Proceedings of the 2007 Winter Simulation Conference
S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, eds.

A METAHEURISTIC ALGORITHM FOR SIMULTANEOUS SIMULATION OPTIMIZATION AND APPLICATIONS TO TRAVELING SALESMAN AND JOB SHOP SCHEDULING WITH DUE DATES

George Jiri Mejtsky

Simulation Research
314 Steeplechase Drive
Exton, PA 19341, U.S.A.

ABSTRACT

We describe a metaheuristic algorithm for simulation optimization. Traditionally, discrete event simulation optimization is carried out by multiple simulation runs executed *sequentially*. At the end of each simulation run, the run is evaluated (using model output – black box approach) by an objective function. If we carry out simulation runs *simultaneously*, then we can evaluate (using model internal data – white box approach) different simulation runs during their execution before the end is reached. Thus, we can eliminate the inferior runs early and allow only the most promising runs to continue to the end. We explore this parallel competition of simulation models on a single processor computer. Applications of the algorithm to traveling salesman and job shop scheduling problems are presented. In conclusion, our results suggest that the algorithm is a suitable approach for solving some combinatorial problems, and it represents a promising "nonsequential" avenue for simulation optimization.

1 INTRODUCTION

Traditionally, simulation optimization is carried out by multiple simulation runs executed *sequentially*. We introduce a different paradigm for simulation optimization – multiple simulation runs executed *simultaneously*. We focus on deterministic discrete event simulation running on a single processor computer.

Currently, the most attractive simulation optimization methods are metaheuristics, such as Tabu Search or evolutionary algorithms. For recent review of literature on simulation optimization, see Fu, Glover, and April (2005).

"The metaheuristic approach to simulation optimization is based on viewing the simulation model as a black box function evaluator," writes April et al. (2003). In this approach, (1) a set of values of input parameters is chosen; (2) the model is run from the simulation time t_0 to t_{end}; (3) response (that is, performance or results) of the run is evaluated by an objective function; (4) based on responses from the runs, a new set of values of input parameters is chosen for the next run; return to (2). This looping goes on until some stopping criterion is met. Hence, during such a simulation optimization, a sequence of simulation runs is carried out, and the simulation model is treated as a *black box* -- no knowledge of the workings of the model is required.

We explore a different approach to simulation optimization. In this approach, the model is treated as a *white box* -- the knowledge of the internals of the model is paramount -- in contrast to the black box approach. Ideas leading to our approach are discussed in Section 2. In the approach, models run concurrently while traversing a decision tree (described in Section 3). Their "local" simulation time is synchronized by a single "global" calendar of events (Section 3.2). When a model encounters a tree node with k branches, the model spawns k new models (detailed in Section 3.2). Each child model traverses a different branch, and the parent model ends running.

As simulation time advances, more and more models are generated, and as a result, the model population expands. To limit the expansion, the mediocre models are weeded out from the population (in pruning events), and only the best performing models are allowed to continue traversing the tree and breeding offsprings (see Section 3.3). After the population size drops (caused by a pruning event), the population grows again until the next pruning event or the end of their simulation runs. The evolution of the population, the way models pass through the tree, we call a sweep (Section 3.4). The sweep algorithm as an evolutionary algorithm, and front-end and back-end savings of computational resources are discussed in Section 3.4.

The application of the sweep algorithm to (1) the traveling salesman problem is given in Section 3, (2) the job shop scheduling problem with the minimum makespan objective function is described in Section 4.1, and finally, (3) the job shop scheduling problem with due dates is presented in Section 4.2. To deal with due dates, backward simulation (Section 4.2.1) and a pruning rule (Section 4.2.2) are applied.

New (unpublished) development of the sweep algorithm is reported in Section 5. New pruning rules and a pruning function for solving job shop scheduling with due dates are introduced in Section 5.1 followed by results of solving standard benchmark examples for job shop scheduling (Section 5.2). Finally, we discuss future improvement in the sweep algorithm (Section 5.3). Multiple and pyramid-building sweeps are proposed in Section 5.3.1 and Section 5.3.2, respectively. In Section 5.3.3, parallelization of the sweep algorithm is sketched for use in parallel or distributed computing.

In conclusion, the results from the experimental evaluation are positive and suggest that the sweep (branch-and-prune) algorithm (1) can be used for solving some combinatorial problems, (2) still has plenty of room for improvement, and (3) appears to be a promising alternative to the traditional simulation optimization.

2 DEVELOPMENT HISTORY

In the 1980s, we used Simula for simulation modeling (Mejtsky 1983) in our research in Prague, Czech Republic. Simula is a simulation language and has been highly influential in modern programming methodology. We were intrigued by language concept: nesting quasi-parallel systems and passing control among them. Our research led to the development of graphical representation of the concept (later called Mejtsky's diagrams) as described in (Mejtsky and Kindler 1980, Mejtsky and Kindler 1981). The graphical representation contributes to better understanding and faster utilizing the concepts, models, and theories. For example, see Kindler (1995). We realized that we have a conceptually powerful tool, Simula, which enables simultaneous simulation. However, how should we utilize it for optimization?

In 1984, we started experimenting with running simulation models simultaneously. This research was performed at the Imperial College of Science and Technology in London, England. For testing, we selected the traveling salesman problem (TSP) as the simplest problem to model.

It can be formulated in terms of graph theory as: given a complete weighted graph (where the vertices would represent the cities, the edges would represent the roads and the weights would be the cost or distance of that road), find a Hamiltonian cycle having the least weight.

Imagine a traveling salesman who has to visit each of a given set of cities by car. What is the shortest route that will enable him to do so and return home, thus minimizing his total driving?

For launching models with a different set of input parameters, we picked branching approach because it had been useful to us in solving another optimization problem: chess problem (Mejtsky 1982).

The basic idea of branching 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 for each possible decision choice. 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 node is a parent of another node, child, when it is one step higher in the hierarchy and closer to the root node. Each decision choice point is represented in a simulation model by a decision choice event (DCE).

3 DESCRIPTION OF ALGORITHM FOR SOLVING TSP

Now, we have almost all parts to assemble the algorithm for solving TSP. The algorithm can be described by initial, branching, and pruning phases.

3.1 Initial Phase

The simulation clock is initialized to zero, t_0, and a single model, the root model, starts running. Events of the root model are executed in order controlled by the event calendar. The simulation clock advances from one event to the next event, if any, until the first DCE is reached. This first DCE encounter ends the initial phase.

3.2 Branching Phase

All decision choice events, DCEs, are processed in this phase. Processing DCE means that a model with k decision choices will branch and create k new models, children. Each child will be a copy of the parent model. This means that each child will inherit from the parent its current state, its history (that is, the path through the decision tree, traversing from the root node to the current node), and its future (that is, planned events in the event calendar). Additionally, each offspring will select a different option from the k decision options. After the parent model has finished its last task -- giving birth to its k offsprings -- and is no longer needed for the optimization, the parent model dies (like all pacific salmon die after spawning). This concludes processing DCE.

As an example, in TSP with n cities, the root model starts running at time t0. In the root model, the salesman is in his hometown and has to choose from a list of $(n - 1)$ unvisited cities which one he should visit first. On the decision tree, this first decision choice point corresponds to the root node with $(n - 1)$ branches. The first decision choice point is represented in the root model by the first decision choice event, DCE. Therefore, during the processing of this first DCE, $(n - 1)$ new models originate from the root model. Each new model contains one salesman

traveling from his hometown to a different destination city. After giving birth to its (n – 1) offsprings, the root model dies. Note that, before the first DCE, there was only one model running: the root model. However, after the first DCE, there are (n – 1) models running concurrently. Consequently, a life of the root model, as measured by the simulation time, was rather short, was it not?

When a new model is born, it starts running independently and in parallel with all other models. The model executes its events, if any, until DCE is reached. During execution of the decision choice event, a set of new models is born and starts running; however, the parent model stops running. As a result, a model runs only from the time it is created in a decision choice event until it executes the next decision choice event.

In TSP, when a new model with one salesman is created, the salesman travels from the current city to a destination city, selected from the model's current list of unvisited cities. The list is inherited from its parent. Upon arrival at the destination city, the salesman has to decide which city from the updated list of remaining k unvisited cities he should visit next. This decision choice point with k options leads to executing DCE.

Only one "global" event calendar is shared by all models which ensures synchronization of the simulation time in each model.

Branching models mirrors the decision tree of an optimization problem. Decision choice points, nodes in the decision tree, correspond to DCEs, nodes in the DCE tree. A model runs only for a short period of simulation time just to traverse an edge between two adjacent DCE nodes. As simulation time advances, more and more models run in parallel; hence, a need for pruning models arises. However, which model should we exterminate?

3.3 Pruning Phase

If an optimization problem has a small solution space, then we can use a complete enumeration to find the optimal solution. This would allow all models to run and branch with no pruning until the models end their runs. If it is not practical to use the complete enumeration (for example, in TSP with n cities, we have $n!$ solutions), then we resort to searching only a subset of the solution space and discarding the rest of solutions. During a simulation run, when the simulation time $t < t_{end}$, a model represents only a partial solution (not all values of the input parameters are known). The solution is obtained only at the end of the run, t_{end}.

In our algorithm, we terminate models during their runs when the models represent only partial solutions. In fact, we are confronted by another optimization problem: a "partial solution" optimization problem. Furthermore, for terminating models, we cannot use the objective function of the problem we are solving. In TSP, when the simulation time $t < t_{end}$, we cannot use the shortest route objective function for pruning because all salesmen, so far, have traveled the same distance. So, how do we find which model to terminate?

That is why we need to look inside the models for help. (Where else can we look?) Therefore, the knowledge of the internals of the model is paramount – the white box approach. We need to construct a function, based on model data, which will wipe out "bad" models, and in doing so, it should remain "friendly" to the objective function. Such a "friendly" function should help the objective function to find the optimal solution and not to hinder it. In weeding out subpar models, a success of such a function (and so a success of the whole algorithm) is based on the assumption that *the optimal solution also has the optimal or near-optimal partial solutions*. If we can compose such a function, then we will have found the key part of our algorithm.

We call the friendly function -- the pruning function (PF) -- and use PF only in a pruning event when the number of models reaches the maximum allowed level of models (CEILING) at simulation time t_P. Each pruning phase is represented by one pruning event. The pruning function reduces the number of models from CEILING to a pruning level (FLOOR), where FLOOR < CEILING; therefore, the pruning function represents a "global" pruning over the whole model population in contrast to a "local" pruning (pruning rule) on an individual model level as discussed in Section 4.2.2.

Setting levels of the two algorithm's parameters, CEILING and FLOOR, has a direct impact on the computational speed and the quality of results. The lower the CEILING is set, the more this algorithm behaves like the greedy algorithm, thus becoming stuck in a local extreme. The higher the CEILING is set, the more the local extreme neighborhoods can be searched, but the sweep runs longer. The same conclusion applies to FLOOR. During a sweep, the levels CEILING and FLOOR can be constant (the simplest approach we used often) or can vary (as we occasionally used it in solving benchmark examples in Section 5.2).

3.4 Discussion of Algorithm

From the simulation time t_0, when the root model starts running, the number of models running concurrently grows (branching phase) until the first pruning event is triggered at the CEILING level. During execution of a pruning event (pruning phase), the population size drops to FLOOR. From this level on, the number of models grows again (branching phase) until the next pruning event is triggered, and so on. Thus, the number of models oscillates as models traverse the DCE tree (sweep).

Other approaches to simultaneous simulation optimization use time dilation (Schruben 1997) or recursive simulation (Gilmer and Sullivan 2000).

A graphical example of a sweep with 3 pruning events at times t_{P1}, t_{P2}, and t_{P3} is depicted in Figure 1 where the

population size (bold line) oscillates between the CEILING (sweep width) and FLOOR limits.

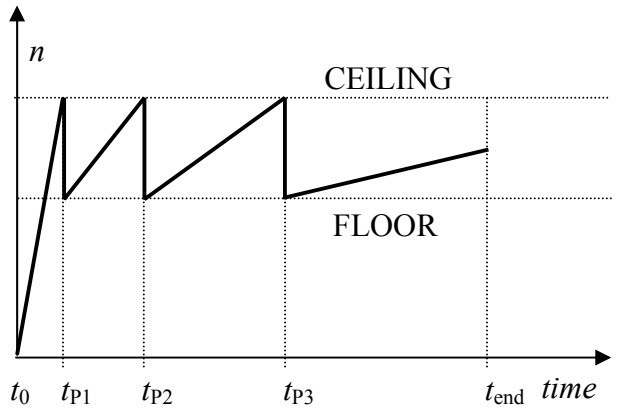

Figure 1: Oscillating population size (the number of models, n) during a sweep (from the simulation time t_0 to t_{end}).

There are similarities with a classical tree search algorithm -- Beam search. Both tree searches use a pruning function to limit tree search to a search width containing the most promising nodes.

At the beginning of a sweep, diversification is strongest and intensification is zero. However, as the sweep progresses, intensification gradually gains intensity at the expense of diversification. Near the end of the sweep, intensification is strongest and diversification is zero.

One could describe the sweep algorithm (tree search) in terms of biological evolution (Darwin's Tree of Life) as an evolutionary algorithm. All models running in parallel would play the role of individuals in a population, and the fitness function (our pruning function) would determine the environment within which the population live. The pruning function acts like a global war for a limited resource – space. As the population grows, a point (CEILING) is reached where no more space is available for all and a violent competition event (pruning event) is triggered. Evolution of the population (sweep) takes place by a repeated application of reproduction, natural selection, and survival of the fittest operators. Reproduction (spawning models) introduces new variation into the population. Natural selection preferentially selects the fittest individuals by applying the fitness function. The fittest individuals evolved with more adaptable traits; consequently, they survive and reproduce. Some traits are inherited (in TSP: a sequence of visited cities), and some traits are new (in TSP: the next visited city).

Traits are known as genes (input parameters of a problem). A solution to a problem is represented as a set of values of input parameters. These parameters are joined together to form a string of values (chromosome). Each individual is represented by a particular chromosome. In a partial solution, the individual is represented by a partial chromosome (in TSP: a sequence of visited cities).

Notice how the genetic algorithm artificially modifies the genetic material, DNA, in individuals to breed a new population, a more desirable breed. In contrast to the *controlled* breeding (genetic engineering) of the genetic algorithm, the sweep algorithm allows every individual natural, *unrestricted* breeding.

The sweep algorithm exhibits front-end and back-end savings of computational resources in contrast to simulation models running from time t_0 to t_{end}. In a decision choice event, DCE, when k models are born, all k siblings have the same front-end part of their simulation runs, from t_0 to the DCE time. This common front-end part is executed only once and not k times when each sibling runs from t0 to tend. The front-end savings are a direct result of using the decision tree approach in the sweep algorithm. Back-end savings are results of eradicating poor quality models in pruning events; therefore, the inferior models are stopped as early as possible before reaching their t_{end}.

In TSP, how do models differ? At any time during a sweep, the models differ by the number of visited cities. So we apply *maximizing the number of visited cities* as the pruning function. *Minimizing the number of unvisited cities* would do the same.

In 1984, we tested the algorithm on small size TSP examples from literature. The results were optimal, and for larger size examples, the results were optimal or near-optimal. We applied the algorithm to the project management problem and the resource allocation problem with similar results. Then we proceeded to the job shop scheduling problem (JSS).

4 APPLICATION OF ALGORITHM TO JSS

We describe JSS 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 (that is, minimizing makespan).

4.1 JSS with Minimum Makespan

We need to construct a decision choice event, DCE, and a pruning function, PF. Metaheuristics have been used to solve JSS. In practice, however, simulation with dispatching rules (scheduling rules or sequencing rules) have been more frequently applied due to their ease of implementation and their low time costs even though dispatching rules are unable to fare better than the local search methods.

Whenever a machine becomes available, a dispatching rule inspects the waiting jobs and selects the next job for processing. The selected job starts processing on the machine without any delay. We substitute the dispatching

rules with DCEs. In DCE with *k* waiting jobs for processing on a machine, k new models (children) are born, and the parent model dies. Each child is a copy of the parent: a complete job shop with all machines and jobs. Additionally, each child selects a different job for processing on the machine.

Notice that the shorter the makespan, the higher the cumulative utilization of the shop (all machines); therefore, we elect the pruning function: *maximizing cumulative utilization of the shop* (PF-U).

The algorithm was tested on JSS examples from literature with similar results as testing on TSP examples. The sweep algorithm and its application to JSS was presented in (Mejtsky 1986a, Mejtsky 1986b).

4.2 JSS with Due Dates

4.2.1 Backward Simulation

In the job shop scheduling with due dates (JSSD), the objective is to find a schedule meeting due dates. By using standard "forward" simulation, it is difficult to find such a schedule. We noticed that if we could somehow run simulation "backward", starting from the due dates, then every schedule would meet the due dates. Backward simulation would have a clear advantage over forward simulation in solving such problems. Our research in using backward simulation is summarized in Mejtsky (1985), which is the first -- to the best of our knowledge -- documented application of backward simulation to JSSD. One important finding is zero-delay dispatching (causing different lengths of the "forward" and the "backward" minimum makespans).

Today, backward simulation, sometimes called reverse-time simulation, is quite a common tool in job shop scheduling. Backward simulation is essentially the reverse of forward simulation. Beginning with the due date of each job, simulation works backward to determine start dates (order release times).

4.2.2 Pruning Rule

The pruning function PF-U is not as effective in finding schedules meeting due dates as it is in finding minimum makespan schedules because PF-U does not deal with the due dates. To increase performance of our JSS algorithm, we need to find a global pruning function or a local pruning rule dealing with due dates directly.

Let us define the following pruning rule PR-JDD1: *In a model, if a job cannot meet its due date (that is, the sum of its remaining processing times is greater than the remaining time to its due date), then eradicate the model from the population.*

We included PR-JDD1 with PF-U in the JSS algorithm (JSSD algorithm). Our testing confirmed an increased performance of the JSSD algorithm, as discussed in (Mejtsky 1986c, Mejtsky 1986d).

This concludes the previously reported research in the algorithm, and the new development in research follows.

5 NEW DEVELOPMENT

In this section, we present new pruning rules and a pruning function for solving JSSD followed by results of JSS standard benchmarks testing. Finally, we discuss possible improvement in the algorithm.

5.1 New Pruning Rules and Pruning Function for Solving JSSD

We added the following 3 new pruning rules to the JSSD algorithm and replaced the pruning function PF-U with the new pruning function PF-LS.

Pruning rule PR-JDD2: *In a model, suppose that a job arrives at a machine for processing; however, the machine is busy processing another job. If the machine remaining processing time is greater than the slack time of the arriving job, then eliminate the model because the arriving job cannot meet its due date (that is, the sum of its remaining processing times plus waiting time for processing on the machine is greater than the remaining time to its due date).*

Pruning rule PR-JDD3: *In a model, suppose a job **a** starts processing on a machine and there is a job **b** in the machine waiting queue. If the processing time of the job **a** on the machine is greater than the slack time of the job **b**, then discard the model because the waiting job **b** cannot meet its due date (that is, the sum of its remaining processing times plus waiting time for processing on the machine is greater than the remaining time to its due date).*

Pruning rule PR-MDD1: *In a model, if a machine cannot meet its due date (that is, the sum of its remaining processing times is greater than the remaining time to the machine due date), then eliminate the model.*

Pruning function PF-LS: *Maximizing least slack time.*

The slack time of a job is determined to be the remaining time until its due date minus the sum of its remaining processing times. The machine due date can be derived from job due dates and operation due dates of the machine.

In each model, find the job with *the least slack time.* This *least slack time* is the performance measure of the model, and the PF-LS function prefers the models with the largest values of the performance measure.

Results from testing the JSSD algorithm point to increased performance in finding schedules meeting due dates.

During a sweep, the pruning rules PR-JDD1 and PR-MDD1 are triggered after jobs consume their slack times which is towards the end of the sweep. The rules PR-JDD2 and PR-JDD3 are activated earlier in the sweep before job slack times are completely spent. In a sweep, the earlier the

poor quality models are discovered and wiped out from the population, the better for the quality of the population and therefore the better for the solution. The pruning function PF-LS outperforms PF-U in examples where due dates are defined tight; therefore, not too much slack time is left for jobs. On the other hand, the function PF-U performs better in cases when there is plenty of slack time for jobs. This leads to a future possible improvement of the JSSD algorithm: using PF-U early in a sweep and then switching to PF-LS.

5.2 Experimental Evaluation

To test the sweep algorithm, we considered 38 instances from four classes of JSS standard benchmark problems:

- Adams et al. (1988) ABZ 5 and ABZ 6;
- 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 21, LA 26, and LA 27.

The goal in solving these JSS problems is minimizing makespan. To obtain the best solution for each instance, we fine-tuned the sweep algorithm by selecting from the following menu:

- the JSS algorithm with the pruning function PF-U,
- the JSSD algorithm with PF-U and the 4 pruning rules, or
- the JSSD algorithm with PF-LS and the 4 pruning rules;
- forward or backward simulation approach; and
- a level of CEILING (FLOOR is CEILING/2).

Table 1 presents the best solution found by our algorithm for each instance. It lists in the first two columns the instance names and sizes (the number of jobs × the number of machines). Column OPT shows the optimum solutions. The next two columns report the best solutions (Best) produced by the algorithm and the corresponding percentage deviations (%) from OPT values. The last column (Time) reports the run times in minutes for the best solution. In Table 1, we did not include results when run time exceeded 25 minutes (mostly when CEILING > 250).

The algorithm was implemented in OpenOffice.org Basic, 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.

Testing stressed the importance of weeding out underperforming models as early as possible either by global pruning (the pruning function in pruning events) or by local pruning (pruning rules on an individual model level).

Table 1: Results of solving JSS benchmark problems.

Name	Size (JxM)	OPT	Best	%	Time (min.)
ABZ 5	10x10	1234	1263	2	5
ABZ 6	10x10	943	980	4	1
ORB 1	10x10	1059	1119	6	8
ORB 2	10x10	888	907	2	17
ORB 3	10x10	1005	1076	7	8
ORB 4	10x10	1005	1060	5	23
ORB 5	10x10	887	911	3	12
ORB 6	10x10	1010	1072	6	1
ORB 7	10x10	397	428	8	4
ORB 8	10x10	899	940	5	3
ORB 9	10x10	934	963	3	17
ORB 10	10x10	944	994	5	5
FT 06	6x6	55	55	0	1
FT 10	10x10	930	981	5	13
FT 20	20x5	1165	1202	3	7
LA 1	10x5	666	666	0	5
LA 2	10x5	655	655	0	4
LA 3	10x5	597	622	4	2
LA 4	10x5	590	611	4	2
LA 5	10x5	593	593	0	2
LA 6	15x5	926	926	0	6
LA 7	15x5	890	899	1	2
LA 8	15x5	863	863	0	3
LA 9	15x5	951	951	0	1
LA 10	15x5	958	958	0	1
LA 11	20x5	1222	1222	0	9
LA 12	20x5	1039	1039	0	1
LA 13	20x5	1150	1150	0	2
LA 14	20x5	1292	1292	0	1
LA 15	20x5	1207	1207	0	18
LA 16	10x10	945	988	5	4
LA 17	10x10	784	794	1	14
LA 18	10x10	848	910	7	12
LA 19	10x10	842	877	4	3
LA 20	10x10	902	949	5	2
LA 21	15x10	1046	1168	12	14
LA 26	20x10	1218	1308	7	11
LA 27	20x10	1235	1395	13	13

Testing confirmed equal importance of backward and forward simulation in solving JSS when we use zero-delay dispatching. The zero-delay dispatching is common in dispatching rule scheduling. As said in Section 4.1, whenever a machine becomes available, a dispatching rule inspects the waiting jobs and selects the next job for processing. The selected job starts processing on the machine without delay (zero-delay). However, sometimes we can get a better schedule by not selecting any waiting job, doing nothing, letting the machine be idle and processing the next arriving job before processing the already waiting jobs.

Therefore, we need to be careful in designing a decision choice event, DCE, when solving an optimization problem. Do we really have only *k* decision choices? Do we need to include the "doing nothing" choice? So far we have not included the "doing nothing" choice in our JSS algorithm since backward simulation (or backward simulation optimization) will partially take care of the problem, as concluded in Mejtsky (1985).

In conclusion, the results suggest that the algorithm still has plenty of room for improvement. There are more effective search algorithms with better results in solving the JSS standard benchmark problems. For comparison see results of the filter-and-fan algorithm in (Rego and Duarte 2006) or results of 13 methods tested in a study by Gonçalves et al. (2005). Our algorithm clearly needs a local search procedure which is currently being implemented and tested. Preliminary results point to improvement in performance.

5.3 Future Improvement

There is still much to learn, discover, and improve on the sweep algorithm. The current version of the single-sweep algorithm (with only one single sweep, only one single wavefront propagating through a DCE tree) represents a simple way to implement simultaneous simulation optimization and the simplest way to search a DCE tree for solutions. We expect that the following enhancements will improve its search performance:

- multiple sweeps with backtracking (depth-first search),
- pyramid-building sweeps (breadth-first search), and
- using parallel or distributed computing with cross-pollination.

5.3.1 Multiple Sweeps with Backtracking (Depth-First Search)

The multiple-sweep algorithm starts running the first sweep from time t_0. After completion of the first sweep at time t_{end}, a backtracking step follows. In the backtracking step, the algorithm rewinds the simulation time to some earlier time t_S, where $t_S < t_{end}$. Then it starts running the next sweep (next wave) from t_S, and so on. For that reason, the algorithm cannot terminate all subpar models found in pruning events. Some inferior models can only be suspended, deactivated, and the rest of the inferior models can be eradicated forever from the search. In a pruning event (at the time t_P), we call the set of deactivated models: an island. Hence, during a sweep, each pruning event drops an island of deactivated models (island population) so the path of the sweep is littered with islands.

At the beginning of a sweep (except the first one), the sweep picks up its initial (seed) population from the population of the nearest island. During the sweep, it encounters islands dropped there by previous sweeps. In each encounter with an island, the sweep needs to decide whether to pick up some models from the island and include them in the sweep population or not.

At the end of each sweep, its best solution is produced. The algorithm should maintain the best solution (BEST) found so far by the sweeps. That is why, during a sweep, the sweep could compare the performance of its population with a benchmark: the BEST performance (as measured by the pruning function during the BEST run). If even the best performers of the sweep population cannot keep up with the BEST performance, then the sweep (1) could be aborted before time t_{end} is reached, and after a backtracking step, the next sweep would be launched; or (2) could unload its population and pick up a better performing population from an island. In the same way, in any island population, the performance of each individual (as measured by the pruning function at the pruning time t_P) can be compared with the benchmark, and for example, the whole island population can be wiped out if underperforming the benchmark.

One could see similarities between the purpose of multiple sweeps with islands and the purpose of time dilation (Schruben 1997). Other similarities one could find between the sweep (branch-and-prune) algorithm and the filter-and-fan algorithm (Glover 1998) are: (1) the purposes of the filter approach and the fan approach of the filter-and-fan algorithm resemble the purposes of the pruning phase and the branching phase, respectively, of the sweep algorithm, (2) the functions of η_1 (filter candidate list) and η_2 parameters of the filter-and-fan algorithm resemble the functions of the CEILING (like $\eta_1.\eta_2$) and the FLOOR (like η_1) parameters of the sweep algorithm.

Notice an analogy with the simulated annealing algorithm where a global parameter *T (*called the *temperature*) has the same function as CEILING and FLOOR – to allow the algorithm to escape a local extreme. Interestingly, when *T* reaches the lowest level ($T = 0$), the simulated annealing algorithm is reduced to the greedy algorithm. Similarly, when FLOOR is set to its lowest level (FLOOR = 1), the sweep algorithm is also degraded to the greedy algorithm.

As well, notice that the lower the CEILING and FLOOR are set, the more the multiple-sweep algorithm becomes the traditional sequential simulation optimization approach.

As possible area for improvement, we should look at variable levels of CEILING (sweep width) and FLOOR during a sweep. Finally, do we need the synchronization of the simulation time in each model by the single "global" event calendar? No. However, pruning events would require modification.

5.3.2 Pyramid-Building Sweeps (Breadth-First Search)

The algorithm with pyramid-building sweeps starts running the first short sweep from time t_0 to only t_{L1}, where $t_0 < t_{L1} < t_{end}$. This first short sweep is the first "stone block" at the first pyramid step level L1. After backtracking to the first (the earliest) island (or another earlier island), the algorithm launches the second short sweep to time t_{L1} , the second stone block at level L1, with the seed population from the island. After building the first pyramid level L1 with several stone blocks, the algorithm builds the second level L2.

Several best performing models from each short sweep in level L1 form the L2 seed population, "material", for building level L2 stone blocks. Each short sweep at level L2 selects its seed population from the L2 seed population and runs from time t_{L1} to t_{L2}, $t_{L1} < t_{L2} < t_{end}$. Several pyramid levels could be build with the smaller number of stone blocks at each higher level (similar to sport rounds: quarterfinal, semifinal, and final). Only the last stone block at the top of the pyramid would be allowed to run to t_{end} and present its best solution – the solution found by this pyramid-like breadth-first search of a DCE tree.

There are many ways to assemble an algorithm by combining multiple sweeps and pyramid-building sweeps. What about embedding multiple sweeps in pyramid-building sweeps or vice versa? It is better to leave nesting sweeps for parallel computers due to their superior computing power.

5.3.3 Using Parallel or Distributed Computing with Cross-Pollination

In using parallel or distributed computing, we can envision our algorithm (with a single sweep, multiple sweeps, pyramid-building sweeps, or any combination of them) running on each processor and searching a different subtree of a DCE tree. After initial distribution of work (a seed population) from a "root" processor, each processor runs asynchronously (without "global" simulation time synchronization) with others since each processor has its own "local" simulation calendar of events. Occasionally, each processor sends the best individuals from its population and its BEST benchmark to other processors. This cross-pollination should increase the quality of overall population and therefore, increase the quality of results. We believe that such large-grain open and scalable parallelism with minimum communication and no synchronization can significantly increase performance of the algorithm.

6 CONCLUSION

We discussed a different *simultaneous* paradigm for simulation optimization than the traditional *sequential* approach. The sweep algorithm (1) utilizes model internal data (white box access) and competition among models running in parallel, and (2) exhibits front-end and back-end savings of computational resources in contrast to simulation models running from time t_0 to t_{end}.

The high modeling capability of simulation and the general nature of decision choices (options) in decision choice events offer a large array of applications for the sweep algorithm. For example, in job shop scheduling, the algorithm can be applied to evaluate different dispatching rules (as, for example, recursive simulation was used in Chong et al. (2005)), where the rules would represent the decision choices.

The results from the experimental evaluation are positive and suggest that the sweep algorithm (1) can be used for solving some combinatorial problems, (2) still has plenty of room for improvement, and (3) appears to be a promising alternative to the traditional simulation optimization.

REFERENCES

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

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

April, J., J. Kelly, F. Glover, and M. Laguna. 2003. Practical introduction to simulation optimization. In *Proceedings of the 2003 Winter Simulation Conference,* ed. S. Chick, T. Sanchez, D. Ferrin, and D. Morrice, 71-78.

Chong, C. S., M. Y. H. Low, A. I. Sivakumar, and K. L. Gay. 2005. Using simulation based approach to improve on the mean cycle time performance of dispatching rules. In *Proceedings of the 2005 Winter Simulation Conference*, ed. M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 2194–2202.

Fisher, H., and G. L. Thompson. 1963. Probabilistic learning combination of local job shop scheduling rules. *Industrial Scheduling*, 225-251.

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.

Gilmer, J. B., and F. J. Sullivan. 2000. Recursive simulation to aid models of decisionmaking. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang and P. A. Fishwick, 958-963. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers.

Glover, F. 1998. A template for scatter search and path relinking. *Artificial Evolution, Lecture Notes in Com-*

puter Science, 1363. J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (eds.), Springer Verlag, 3-51.

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

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

Kindler, E. 1995. Tutorial on Mejtsky's diagrams. In *Proceedings of the 21st Conference of ASU*, Association of SIMULA Users, Stockholm, Sweden.

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, J., and E. Kindler. 1980. Diagrams for quasi-parallel sequencing - part 1. *SIMULA Newsletter* 8:13-14.

Mejtsky, J., and E. Kindler. 1981. Diagrams for quasi-parallel sequencing - part 2. *SIMULA Newsletter* 9:17-18.

Mejtsky, J. 1982 Sachy v SIMULE [in English: Chess in SIMULA]. Lecture, SIMULA User's Group, Czech Cybernetic Society of Czech Academy of Sciences, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic.

Mejtsky, J. 1983. Simulace obecneho systemu vyrobniho strediskas dvema radami technologickych pracovist a obsluhou [in English: Simulation of a production system with two rows of production centers]. *Algorithms 1983, The 7th Algorithm Symposium*, Society of Slovak Mathematicians, Strbske Lake, Slovak Republic.

Mejtsky, G. J. 1985. Backward simulation and multiple-objective optimization of job shop scheduling with zero tardiness and minimum makespan. In *Proceedings of the 1985 Summer Computer Simulation Conference*, 716-720.

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. 1986d. An expert system for job shop scheduling. Dinner Meeting of Treasure Valley Chapter of APICS.

Rego, C., and R. Duarte. 2006. A filter and fan approach for the job shop scheduling problem. School of Business Administration, University of Mississippi, MS.

Schruben, L. W. 1997. Simulation optimization using simultaneous replications and event time dilation. In *Proceedings of the 1997 Winter Simulation Conference,* ed. Winter Simulation Conference Board of Directors, 177-180.

AUTHOR BIOGRAPHY

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