USA

ABSTRACT

We present an improved sweep metaheuristic for discrete event simulation optimization. The sweep algorithm is a tree search similar to beam search. The basic idea is to run a limited number of partial solutions in parallel and to search for solutions by searching the partial solutions. Traditionally, simulation optimization is carried out by multiple simulation runs executed *sequentially*. In contrast, the sweep algorithm executes multiple simulation runs *simultaneously*. It uses branching and pruning simulation models to carry out optimization. We describe new components of the algorithm, such as backtracking and local search. Then, we compare our approach with 13 metaheuristics in solving job shop scheduling benchmarks. Our approach ranks in the middle of the comparison which we regard as a success. The general nature of tree search offers a large array of sequential decision applications for the sweep algorithm, such as resource-constrained project scheduling, traveling salesman, or (real-time) production scheduling.

1 INTRODUCTION

We present an improved sweep algorithm; then we test and compare our algorithm with 13 metaheuristics. The sweep algorithm, simulation-based metaheuristic, is a tree search similar to beam search (Zhou and Hansen 2005). The basic idea is to run a limited number of partial solutions in parallel (population-based algorithm) and to search for solutions by searching the partial solutions.

In simulation optimization, the goal is to find a set of input parameters (decision variables) that minimizes an objective function. The objective function is not available directly, but it is estimated by simulation. We focus on deterministic discrete event simulation which runs on a single processor computer.

Today, the most attractive simulation optimization methods are metaheuristics, such as tabu search, simulated annealing, or genetic algorithm. Metaheuristics are algorithms which guide a series of simulation runs to produce better solutions than simple heuristics can generate.

A metaheuristic algorithm, a separate module from a simulation model, caries out (designs and evaluates) multiple simulation runs executed *sequentially*. In this traditional simulation optimization, the simulation model is used as a black box function evaluator. For recent review of literature on simulation optimization, see Fu, Glover, and April (2005) or Henderson and Nelson (2006).

In contrast, the sweep algorithm executes multiple simulation runs *simultaneously*, and it uses the white box approach. Apart from discrete events, called simulation events, which describe dynamics of a simulated system, the sweep algorithm introduces two new events, called optimization events. These events carry out the optimization part of the simulation optimization (Mejtsky 1986a, Mejtsky 1986b, and Mejtsky1986c). The two optimization events are branching and pruning events. During a simulation run, when a model (experiment) encounters a decision point with k options, a branching event is triggered. In the branching event, the model (parent) spawns k new models (children). Each child model (complete copy of the parent model) picks a different option and continues in simulation run. However, the parent model ends its run.

The search process of the basic sweep algorithm starts by running a single simulation model – root model. As simulation time advances, branching events are triggered; therefore, more and more models run simultaneously. When the population size (the number of models) reaches an upper limit (CEILING), a pruning event is triggered to reduce the overpopulation. In the pruning event, the entire population is sorted by a heuristic pruning function, and only several best individuals (lower limit, FLOOR) are allowed to continue running. Remaining individuals are eliminated from the population (a Darwinian process of purging weaker ones). Notice the similarity between a pruning event and the screening (subset selection) approach of ranking and selection procedures.

After the population size drops (from CEILING to FLOOR) in a pruning event, the population grows again

until the next pruning event or the end of simulation is reached. This evolution of population, repeated application of spawning (reproduction) and pruning (survival of the fittest), we call a sweep. In the sweep, only one global event calendar is shared by all models, which ensures synchronization of the simulation time in each model. For a detailed description of branching and pruning events, see Mejtsky (2007). For another approach to simultaneous simulation optimization, see Weinberger (1982).

During a sweep, the bias of search process towards diversification (global exploration of the search space) is strongest at the beginning of the sweep; however, as the simulation time progresses, the bias is gradually shifting towards intensification (exploitation of promising regions by local search). This shifting bias from diversification to intensification is similar to the shifting bias in simulated annealing during cooling temperature T.

We describe a search framework of the improved sweep algorithm in Section 2. The two main enhancements are (1) backtracking (Section 2.1) to increase diversification and intensification, and (2) local search (Section 2.2) to search in the neighborhood of a solution for an improved solution. In order not to duplicate searches of already searched regions of a solution space, we implemented a nodeset savings procedure (Section 2.3). The purpose of the sweep signature concept (Section 2.3) is to increase the quality of the sweep population (relative to populations of other sweeps) by eliminating weaker individuals, and possibly, eliminating the entire weak population.

We tested the algorithm on standard benchmark problems from job shop scheduling (JSS); the results are presented in Section 3. We describe a JSS problem by a set of jobs with ordered operations to be processed on a set of machines. Each machine can process only one operation at a time, and each operation has fixed time duration. The objective is to minimize the duration of the longest job in the schedule (minimizing makespan).

Our test results are compared with 13 metaheuristics in Section 4. The improved sweep algorithm ranks in the middle of the comparison. Notice that only the sweep algorithm uses simulation for modeling, while other metaheuristics use disjunctive graph. To the best of our knowledge, we are not aware of any traditional simulation optimization algorithm solving the JSS benchmarks with better results than the sweep algorithm. We outline ideas for future research, such as a marriage with dispatching rules, in Section 5.

In conclusion, the research contribution of this paper is in developing the sweep algorithm – the first competitive simulation-based metaheuristic. The sweep algorithm shows that when simulation is used with the white box approach (not as a black box function evaluator) then simulation is as effective for optimization as disjunctive graph. The sweep algorithm (1) is suitable for solving sequential decision problems, (2) is quite competitive with other metaheuristics using disjunctive graph, (3) is suitable for real-time production scheduling, and (4) can evaluate dispatching rules when the rules are options in branching events.

2 ALGORITHM

The search process of the improved sweep algorithm progresses through three steps, called search framework, where an output of one step becomes an input of the next step. In step 0, the search space is divided into disjoint zones. In step 1, the zones are separately searched for good solutions. In the last step, a local search is applied to improve the solutions found in the most promising zones.

In step 0 (search space partitioning), the global partitioning of search space is carried out by running a sweep with the root model (input of this step, ancestor of all other models) as a seed population. This sweep is special: no pruning events are allowed. The purpose of the sweep is to branch until the sweep population size reaches the required number of zones, algorithm's parameter. When the number is reached, the sweep stops (step 0 is complete). The sweep population, one model for each zone, represents zone seeds (output of this step, zone root models) for zone searches. All nodes expanded in this step form a set called zone-creation nodeset. This special short sweep "divides" the decision tree into disjoint subtrees, creating one subtree for each zone.

In step 1 (zone search), one by one, the zone subtrees are searched by a sweepset search (that is sweep with backtracking as described in Section 2.1). For each zone, the sweepset search runs with a different model from the zone seeds (input of this step). This model (zone root model) is the seed population for launching the first sweep of the sweepset. The algorithm maintains a list (output of this step, zone elite) of the elite solutions found during the zone sweepset searches.

It is tempting to continue with the iterative search process of nested partitioning (Shi and Ólafsson 2000) as started in step 0 and 1. In step 0, the whole search space is partitioned into zones. In step 1, the zones are evaluated, and the most promising zones are selected for partitioning into subzones and subsequent evaluating the subzones in the next iteration. However tempting it is to continue with nested partitioning in the following steps, we are more intrigued by exploring the potential of the sweep's local search (Section 2.2).

In step 2 (local search), the zone elite (input of this step) are selected for the local search by the best first rule. For each zone elite (initial solution), its neighborhood is searched for improvement. The local search works as an iterative improvement process. In each iteration, the neighborhood of an elite solution found in previous iterations is searched for solution improvement by the sweepset search. The algorithm keeps track of the best solution found thus far (global best, *alpha* solution).

The search framework utilizes four new components – backtracking, local search, nodeset savings, and sweep signature – which are described next.

2.1 Backtracking

Backtracking is a traditional search approach. During a search, if an alternative does not work, the search backtracks to a decision choice point, a place with different alternatives, and tries the next alternative. For a backtracking example, see beam search with backtracking in Zhou and Hansen (2005).

In the basic sweep algorithm, there is only one single sweep producing a solution at the end. If the solution is not optimal, then it means that the single sweep eliminated a model (partial solution) leading to the optimal solution. The elimination of a potential optimal solution (inadmissible pruning) could occur only in a pruning event, where weaker models are eliminated from the search.

Therefore, to increase the chance of finding an improved solution, we modified pruning events by saving the weaker models. This allows the algorithm, after a sweep ends and a solution is found, to go back (backtrack) to a saved model (or models) and to run the next sweep with the saved model(s) as the seed population. This idea of backtracking is similar to the reheating idea of simulated annealing.

In the new pruning event, the sweep population is placed in a queue, called island, and sorted by a pruning function. Only the FLOOR best performing individuals of the sweep population are removed from the island and allowed to continue in the sweep. The rest of the sweep population stays on the island.

During a sweep, a number of pruning events can arise, and each pruning event creates an island with population – islanders. Therefore, when the sweep ends, the sweep algorithm backtracks to an island, picks up one or more individuals (seed population for the next sweep) from the islanders, and starts running the next sweep from the simulation time of the pruning event which created the island (island creation time, island time).

There are a number of schemes (heuristics) for selecting the seed population of the next sweep. We implemented two alternatives as an algorithm's parameter:

- Find the earliest island (the island with the lowest value of island time, FIRST ISLE) and select several top performers from that island; or
- Find the latest island (the island with the highest value of island time, LAST ISLE) and select several top performers from that island.

For fast execution, all models are stored in the CPU fast memory. This is in contrast to the implementation of the basic sweep algorithm where only one model was in the memory, and the rest was stored in a direct-access file. Due to memory limitation, there are limits for the number of islands and the number of models (sweep population plus islanders on all islands plus others). Therefore, when anyone of the two limits is being approached, the following pruning events do not save weaker models on islands anymore. The weaker models are eliminated forever from the search. Some memory space for new models needs to be always reserved so that branching can go on, and therefore, the sweep can continue.

During a sweep, it encounters (except the first sweep) a number of inhabited islands created by pruning events of previous sweeps. The sweep is like a ship on her voyage passing many islands. When the ship passes an island, it has two alternatives (algorithm's parameter): (1) do not stop on the island and continue on its voyage; or (2) stop on the island, pick the best individuals, and continue on its voyage.

When the ship stops on the island, (1) everyone leaves the ship and enters the island, (2) all individuals on the island (ship population and islanders) are sorted by the pruning function, and (3) the best performing individuals board the ship and depart the island. On departure, the ship population size is the same as on arrival, except in the case when on arrival the size is smaller than FLOOR. In this case, the size on departure is equal to FLOOR.

The sweep algorithm with backtracking executes a sequence of sweeps, called sweepset, until there is no more seed population for the next sweep, or until a limit for the number of sweeps (algorithm's parameter) in the sweepset is reached. The algorithm maintains a list (sweepset elite) of the best solutions found during the sweepset search. At the end of every sweepset search, all islands are emptied (islanders are eliminated) and destroyed; therefore, memory is freed, and the next sweepset search can create new islands to inhabit.

2.2 Local Search

The sweep algorithm is a tree search. The basic idea of tree search is to conceptualize an optimization problem as a decision tree. Each decision choice point – a node – corresponds to a partial solution – a value of an input parameter. From each node, several new branches emanate, one branch for each decision choice (option). This branching process continues until leaf nodes, which cannot branch any further, are reached. These leaf nodes are solutions to the optimization problem (values of all input parameters are known). The starting node is called the root node, ancestor of all other nodes.

A solution is defined by a unique sequence of nodes (a path through the tree) starting with the root node and ending with a leaf node. In a node of the solution, an option is selected (partial solution). All remaining non-selected options of the node form a neighborhood of the node. Our lo-

cal search explores the neighborhoods of the nodes – the neighborhood along the path (path neighborhood) of a solution. The neighborhood of a solution is defined by all solutions which can be generated by exploring the neighborhood along the path of the solution.

The purpose of our local search is to search for an improved solution in the neighborhood of a solution. Every node in a tree can be seen as the root node of the subtree rooted at that node. Likewise, at any point of simulation time, a model represents the root model of the subtree rooted at that model.

(Note: One can describe a sweep in terms of the nested partitions method. The basic concept of nested partitioning is to iteratively partition the most promising regions (trees) selected at the previous iteration into disjoint subregions (subtrees) which will be evaluated in the next iteration. In a sweep, each pruning event evaluates subregions (the partitioned regions, models) from the previous branching phase and selects the most promising subregions for partitioning in the next branching phase.)

Our local search explores subtrees of a path neighborhood by the sweepset search. The root model of a subtree is used as a seed population for the sweepset search of the subtree.

To start the sweepset search, a seed population is needed to launch the first sweep of the sweepset. The local search supplies the seed population from the path neighborhood of a solution by running a simulation model, called lead model, according to the solution. The lead model follows the path of the solution. When the lead model encounters a decision point – a node – with k options, a branching event is triggered. In the branching event, the lead model selects for itself the option according to the solution and spawns $(k - 1)$ new models. Each new model picks a different option from the remaining $(k - 1)$ options – the node neighborhood. The new models are spawned only if they are needed as the seed population. As the seed population is being spawned (that is, the first sweep of a sweepset search is being launched), the sweep population and the lead model run in parallel. The lead model only generates the seed population and does not participate in pruning events of the first sweep.

There are a number of schemes, neighborhood structures, for selecting the seed population from the neighborhood along the path of a solution. One scheme can select one or more options from the neighborhood of a randomly selected node. In such a case, the lead model would advance along the path, passing nodes without spawning models, until the randomly-selected node is encountered. At this branching event, one or more required models would be spawned. Imagine a scheme selecting just one option from every other node along the paths of two solutions (in a local search of two solutions), then two lead models are needed to run and generate the desired seed population. Notice that the sweep population and the lead model need not run in parallel. The generation of the seed population can be run first, followed by the sweepset search.

We now describe our implementation of the local search. The local search is an iterative improvement process. It takes solutions one by one from the zone elite and iteratively searches the neighborhood of the solution for an improved solution.

In the first iteration, a solution from the zone elite is selected as an initial solution. We need to decide which lead model scheme will select the seed population from the path neighborhood of the solution. All candidates for selecting the seed population are in the path neighborhood with the exception of the first several nodes of the path. The excluded nodes were used in step 0 for search space partitioning and helped with the creation of the zone root model of the zone from which the solution comes. The excluded nodes belong to the zone-creation nodeset. Therefore, the very next node after the excluded nodes is the *first branching node* on the solution path with candidates for the seed population. This *first branching node* (1) is the first node which the zone root model encountered when it started running in step 1 and (2) is inherited by all solutions of the zone in step 1. The *first branching node* and the rest of path nodes following the *first branching node* are called the *branching nodeset* of the solution. So, all candidates for selecting the seed population are in the neighborhood of the *branching nodeset*.

In our scheme, the *branching nodeset* is divided in the middle, called *branching midpoint*, thus creating two segments. For each segment, all candidates from the segment are selected as the seed population, and the sweepset search is applied.

In more detail, the lead model carries out its simulation run according to the solution. For each node of the first segment (starting with the *first branching node*), a branching event is triggered. In the branching event, the lead model (1) selects for itself the option according to the solution, (2) spawns a new model for each option of the node neighborhood, and (3) defines the *first branching node* for each new model. The *first branching node* of a new model is defined as the first node which the new model encounters, at a simulation time t_F, after being spawned by the lead model. The information about its *first branching node* is passed from each parent model to its child model up to a solution. Therefore, each solution found by the local search inherits this information defined by its lead model.

The spawned models (the seed population of sweepset search of the first segment) run with the lead model in parallel. After spawning the entire seed population of the first segment, the lead model encounters the *branching midpoint* at simulation time t_{M1}. By now, the lead model has done all required work for the first segment and takes a short break at the midpoint (rest stop). While the lead

model is resting, the sweepset search of the first segment is busy searching sweep by sweep for elite solutions.

The local search maintains a candidate list (output of every iteration and input of the following iteration) of elite solutions found by the sweepset searches during the local search. If an elite solution did not encounter a pruning event during its sweep since it was spawned by its lead model, then this elite solution is not included in the candidate list because it was already fully searched by the sweep.

When the sweepset search of the first segment is completed, the lead model resumes running from the rest stop. For the second segment, the lead model spawns all new models and defines their *first branching nodes* in the same way as it has done for the first segment. The new models, running in parallel with the lead model, start the sweepset search of the second segment. The end of this sweepset search concludes the first iteration of the local search. Notice that the lead model runs only in the first sweep of the sweepset searches of both segments. This ends the description of the first iteration of the local search.

Descriptions of all subsequent iterations are identical and differ slightly from the description of the first iteration. In a subsequent iteration, a solution from the candidate list needs to be selected as an initial solution. To search solutions with larger subtrees first, the algorithm selects the best solution with a long *branching nodeset*. A long *branching nodeset* has time t_F of its *first branching node* equal or smaller than time of the *branching midpoint* which has been defined in the first iteration (that is, $t_F \leq t_{M1}$). A short *branching nodeset* has $t_F > t_{M1}$. The selected initial solution is "processed" in the same way as the initial solution of the first iteration: its *branching nodeset* is divided into two segments, its lead model generates seed populations for the sweepset searches of both segments, and so on.

However, when only solutions with short *branching nodesets* are in the candidate list, the best solution is selected and processed in the same way as a long *branching nodeset* solution with only a small difference at the end of both segment searches. From all solutions found by both searches, only (1) the better solutions than the initial solution of the iteration or (2) the global best solutions are inserted into the candidate list. When the candidate list is empty or there is no improvement over a certain number of iterations, the local search of the current solution selected from the zone elite is completed. The algorithm then selects the next best solution from the zone elite for the next local search. The sweep algorithm ends when all zone elite are searched for improvement by the local search or the algorithm reaches a time limit.

A tree graph on Figure 1 illustrates the local search with 3 iterations. The first horizontal line represents a path of the lead model of the first iteration. The lead model traverses the tree according to an initial solution (S1) selected from the zone elite with the objective function value of 15. The goal is to minimize the function value. The lead model starts its simulation run (point S) at time t_0 and spawns models (seed population of the first segment) from the *first branching node* (A) up to its *branching midpoint* (M1). After a rest stop at M1, it spawns models for the second segment up to the end of its simulation run (point S1). Notice that at node B, it spawned a model leading to the best solution (S2) found in the first iteration with the improved objective function value of 14. In the second iteration, its lead model follows the path of S2: starting at S, passing A and B nodes, starting spawning models from its *first branching node* (C), taking a rest at its *branching midpoint* (M2), and so on. At node D, it spawned the best solution (S3) of the iteration. In the third iteration, its lead model follows the path of S3: starting at S, passing A, B, C, and D nodes, starting spawning models from its *first branching node* (E), taking a rest at its *branching midpoint* (M3), and so on. No elite solution is found in this iteration, so the local search of S1 ends with the improved objective function value of 12.

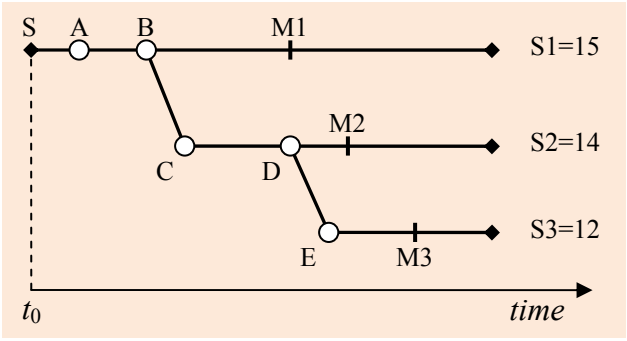

Figure 1: Tree graph of local search with 3 iterations.

Notice the purpose of the *first branching node*: the use of the *first branching nodes* ensures that with each subsequent iteration, smaller and smaller subtrees (subregions) are searched. In this way, the local search keeps zooming in on the most promising subregion. Thus, this local search is yet another example of nested partitioning used in the sweep algorithm.

2.3 Nodeset Savings and Sweep Signature

Nodeset Savings: In the spirit of the front-end savings of computational resources (Mejtsky 2007), we can find savings among candidate solutions, such as the zone elite, for the local search. The idea is: if two or more solutions have the same front part of their tree paths, then search the front part only once. This is done by repositioning the *first branching node* to the first node following the common front part for each solution that we do not want to repeat the search. In detail, the nodeset savings procedure com-

pares all pairs of the candidate solutions. For each pair, if both solutions have their *first branching nodes* inside the common front part, then redefine the *first branching node* of one of the pair to the next node following the common front part.

We reuse Figure 1 to illustrate the concept of the procedure. Suppose we have 3 solutions in the zone elite. The tree paths of the solutions (S1, S2, and S3) are shown on redefined Figure 1. Their *first branching node* is the same node A (that is, the solutions are from the same zone). When comparing S1 and S2, both have their *first branching node* (A) inside their common front part (from S to node B). Therefore, the procedure redefines the *first branching node* of S2 from node A to node C. The same is true and done when we compare S1 and S3. When comparing S2 and S3, both have their redefined *first branching node* (C) inside their common front part (from S to node D). Therefore, the procedure redefines the *first branching node* of S3 from node C to node E. Notice that the *branching nodeset* of S3 shrank from the original nodeset (A to S3) to a shorter nodeset (E to S3) resulting in nodeset savings for the local search of S3. Similarly, we have S2 nodeset savings: from (A to S2) to just (C to S2).

Sweep Signature: This concept is a result of the following concern: During a sweep, a pruning function takes care of purging weaker individuals from the sweep population. However, this purging is relative only to the population of the sweep. Imagine a sweep where the entire population is of poor quality relative to the populations of previous sweeps. In such a case, there is no need to continue with such a sweep. One solution to this problem is to compare, during a sweep, each individual to the *alpha* solution, called benchmark BEST in (Mejtsky 2007), and eliminate all weak individuals.

We implemented a slightly different version where the benchmark is the best sweep signature. In each pruning event during every sweep, the sweep collects the value of the pruning function of the sweep's best individual at that time. The sequence of the values, different for each sweep, forms a curve – sweep signature. The algorithm keeps track of the best sweep signature found thus far – *alpha* signature. In pruning events during each sweep, the function value of each individual is compared with the value of the *alpha* signature for that time. If the function value of an individual deviates from the *alpha* value more than an allowed percentage deviation (limit), then the individual is eliminated forever. The algorithm collects percentage deviations for elite solutions so that we can set the limit (algorithm's parameter) not to eliminate the elite.

3 EXPERIMENTAL EVALUATION

To test the improved sweep algorithm, we considered 58 instances from four classes of job shop scheduling (JSS) standard benchmark problems:

- Adams et al. (1988) ABZ 5 – ABZ 9;
- Applegate and Cook (1991) ORB 1 – ORB 10;
- Fischer and Thompson (1963) FT 6, FT 10, and FT 20; and
- Lawrence (1984) LA 1 – LA 40.

To obtain the best solution for each instance, we fine-tuned (optimization of optimization) the sweep algorithm by selecting from the menu of the basic sweep algorithm and from the menu of the algorithm's parameters. For a detail description of applying the basic sweep algorithm to JSS, see Mejtsky (2007).

Tables 1 and 2 present the best solution found by our algorithm for each instance. The tables list in the first two columns the instance names and sizes (the number of jobs × the number of machines). Column OPT shows the optimum or the best known solutions. The next two columns report the best solutions (Sweep) produced by the algorithm and the corresponding percentage deviations (%) relative to OPT values. The last column (Time) reports the run times in minutes for the best solutions. Time 0 means the run time was smaller than one minute.

Table 1: Results for ABZ, ORB, and FT problems.

Name	Size (JxM)	OPT	Sweep	%	Time (min.)
ABZ 5	10x10	1234	1242	0.7	5
ABZ 6	10x10	943	943	0	12
ABZ 7	20x15	656	704	7.3	14
ABZ 8	20x15	665	710	6.8	44
ABZ 9	20x15	679	724	6.6	31
ORB 1	10x10	1059	1060	0.1	0
ORB 2	10x10	888	902	1.6	0
ORB 3	10x10	1005	1032	2.7	0
ORB 4	10x10	1005	1032	2.7	1
ORB 5	10x10	887	908	2.4	5
ORB 6	10x10	1010	1013	0.3	2
ORB 7	10x10	397	405	2.0	0
ORB 8	10x10	899	921	2.5	16
ORB 9	10x10	934	948	1.5	0
ORB 10	10x10	944	967	2.4	0
FT 06	6x6	55	55	0	0
FT 10	10x10	930	941	1.2	26
FT 20	20x5	1165	1165	0	4

The algorithm was implemented in MS Visual C++, and the tests were carried out on a HP Compaq Presario PC with a 2.19 GHz AMD Athlon 64 Processor, with 448 MB of RAM, on the MS Windows XP Home Edition 2002 SP 2 operating system.

Table 2: Results for LA problems.

Name	Size (JxM)	OPT	Sweep	%	Time (min.)
LA 1	10x5	666	666	0	0
LA 2	10x5	655	655	0	0
LA 3	10x5	597	604	1.2	1
LA 4	10x5	590	590	0	0
LA 5	10x5	593	593	0	0
LA 6	15x5	926	926	0	0
LA 7	15x5	890	890	0	0
LA 8	15x5	863	863	0	0
LA 9	15x5	951	951	0	0
LA 10	15x5	958	958	0	0
LA 11	20x5	1222	1222	0	0
LA 12	20x5	1039	1039	0	0
LA 13	20x5	1150	1150	0	0
LA 14	20x5	1292	1292	0	0
LA 15	20x5	1207	1207	0	0
LA 16	10x10	945	970	2.7	16
LA 17	10x10	784	786	0.3	12
LA 18	10x10	848	859	1.3	2
LA 19	10x10	842	850	1.0	25
LA 20	10x10	902	916	1.6	8
LA 21	15x10	1046	1090	4.2	5
LA 22	15x10	927	963	3.9	1
LA 23	15x10	1032	1032	0	0
LA 24	15x10	935	960	2.7	3
LA 25	15x10	977	1008	3.2	6
LA 26	20x10	1218	1218	0	240
LA 27	20x10	1235	1283	3.9	2
LA 28	20x10	1216	1226	0.8	36
LA 29	20x10	1157	1216	5.1	120
LA 30	20x10	1355	1355	0	2
LA 31	30x10	1784	1784	0	0
LA 32	30x10	1850	1850	0	0
LA 33	30x10	1719	1719	0	0
LA 34	30x10	1721	1721	0	3
LA 35	30x10	1888	1888	0	0
LA 36	15x15	1268	1294	2.1	10
LA 37	15x15	1397	1441	3.2	23
LA 38	15x15	1196	1245	4.1	14
LA 39	15x15	1233	1271	3.1	17
LA 40	15x15	1222	1244	1.8	3

We implemented due date (DD) control as described in Mejtsky (2007). Even then due dates are not explicitly stated in the JSS problem, the DD control is used to weed out partial solutions (admissible pruning) which would have led to solutions with makespan exceeding the imposed DD limit. An initial DD, the algorithm's parameter, is used in step 1 (zone search) for each sweep. In the local search, each iteration has its DD equal to the makespan of its initial solution.

We included an option for delay schedule search. The delay option is implemented as suggested ("doing nothing" choice) in Mejtsky (2007). The delay schedule is a schedule in which a machine is kept idle when it could start processing a job from its queue. The option inclusion resulted in a significant enlargement of solution space containing many solutions with large delay times, and therefore, poor quality in terms of makespan. In order to reduce the solution space and to control the delay times, we developed the sweep signature concept. However, our results still did not significantly improve and they underperformed the results reached by not using the delay option.

4 COMPARISON WITH OTHER ALGORITHMS

We compare the improved sweep algorithm with the basic sweep algorithm and with 13 metaheuristics from a comparative analysis study (Gonçalves et al. 2005). The comparison is done by average percentage relative deviation (APRD) from the optimum. In the study, their hybrid genetic algorithm and local search was compared by APRD with 12 metaheuristics from the JSS literature; no comparison by run time was made. The metaheuristics compared on the JSS benchmarks include tabu search, genetic algorithms, hybrid of genetic algorithm and simulated annealing, and GRASP.

Table 3 compares the improved sweep algorithm with other algorithms. The list of other algorithms is given in the first column. Since other algorithms solved different subsets of the JSS instances within FT and LA classes, the second column shows the number of instances solved (NIS). The average percentage relative deviations (APRD) from the optimum (or the best known solution) for the other algorithms (OA) and for the improved sweep algorithm (Sweep) are calculated in the last two columns.

The clear winner in the comparison of these algorithms is the tabu search approach of Nowicki and Smutnicki (1996). This tabu search solved all 43 instances of the LA and FT classes of the JSS problems with an average relative deviation of only 0.1% from the optimum. The improved sweep algorithm solved the same 43 instances with an average relative deviation of 1.1%. Our improved sweep algorithm outperformed 7 of the 13 metaheuristics. Therefore, our improved sweep algorithm ranks in the middle of the relative performance comparison which we regard as a success. (Note: Currently, we are testing a preemption approach to searching delay schedule space. The preliminary results for the 43 instances have an average relative deviation of only 0.6%.)

We are pleased to report on a significant improvement of the sweep algorithm: The basic sweep algorithm solved 26 instances of the JSS problems with an average relative deviation of 2.7%; however, the improved sweep algorithm solved the same 26 instances with an average relative deviation of only 0.7%.

Table 3: Comparison of algorithms by APRD.

Algorithm	NIS	APRD	
		OA	Sweep
Problem and Heuristic Space			
Storer	11	2.4	1.9
Genetic Algorithms			
Aarts – GLS1	42	2.0	1.1
Aarts – GLS2	42	1.7	1.1
Croce	12	2.4	0.9
Dorndorf – PGA	37	4.6	1.3
Dorndorf – SBGA (40)	35	1.4	1.3
Dorndorf – SBGA (60)	20	1.9	2.2
Gonçalves (1999)	43	0.9	1.1
GRASP			
Binato	43	1.8	1.1
Aiex	43	0.4	1.1
Hybrid GA/SA			
Wang and Zheng	11	0.3	0.9
Tabu Search			
Nowicki & Smutnicki	43	0.1	1.1
Hybrid GA/LS			
Gonçalves (2005)	43	0.4	1.1
Basic Sweep Algorithm			
Mejtsky (2007)	26	2.7	0.7

5 DISCUSSION

We sketch some ideas for future research.

Sequential Sweep: In a sweep, all encountered nodes are fully expanded, and pruning events (PEs) are triggered dynamically. In each branching event, one model is spawned for each option. However, we can have a sweep where the nodes are only partially expanded and PE times are fixed. In such a sweep, the root model starts running and encountering nodes. In each branching event, only one option is selected for the model to continue. When the model reaches the first PE time, the model pauses; the sweep backtracks to a partially expanded node, picks another option, and spawns a model.

The spawned model starts running and branching like the root model until it pauses at the first PE time. The sweep keeps backtracking for new models until there are enough models assembled at the first PE time to trigger the pruning event. After sorting by a pruning function, only the elite models can continue running; the rest stays on this first island. The elite run sequentially, one by one, from the first PE time to the second PE time in the same way as they were running to the first PE time. When the sweep needs to assemble models at a PE time to trigger the pruning event, it backtracks to an island or a partially expanded node, preferably to the node left over by some elite model (a local search in the neighborhood of the elite partial solution), for another model. In this way, the models move sequentially and uninterrupted from one PE to the next PE, and the number of partially expanded nodes (mini-islands) grows. A mini-island is a parent model which has not finished spawning models.

Eventually, some models finish their simulation runs while others are still running. The sweep can use elite solutions for the local search by backtracking to their leftover partially expanded nodes and searching the neighborhoods of the nodes. Since models run sequentially and not in parallel, there is no need for a single centralized global calendar of events to synchronize their simulation times. Therefore, in this sequential sweep, each model has its own event calendar.

We need such sequential sweep, where a model runs uninterrupted for a while (from a PE to the next PE), in case, (1) we can have only one model in the CPU memory (because of its large size) and the rest of models is in a direct-access file, and (2) there is a large overhead cost for frequent swapping models between the CPU memory and the file. The sequential sweep reduces the swapping cost.

Pruning Function: In JSS, if the objective is minimizing total (weighted) tardiness, then use the pruning function: maximizing total (weighted) slack time (including lateness as negative slack). Notice the similarity with a (weighted) slack-based criticality measure of shifting bottleneck heuristic.

Marriage with Dispatching Rules: In production scheduling/dispatching, schedules need to be generated within a time limit. During a sweep, the user (or the sweep algorithm automatically) can control the execution speed of the sweep: To every decision point in a simulation model, a dispatching rule or the branching event can be applied. If only dispatching rules are applied and no branching is used, then the schedule is generated fastest with only one simulation run. However, the more the branching is applied, at the expense of fewer dispatching rules, the slower the schedule is generated but with more search for a better schedule. Notice that a variable CEILING and FLOOR can be used for the sweep's speed control as well.

Thanks to the sweep algorithm, new (generalized) dispatching rules can be devised, such as (1) a rule with more than one selected options, (2) more than one rule applying to a decision point, (3) a hybrid of rules and branching, (4) if the algorithm's speed is too fast/slow, use a rule/branching; or (5) if a machine is a bottleneck, then use branching, else use a rule.

The sweep algorithm with speed control can be utilized for real-time manufacturing operation scheduling. Imagine a production where the algorithm generates a two-hour schedule every one hour (or anytime). In this cruising phase, the algorithm is running only one sweep, and its simulation time maintains a constant two-hour lead ahead of real time. With this slow advancing of simulation time, the algorithm can afford to apply more branching and fewer

dispatching rules. When an unexpected event hits, such as a machine breakdown, a recovery phase starts. In this phase, the algorithm must quickly generate a new one-hour schedule for the affected area, so it applies only dispatching rules. After rescheduling the area, the algorithm returns to the cruising phase in its never-ending run.

Backward Simulation: In backward simulation, see (Mejtsky 2007), if branching is used instead of dispatching rules, then problems with reversing the rules are eliminated.

Evaluating Dispatching Rules: When the rules are options in branching events, the sweep algorithm can be used for evaluating dispatching rules.

6 CONCLUSION

We discussed new additions, such as backtracking and local search, to the basic sweep algorithm. The additions, along with the new search framework, increased diversification and intensification of our hierarchical search process. The improved search process led to improved performance as documented in the comparison between the basic and improved sweep algorithm. Also, we compared the improved sweep algorithm with 13 metaheuristics on the JSS standard benchmark problems. Our algorithm ranks in the middle of the comparison. Because of the research contribution of this paper, simulation now has its own optimization tool which is quite competitive with metaheuristics using disjunctive graph.

The high modeling capability of simulation and the general nature of tree search offer a large array of applications for the sweep algorithm. For example, this algorithm – simulation-based metaheuristic – can be applied to solving sequential decision problems, such as resource-constrained project scheduling, traveling salesman, packing, or (real-time) production scheduling.

REFERENCES

Adams, J., E. Balas, and D. Zawack. 1988. The shifting bottleneck procedure for job shop scheduling. *Management Science* 34(3):391-401.

Applegate, D., and W. Cook. 1991. A computational study of the job shop scheduling problem. *ORSA Journal on Computing* 3(2):149-156.

Fisher, H., and G. L. Thompson. 1963. Probabilistic learning combination of local job shop scheduling rules. In *Industrial Scheduling*, ed. J. F. Muth and G. L. Thompson, 225-251. Englewood Clifs, New Jersey: Prentice Hall.

Fu, M. C., F. Glover, and J. April. 2005. Simulation optimization: A review, new developments and applications. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 83–95. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. .

Gonçalves, J. F., J. J. M. Mendes, and M. G. C. Resende. 2005. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research* 167(1):77–95.

Henderson, S. G., and B. L. Nelson. (Eds.) 2006. Handbook of simulation. Elsevier. Forthcoming.

Lawrence, S. 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Mejtsky, G. J. 1986a. Toward expert simulation systems in job shop scheduling. In *Proceedings of the 1986 National Computer Conference*, 143-147.

Mejtsky, G. J. 1986b. A new combinatorial method: parallel modeling. *International Congress of Mathematicians*, University of California, Berkeley, CA.

Mejtsky, G. J. 1986c. Application of expert simulation system to job shop scheduling. In *Proceedings of the 1986 APICS Spring Seminar*, American Production and Inventory Control Society, 248-257.

Mejtsky, G. J. 2007. A metaheuristic algorithm for simultaneous simulation optimization and applications to traveling salesman and job shop scheduling with due dates. In *Proceedings of the 2007 Winter Simulation Conference*, ed. S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 1835-1843. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc. .

Nowicki, E., and C. Smutnicki. 1996. A fast taboo search algorithm for the job-shop problem. *Management Science* 42(6):797-813.

Shi, L., and S. Ólafsson. 2000. Nested Partitions Method for Global Optimization. *Operations Research* 48(3): 390-407.

Weinberger, J. 1982. An optimization strategy based on quasiparallel handling. *Simula Newsletter* 10(2):8-9.

Zhou, R., and E. Hansen. 2005. Beam-stack search: integrating backtracking with beam search. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, 90–98.

AUTHOR BIOGRAPHY

GEORGE JIRI MEJTSKY has over twenty years of experience in simulation. He holds an M.S. Degree in Operations Research from University of Economics, Prague, Czech Republic. His research interests include discrete-event simulation optimization, production scheduling, and stock market optimization. He is a member of the INFORMS Simulation Society. His email address is <george.mejtsky@yahoo.com>