On the Design of an Adaptive Simulated Annealing Algorithm

Vincent A. Cicirello

Computer Science and Information Systems
The Richard Stockton College of New Jersey
Pomona, NJ 08240
cicirelv@stockton.edu

Abstract. In this paper, we demonstrate the ease in which an adaptive simulated annealing algorithm can be designed. Specifically, we use the adaptive annealing schedule known as the modified Lam schedule to apply simulated annealing to the weighted tardiness scheduling problem with sequence-dependent setups. The modified Lam annealing schedule adjusts the temperature to track the theoretical optimal rate of accepted moves. Employing the modified Lam schedule allows us to avoid the often tedious tuning of the annealing schedule; as the algorithm tunes itself for each instance during problem solving. Our results show that an adaptive simulated annealer can be competitive when compared to highly tuned, hand crafted algorithms. Specifically, we compare our results to a state-of-the-art genetic algorithm for weighted tardiness scheduling with sequence-dependent setups. Our study serves as an illustration of the ease with which a parameter-free simulated annealer can be designed and implemented.

1 Introduction

One of the often touted advantages of metaheuristic search algorithms such as simulated annealing [1, 2], tabu search [3, 4], genetic algorithms (GA) [5, 6], and the like, is that they are often argued to be relatively simple to implement for new problems. While in some sense this is generally true, in another sense this "easy to implement" advantage largely ignores the next step of tuning parameters to actually get the algorithm to perform well for your problem.

Most metaheuristics are controlled by several parameters. For example, the simple genetic algorithm (SGA) has a population size, a crossover rate, a mutation rate. GAs that are slightly fancier than the SGA can have additional parameters such as generation gap, scaling window, elitism rate, etc.[1] Likewise, algorithms such as simulated annealing (SA) and tabu search (TS) can also have several control parameters. For example, a run of SA is largely controlled by the value of something called the temperature. The temperature in SA search generally begins at a high value and decays during execution of the search. This temperature decay is controlled by something usually referred to as the annealing schedule. All of the most commonly used annealing schedules have parameters. These parameters, like the parameters of a GA, must be set in some way.

[1] For the sake of this paper, you do not need to know what any of these GA parameters are. The point is that there can be several of them.

Perhaps the most common approach to tuning the parameters of a GA, SA, TS, or some other metaheuristic is, unfortunately, a tedious hand-tuning. That is, the algorithm implementer systematically (or not so systematically) tries out various sets of parameter values on some set of problem instances, and selects the set of parameter values that appears to perform well. Fortunately, there are others who have taken a more rigorous approach to the problem. For example, the earliest formal examination of the problem of parameter tuning in a GA was that of De Jong [7], who studied the performance of the GA on a class of function optimization problems and empirically determined an "optimal" set of parameter values for that class of problem. For decades since that study, De Jong's parameter set is often used by others in designing GAs for new problems. However, De Jong's parameter set is really only relevant for the class of problems considered in his study. Others later followed De Jong's lead and began looking for ways to automate the parameter tuning process. Grefenstette, for example, introduced the idea of using a GA to optimize the parameters of a GA [8]. Since then, a host of related meta-level control parameter optimization approaches have been introduced, both for tuning GA parameters [9–12] as well as for parameters of other problem solving systems [13–16]. Others view the parameter tuning problem as one that should coincide with problem solving. That is, rather than tuning control parameters ahead of time, some argue that problem solving feedback can be used to adapt control parameters to the problem instance at hand (e.g., [17–19]).

In this paper, we too take the approach of adapting control parameters during the search. Specifically, we focus on simulated annealing and use an existing adaptive annealing schedule to demonstrate the ease with which a parameter-free simulated annealer can be implemented. Our empirical results on the NP-Hard optimization problem known as weighted tardiness scheduling with sequence-dependent setups illustrates that such an approach can be competitive with even highly-tuned hand-crafted algorithms.

The remainder of the paper proceeds as follows. In Section 2 we overview simulated annealing, and specifically describe the adaptive annealing schedule known as Modified Lam [19]. Next in Section 3 we provide an overview of the weighted tardiness scheduling problem with sequence-dependent setups that we use for our experimental study. In Section 4 we provide further details on the design of our simulated annealer. We then present empirical results in Section 5 and conclude in Section 6.

2 The Modified Lam Annealing Schedule

Figure 1 provides pseudocode for the typical Simulated Annealing algorithm. Initially, SA begins with some arbitrary solution to the problem. This is often randomly generated, although sometimes it can be the result of some other procedure. SA then considers a series of random moves from the current state S using a neighborhood function. For example, if the problem SA is solving is an ordering problem such as the traveling salesperson problem, then the neighborhood function might be pairwise interchanges of randomly selected cities from the current ordering. If the state that results is of a lower cost than the current state, the move is accepted and this new state S' becomes the current state S. Otherwise, if the new state is of higher cost (lesser quality solution), SA accepts it with probability, $e^{(\mathbf{Cost}(S)-\mathbf{Cost}(S'))/T}$. This is known as the Boltzmann

```
SimulatedAnnealing

$S \leftarrow$ GenerateInitialState
$T \leftarrow$ Some Initial High Temperature,$T_0$
for $i$ from 1 to Evals$_max$
    $S' \leftarrow$ PickRandomState(Neighborhood($S$))
    if Cost($S'$) $<$ Cost($S$)
        $S \leftarrow S'$ {Note: accepting a move }
    else
        $r \leftarrow$ Random$(0, 1)$
        if $r < e^{(\mathbf{Cost}(S) - \mathbf{Cost}(S'))/T}$
            $S \leftarrow S'$ {Note: accepting a move }
        end
    end
    $T \leftarrow T_0 \cdot \alpha^{\lfloor \frac{i}{R} \rfloor}$
end
```

Fig. 1. Pseudocode for a standard Simulated Annealing algorithm using the common geometric annealing schedule.

distribution. That is, SA accepts "bad" moves with a probability that depends on how "bad" that move is and which also depends on the current value of the temperature T.

The problem-solving effectiveness of SA largely depends on the design of what is referred to as the annealing schedule. The annealing schedule refers to the way in which the temperature parameter T is updated during the search. The most commonly used annealing schedule is the geometric schedule, which is also the annealing schedule suggested by the originators of SA [1]. The geometric annealing schedule is defined as:

$$T \leftarrow T_0 \cdot \alpha^{\lfloor \frac{i}{R} \rfloor}, \tag{1}$$

where T_0 is an initially "high" temperature, α is referred to as the *cooling rate*, i is simply the index of the current evaluation and corresponds to the i in the pseudocode of Figure 1, and R is the *round length* which controls how many evaluations are made for each temperature T. The values of T_0, α, and R are all parameters that need to be tuned during the design of the SA.

A well known theoretical result indicates that if the annealing schedule "cools" the temperature at a sufficiently slow rate to maintain the system in a state of thermal equilibrium, then with probability 1 simulated annealing will find the globally optimal solution. The problem with this result is that it requires such a slow cooling rate that searches require a prohibitively long annealing run to settle upon a solution.

Lam and Delosme proposed an approximate thermal equilibrium they call D-equilibrium which balances the trade-off of required computation time and the quality of the solution found by the run of SA [20]. Under certain assumptions about the forms of the distribution of the cost values and the distribution of cost value changes, they analyzed their model and determined the annealing schedule that maintains the system in

```
**SimulatedAnnealing with Modified Lam Annealing Schedule**

$S \leftarrow$ **GenerateInitialState**
$T \leftarrow 0.5$
**AcceptRate** $\leftarrow 0.5$
**for** $i$ from 1 to $\text{Evals}_{\max}$
    $S' \leftarrow$ **PickRandomState**(**Neighborhood**($S$))
    **if** **Cost**($S'$) $<$ **Cost**($S$)
        $S \leftarrow S'$ {Note: accepting a move }
        **AcceptRate** $\leftarrow \frac{1}{500}(499 \cdot \textbf{AcceptRate} + 1)$
    **else**
        $r \leftarrow$ **Random**$(0, 1)$
        **if** $r < e^{(\textbf{Cost}(S) - \textbf{Cost}(S'))/T}$
            $S \leftarrow S'$ {Note: accepting a move }
            **AcceptRate** $\leftarrow \frac{1}{500}(499 \cdot \textbf{AcceptRate} + 1)$
        **else**
            {Note: rejecting a move }
            **AcceptRate** $\leftarrow \frac{1}{500}(499 \cdot \textbf{AcceptRate})$
        **end**
    **end**
    **if** $i/\text{Evals}_{\max} < 0.15$ **then** **LamRate** $\leftarrow 0.44 + 0.56 \cdot 560^{-i/\text{Evals}_{\max}/0.15}$
    **if** $0.15 \leq i/\text{Evals}_{\max} < 0.65$ **then** **LamRate** $\leftarrow 0.44$
    **if** $0.65 \leq i/\text{Evals}_{\max}$ **then** **LamRate** $\leftarrow 0.44 \cdot 440^{-(i/\text{Evals}_{\max} - 0.65)/0.35}$
    **if** **AcceptRate** $>$ **LamRate**
        $T \leftarrow 0.999 \cdot T$
    **else**
        $T \leftarrow T/0.999$
    **end**
**end**
```

Fig. 2. Pseudocode for Simulated Annealing using Boyan's version of Swartz's Modified Lam annealing schedule.

D-equilibrium (i.e., the annealing schedule that optimally balances the computational cost / solution quality trade-off). This "optimal" annealing schedule adjusts the temperature based on the parameter λ which controls the cost-quality trade-off and more importantly based on the current rate of accepted moves. Analyzing their annealing schedule, Lam and Delosme determined that the temperature is reduced the quickest when the probability of accepting a move is equal to 0.44. Since a faster cooling rate leads to a shorter annealing run, Lam and Delosme proposed allowing the size of the neighborhood considered for moves to dynamically fluctuate up or down to match this target move acceptance rate of 0.44 as closely as possible [20]. The general idea is that you can increase the acceptance rate by decreasing the maximum distance from your current state that the algorithm considers for next states; and likewise decrease the acceptance rate by increasing this distance. The motivation for this is the assumption that nearby search states are of similar quality. If you make shorter distance moves, the difference in fitness of your current and potential future states should be smaller on average than if you allow larger moves. Lam and Delosme validated their model on the Traveling Salesperson Problem as well as on a cell placement problem, but the annealing schedule itself is the result of a problem independent theoretical analysis of their approximate thermal equilibrium model.

Swartz later extensively studied Lam and Delosme's annealing schedule and presented a modification [21]. First, he performed additional empirical analysis of the Lam schedule and determined that at the beginning of the search, the move acceptance rate is near 1.0 (i.e., all moves at the beginning are accepted). The acceptance rate rapidly decreases (at an exponential rate) during the first 15% of the run until it reaches the target of 0.44. It remains nearly constant for the next 50% of the run, and then begins an exponential decline to an acceptance rate of 0% by the end of the run. Swartz then noted that adjusting the size of the move neighborhood is not the only way to track this theoretical target acceptance rate. Specifically, he presented an approach for tracking this target acceptance rate by adapting the value of the temperature parameter up or down; whereas Lam and Delosme used a monotonically decreasing temperature schedule and adjusted the neighborhood function. Boyan presented a similar approach to adapting the temperature parameter [19] to track the theoretical "optimal" acceptance rate without modifying the neighborhood function during the search. Swartz's (and Boyan's) Modified Lam annealing schedule has the advantage over the original Lam schedule in that it is problem independent; whereas adapting the neighborhood function would require a problem dependent analysis. In the experiments of this paper, we use Boyan's recommended implementation [19] of Swartz's Modified Lam annealing schedule [21]. Pseudocode for SA using the Modified Lam annealing schedule is shown in Figure 2. It requires no parameter tuning by the implementer. The algorithm uses problem-solving feedback to adjust the temperature T appropriately.

3 Weighted Tardiness Scheduling with Sequence-Dependent Setups

The weighted tardiness scheduling problem with sequence-dependent setups can be defined as follows. The problem instance consists of a set of jobs $J = \{j_0, j_1, \ldots, j_N\}$. Each of the jobs j has a weight w_j, duedate d_j, and process time p_j. Furthermore, $s_{i,j}$ is defined as the amount of setup time required immediately prior to the start of processing job j if it follows job i on the machine. It is not necessarily the case that $s_{i,j} = s_{j,i}$. The 0-th "job" is the start of the problem ($p_0 = 0$, $d_0 = 0$, $s_{i,0} = 0$, $w_0 = 0$). Its purpose is for the specification of the setup time of each of the jobs if sequenced first.

The weighted tardiness objective is to sequence the set of jobs J to minimize:

$$T = \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(c_j - d_j, 0), \tag{2}$$

where T_j is the tardiness of job j; and c_j, d_j are the completion time and duedate of job j. The completion time is equal to the sum of the process times and setup times of all jobs that come before j in the sequence plus the setup time and process time of the job j. Specifically, let $\pi(j)$ be the position in the sequence of job j. Define c_j as:

$$c_j = \sum_{i,k \in J, \pi(i) <= \pi(j), \pi(i) = \pi(k)+1} p_i + s_{k,i}. \tag{3}$$

Single-machine scheduling to optimize the weighted tardiness objective function is NP-Hard even if setups are independent of job ordering [22]. In the case that we consider here, the challenge of the problem is greatly magnified by the sequence-dependent setup constraints. Sen and Bagchi discuss the significance of the challenge that sequence-dependent setups pose for exact solution procedures [23]. Specifically, they discuss how sequence-dependent setups induce a non-order-preserving property of the evaluation function. At the time of their writing, exact solution procedures such as A*, Branch-and-Bound algorithms, and their own GREC [23] for sequencing problems with sequence-dependent setups were limited to solving instances with no more than approximately 25-30 jobs, even for easier objective functions. Problem instances of larger size require turning to inexact solution procedures such as metaheuristic search.

4 Designing An Adaptive Simulated Annealing Algorithm

In this Section, we detail our design for a simulated annealer for the weighted tardiness scheduling problem with sequence-dependent setups. We use the Modified Lam annealing schedule as discussed earlier to design a SA algorithm free of any parameters that require tuning. There are simply two design decisions to make: (1) what neighborhood function to use; and (2) how to generate the initial solution. We consider each of these in turn below.

Choosing the Neighborhood Function: The weighted tardiness scheduling problem with sequence-dependent setups is a one machine sequencing problem. Any ordering of the jobs is a valid solution. The problem is to find a good ordering (i.e., one with a low value of the weighted tardiness objective). Throughout our SA search, the current search state is a complete ordering over the set of jobs. We consider three alternative problem-independent neighborhood operators:

- **Insert:** The insert operator randomly selects a job from the current sequence, removes that job, and reinserts it into a randomly selected new position.
- **Swap:** The swap operator randomly selects two jobs from the current sequence and swaps their positions.
- **Scramble:** The scramble operator randomly selects two positions in the sequence and randomly shuffles the jobs located between those two positions.

These three alternative operators are applicable to any ordering problem and are common choices of neighborhood operator. Later, we discuss empirical results using all three. More sophisticated problem-dependent options are also possible here, but our goal is to illustrate the power of a simple problem-independent adaptive SA.

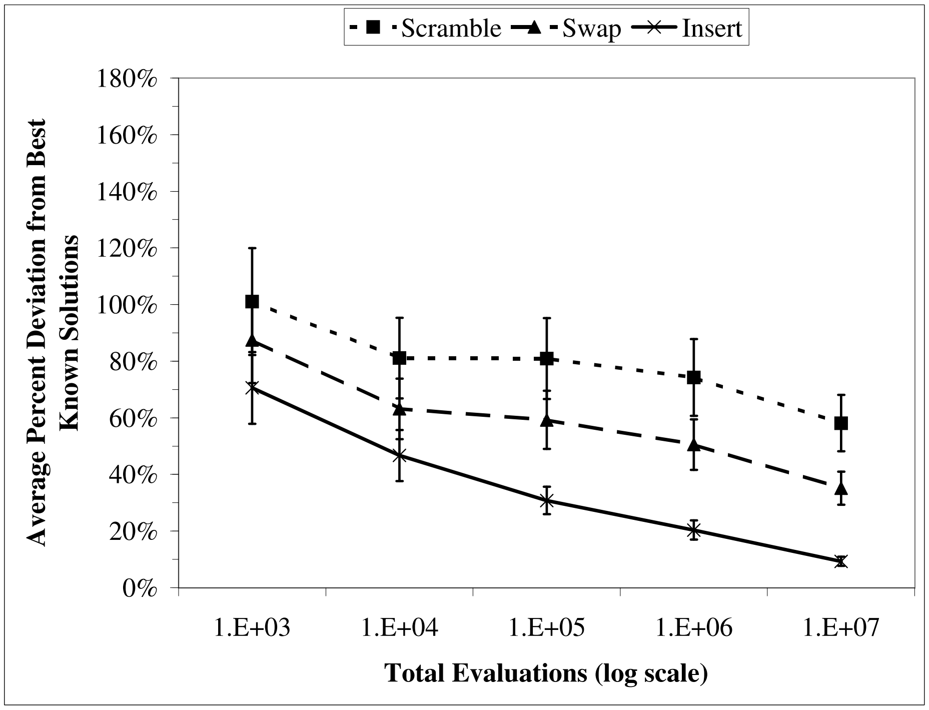

Fig. 3. Comparison of neighborhood operators (scramble, swap, and insert). Shows average percent deviation from best known solutions to the benchmark set for different length runs of the adaptive SA using a heuristically determined initial solution. 99% confidence intervals are shown.

Random Start vs Heuristic Start: The next decision we must make is how to generate the initial configuration of the search. If we wanted our SA approach to be completely problem-independent (aside from the optimization objective), then the SA search could simply begin with a random permutation of the jobs. We consider this option in our experimental results and refer to it as **Random**.

We also, however, consider the use of a problem specific heuristic to seed the initial state of the search. Specifically, we consider the use of the *Apparent Tardiness Cost with Setups (ATCS)* heuristic [24]. ATCS was designed for use as a dispatch policy. That is, its intended use is as a means of selecting the next job for the sequence in a dynamic setting. It has also been used as a search heuristic for stochastic sampling algorithms [25, 26]. ATCS is defined as follows:

$$\text{ATCS}_j(t,l) = \frac{w_j}{p_j} \exp\left(-\frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}} - \frac{s_{l,j}}{k_2 \bar{s}}\right), \tag{4}$$

where t is the current time (or the sum of the process and setup times of the jobs already sequenced); l is the index of the job most recently added to the schedule; $\bar{p}$ is the average processing time of all jobs; and $\bar{s}$ is the average setup time.

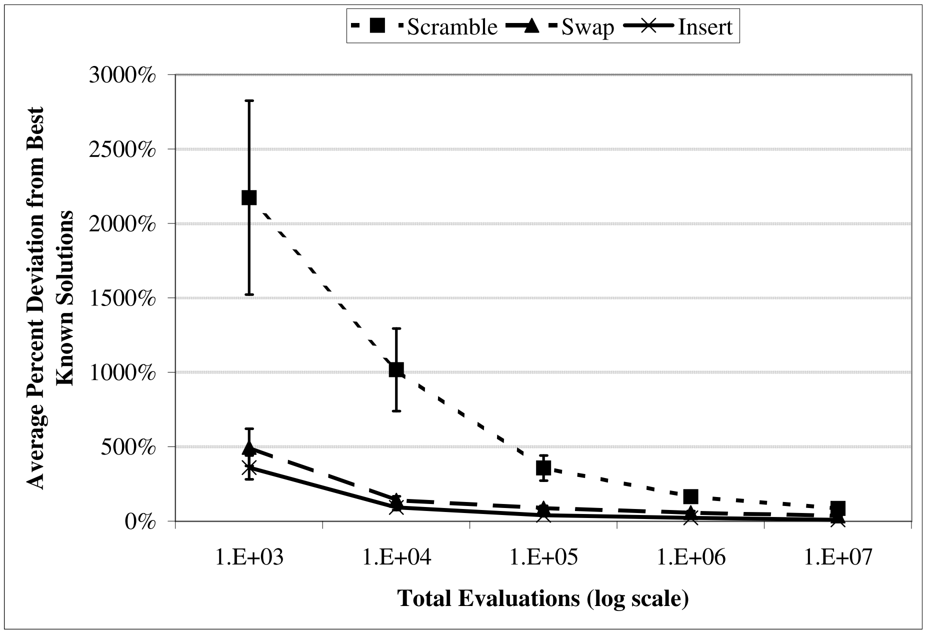

Fig. 4. Comparison of neighborhood operators (scramble, swap, and insert). Shows average percent deviation from best known solutions to the benchmark set for different length runs of the adaptive SA using a random initial solution. 99% confidence intervals are shown.

The k_1 and k_2 are parameters that are automatically tuned based on characteristics of the problem instance as follows.

$$k_1 = \begin{cases} 4.5 + R & \text{if } R \leq 0.5 \\ 6.0 - 2R & \text{otherwise} \end{cases}, \tag{5}$$

where R is called the duedate range factor which is an indicator of how spread out the duedates of the jobs are, and is defined as: $R = \frac{d_{\max} - d_{\min}}{C_{\max}}$ where $d_{\max}$, $d_{\min}$ are the maximum and minimum duedates, and $C_{\max}$ is an estimate of the makespan (or completion time of the last job).

$$k_2 = \frac{\tau}{2\sqrt{\eta}}. \tag{6}$$

where τ is called the duedate tightness factor, defined by $\tau = 1 - \frac{\bar{d}}{C_{\max}}$; and η is the setup time severity factor, defined by $\eta = \frac{\bar{s}}{\bar{p}}$. The τ is an indicator of how urgent the duedates of the problem are and η is an indicator of how large an effect the setup times can have on problem solving.

In our experiments, we consider the ATCS heuristic as an alternative to random initial search states. In this option, ATCS generates a single solution which becomes the seed for the run of SA. We refer to this option in the experiments as **Heuristic**.

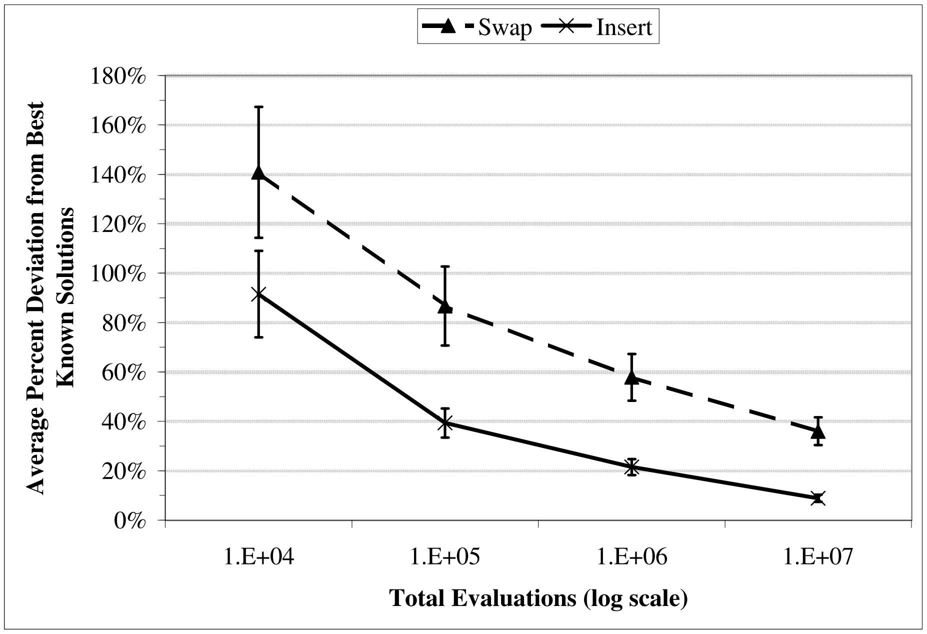

Fig. 5. Comparison of neighborhood operators (swap and insert). Shows average percent deviation from best known solutions to the benchmark set for different length runs of the adaptive simulated annealing algorithm using a random initial solution. 99% confidence intervals are shown.

5 Experimental Results

In this Section, we present experimental results. We begin with comparisons of the three alternative neighborhood operators and the two alternatives for the initial search state. We use an existing set of benchmark instances [26]. This set consists of 120 problem instances, each with 60 jobs, with varying levels of duedate tightness, duedate range, and setup time severity. No currently available approach has been able to provide provably optimal solutions to the problem instances in this benchmark set. There are, however, a number of metaheuristics designed for the problem.

Figure 3 shows a comparison of the three neighborhood operators for the SA using the ATCS heuristic to seed the initial state for the search. Five different length searches are shown with a total number of evaluations of 1000, 10^4, 10^5, 10^6, and 10^7. The total number of evaluations is on the X axis using log scale. The Y axis shows the average percent deviation from the best known solutions to the benchmark instances. Specifically, for each length run and for each choice of operator, we report averages of 1200 runs (10 runs on each of the 120 benchmark instances). 99% confidence intervals are shown. When using the heuristic to seed the starting configuration, the clearly dominant choice of neighborhood operator is the Insert operator.

Figure 4 similarly shows a comparison of the three neighborhood operators, but this time using random permutations to seed the SA search. Again, the results are averages

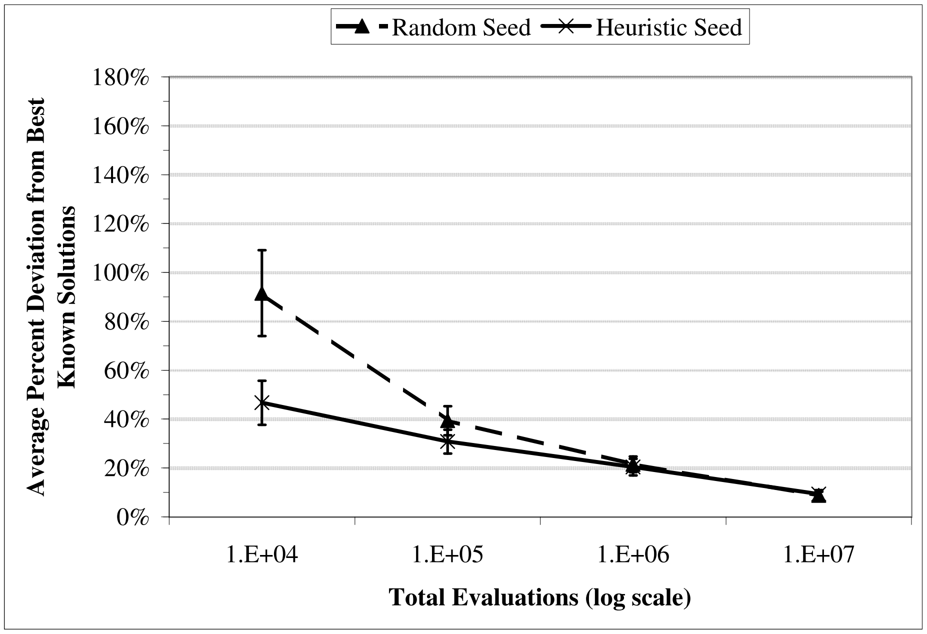

Fig. 6. Comparison of the adaptive simulated annealing algorithm using a heuristically determined initial solution and using a random initial solution. Both versions of the algorithm use the insert neighborhood operator. Shows average percent deviation from best known solutions to the benchmark set for different length runs. 99% confidence intervals are shown.

of 1200 runs and 99% confidence intervals are shown. In this case, due to the particularly slow convergence when using the Scramble operator, it is difficult to pick out a clear winner. We can, however, see that Scramble is clearly dominated by either of the other two neighborhood operators. Note, however, that for especially long searches (e.g., 10^7 total number of evaluations), the performance of the SA using Scramble appears to be catching up to the other two.

To help distinguish the performance of the neighborhood operators when using random starting configurations, Figure 5 illustrates the results of just the Swap and Insert operators, for four different length runs (total number of evaluations of 10^4, 10^5, 10^6, and 10^7). Just like in the case of heuristically determined initial states, we see that the Insert neighborhood operator is the clearly dominating performer when using random starting configurations.

In Figure 6, we consider the use of the ATCS heuristic to seed the search vs using random initial configurations. Both options in this figure use the Insert neighborhood operator. Again, the results are averages of 1200 runs and 99% confidence intervals are shown. For really short searches (e.g., 10^4 total evaluations), using the heuristic to seed the search has a clear advantage. For slightly longer searches (e.g., 10^5 total evaluations), the heuristically seeded SA still has an advantage, though not as strong. For longer searches of length 10^6 evaluations, there is negligible difference; and by 10^7

Table 1. Comparison of the two variations of the adaptive simulated annealer and the current best available metaheuristic for the problem, a genetic algorithm for different length runs. $\%\Delta B$ is the average percent deviation from the best known solutions with 99% confidence intervals shown.

Algorithm	$\%\Delta B$	Time (seconds)
SA (Heuristic) (10^4 evals)	$46.7\% \pm 9.0\%$	0.039
SA (Random) (10^4 evals)	$91.5\% \pm 17.5\%$	0.026
SA (Heuristic) (10^5 evals)	$30.8\% \pm 4.9\%$	0.260
GA [27] (500 generations)	$37.0\% \pm 5.6\%$	0.255
SA (Random) (10^5 evals)	$39.3\% \pm 5.9\%$	0.254
GA [27] (5000 generations)	$19.0\% \pm 2.6\%$	2.389
SA (Heuristic) (10^6 evals)	$20.4\% \pm 3.4\%$	2.568
SA (Random) (10^6 evals)	$21.5\% \pm 3.3\%$	2.543
SA (Random) (10^7 evals)	$8.84\% \pm 1.5\%$	26.061
SA (Heuristic) (10^7 evals)	$9.32\% \pm 1.6\%$	26.220

evaluations all statistical difference vanishes. For long searches, there is no difference between the performance of the completely problem independent SA and the SA that uses a problem dependent heuristic to initialize the search.

Table 1 summarizes a comparison between the two variations of our adaptive simulated annealer and an existing genetic algorithm (GA) for the problem. The specific GA that we compare to is the current best performing metaheuristic for the problem, which uses a permutation crossover operator known as Non-Wrapping Order Crossover (NWOX) and an Insertion mutation operator [27]. The insertion mutation operator is the same as our Insert neighborhood operator. The parameters of the GA of [27] were tuned through an extensive automated tuning process on a set of randomly generated instances distinct from the benchmark set. This GA represents the state-of-the-art in solution procedure for this problem.

The experiments conducted were performed on a Dell Dimension 8400 with a 3.0 GHz Pentium 4 CPU and 1GB memory. Both our SA algorithm and the GA of [27] are implemented in Java 1.5. Similar length runs in average CPU time are considered in the comparisons. Approximately 500 generations of the GA take approximately the same length of time to execute compared to a run of SA with 10^5 total evaluations.

For the shortest length run (the first two rows of Table 1), the SA with heuristic initial solution finds solutions significantly better than when random initial solutions are used ($t = 11.1$ for a T-test with 1199 dof, $p < 2 * 10^{-27}$).

For the next length run (approximately 0.25 seconds of CPU time), the SA with heuristic initial solution again significantly outperforms the SA when random starting configurations are used ($t = 7.3$ for a T-test with $p < 3 * 10^{-13}$). It also significantly outperforms the GA ($t = 6.5$ for a T-test with 1199 dof, $p < 5 * 10^{-11}$).

For the slightly longer length run (approximately 2.5 seconds), the GA is the better performer compared to the better of the two variations of our adaptive SA at a significance level $p < 0.02$ (T-test with 1199 dof, $t = 2.1$). Only marginal significance is seen between the two variations of the SA for this length run ($t = 1.4$ for a T-test with 1199 dof, $p < 0.09$). But the adaptive SA is still competitive with this GA, which is impressive given that this GA is the currently dominant performer for the problem.

Data for the GA was not available for a comparable length run of the longest run of the SA that we consider (approximately 26 seconds per run). But by this length run, little significance remains between initializing the SA with a heuristic as compared to random initialization ($t = 1.0$ for a T-test with 1199 dof, $p < 0.16$).

6 Conclusions

In this paper, we illustrated the ease with which a parameter-free simulated annealing search could be designed using an existing adaptive annealing schedule. Specifically, we used the Modified Lam annealing schedule which has no parameters to tune, and which instead adapts the temperature parameter during the search to track the theoretical "optimal" rate of accepted moves during the course of the search. No parameter tuning during the algorithm design and implementation phases was necessary. The only design decisions that need be made are how to initialize the starting solution and what neighborhood operator to use.

We specifically used a computationally hard scheduling problem known as weighted tardiness scheduling with sequence-dependent setups for our empirical analysis. For short searches, the adaptive SA outperforms the current best metaheuristic for this NP-Hard scheduling problem; while for slightly longer searches, the highly-tuned GA is still better although our SA is competitive. During the course of this research, the two variations of the SA were able to improve upon the best known solutions to several of the benchmark instances. The objective values of these new best known solutions can be found in Appendix A. The simplicity of a problem-independent self-adaptive SA is a worthy competitor to even a highly optimized and customized procedure.

Additionally, our experimental results show that for short searches, there is a benefit to using a problem dependent heuristic to initialize the starting configuration of the SA search. This benefit disappears for longer searches where little, if any, difference is seen in the results, illustrating that a completely problem independent adaptive SA (aside from the optimization objective function) can be effective even for especially hard scheduling problems.

Acknowledgements

This research was supported by the Richard Stockton College's Research and Professional Development Program.

A Appendix: New Best Known Solutions to Benchmarks

During the course of the experimental part of our research, several new best known solutions to the benchmark instances were found. The objective values of those new best known solutions are listed here. The first column is the instance number. This is followed by the value of the new best known solution and the variation of SA that found it. SA-H is for the SA initialized by the heuristic. SA-R is for the SA initialized randomly. Both versions used the Insert neighborhood operator.

Instance	Old Best	New Best	Found By
1	866	790	SA-H
2	5907	5824	SA-H
4	7251	6840	SA-H
5	5233	5017	SA-R
6	8131	7824	SA-H
7	4130	3933	SA-H
8	299	298	SA-R
9	7421	7059	SA-H
10	2451	2125	SA-H
11	5263	5088	SA-H
14	3941	3761	SA-R
15	2915	2039	SA-R
16	6711	5559	SA-R
17	461	387	SA-R
18	2514	1918	SA-R
19	279	239	SA-H
20	4193	3805	SA-R
24	1103	1092	SA-H
27	74	57	SA-R
30	333	215	SA-R
37	1757	1008	SA-R
41	73176	71242	SA-H
42	61855	59493	SA-H
43	149990	147737	SA-H
44	37584	36265	SA-H
45	61573	59696	SA-R
46	37112	36175	SA-R
47	76629	74389	SA-H
48	67633	65129	SA-R
49	81220	79656	SA-R
50	33877	32777	SA-R
51	58574	54707	SA-H
52	105367	100793	SA-H
53	95452	94394	SA-R
55	74342	72420	SA-R
56	84614	80258	SA-H
57	70414	68535	SA-H
58	53460	46978	SA-R
59	59060	56181	SA-H
60	73328	68395	SA-R
61	78667	76769	SA-R
62	45522	44781	SA-H
63	78822	76059	SA-H
64	94165	93079	SA-H
65	132306	127713	SA-R
66	62266	59717	SA-H
67	29443	29394	SA-R
68	23614	22653	SA-R
69	72238	71534	SA-H
70	78137	76140	SA-R
71	161233	155036	SA-R
72	51634	49886	SA-H
73	34859	30259	SA-H
74	38292	32083	SA-R
75	24429	21602	SA-R
76	66656	57593	SA-H
77	40558	35380	SA-H
78	24030	21443	SA-H
79	125824	121434	SA-H
80	25212	20221	SA-H
81	386782	385918	SA-H
82	412721	410550	SA-H
83	466070	459939	SA-R
84	331659	330186	SA-R
85	558556	557831	SA-R
86	365783	364474	SA-R
87	401049	400264	SA-R
88	436855	434176	SA-R
89	415433	411810	SA-H
90	406939	403623	SA-R
91	347175	344428	SA-R
92	365779	363388	SA-R
94	336299	334180	SA-H
95	527909	524463	SA-R
97	420287	418995	SA-H
99	374781	374607	SA-R
101	354906	353575	SA-R
102	496131	495094	SA-H
104	361718	358738	SA-R
105	456364	450806	SA-R
106	459925	457284	SA-H
107	355537	353564	SA-H
108	467651	462675	SA-R
109	415817	413918	SA-H
110	421282	419014	SA-R
111	350723	348796	SA-H
112	377418	375952	SA-H
113	263200	261795	SA-H
114	473197	471422	SA-H
116	540231	537593	SA-H
117	518579	507188	SA-H
119	583947	581119	SA-H

References

1. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220** (May 1983) 671–680
2. van Laarhoven, P.J.M., Aarts, E.H.L.: Simulated Annealing: Theory and Applications. D. Reidel Publishing Company (Kluwer Academic Publishers) (1987)
3. Glover, F.: Tabu search – part I. ORSA Journal on Computing **1**(3) (Summer 1989) 190–206
4. Glover, F.: Tabu search – part II. ORSA Journal on Computing **2**(1) (Winter 1990) 4–32
5. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press (1975) Second Edition: MIT Press, 1992.
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley (1989)
7. De Jong, K.A.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan, Ann Arbor, MI (1975)
8. Grefenstette, J.: Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics **16**(1) (Jan-Feb 1986) 122–128
9. Cicirello, V.A., Smith, S.F.: Modeling GA performance for control parameter optimization. In Whitley, D., Goldberg, D., Cantú-Paz, E., Spector, L., Parmee, I., Beyer, H., eds.: GECCO-2000: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers (8-12 July 2000) 235–242
10. Cao, Y.J., Wu, Q.H.: Optimization of control parameters in genetic algorithms: A stochastic approach. International Journal of Systems Science **30**(5) (May 1999) 551–559
11. Wu, S.J., Chow, P.T.: Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. Engineering Optimization **24**(2) (1995) 137–159
12. Bramlette, M.F.: Initialization, mutation and selection methods in genetic algorithms for function optimization. In Belew, R.K., Booker, L.B., eds.: Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, CA, Morgan Kaufmann (1991) 100–107
13. Koch, T.E., Scheer, V., Wakunda, J., Zell, A.: A parallel, hybrid meta-optimization for finding better parameters of an evolution strategy in real world optimization problems. In Wu, A.S., ed.: Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program. (July 2000) 17–19 Evolutionary Computation and Parallel Processing Workshop.
14. Campos, M., Bonabeau, E., Théraulaz, G., Deneubourg, J.: Dynamic scheduling and division of labor in social insects. Adaptive Behavior **8**(2) (2000) 83–96
15. Morley, D.: Painting trucks at general motors: The effectiveness of a complexity-based approach. In: Embracing Complexity: Exploring the Application of Complex Adaptive Systems to Business, The Ernst and Young Center for Business Innovation (1996) 53–58
16. De Jong, K.: Adaptive system design: A genetic approach. IEEE Transactions on Systems, Man, and Cybernetics **10**(9) (1980) 566–574
17. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation **3**(2) (July 1999) 124–141
18. Cicirello, V.A.: Boosting Stochastic Problem Solvers Through Online Self-Analysis of Performance. PhD thesis, The Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA (21 July 2003) Also available as technical report CMU-RI-TR-03-27.
19. Boyan, J.A.: Learning Evaluation Functions for Global Optimization. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania (1998)

20. Lam, J., Delosme, J.: Performance of a new annealing schedule. In: Proceedings of the 25th ACM/IEEE Design Automation Conference. (1988) 306–311
21. Swartz, W.P.: Automatic Layout of Analog and Digital Mixed Macro/Standard Cell Integrated Circuits. PhD thesis, Yale University (May 1993)
22. Morton, T.E., Pentico, D.W.: Heuristic Scheduling Systems: With Applications to Production Systems and Project Management. John Wiley and Sons (1993)
23. Sen, A.K., Bagchi, A.: Graph search methods for non-order-preserving evaluation functions: Applications to job sequencing problems. Artificial Intelligence **86**(1) (September 1996) 43–73
24. Lee, Y.H., Bhaskaran, K., Pinedo, M.: A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transactions **29** (1997) 45–52
25. Cicirello, V.A., Smith, S.F.: Amplification of search performance through randomization of heuristics. In Van Hentenryck, P., ed.: Principles and Practice of Constraint Programming – CP 2002: 8th International Conference, Proceedings. Volume LNCS 2470 of Lecture Notes in Computer Science., Springer-Verlag (7-13 September 2002) 124–138
26. Cicirello, V.A., Smith, S.F.: Enhancing stochastic search performance by value-biased randomization of heuristics. Journal of Heuristics **11**(1) (January 2005) 5–34
27. Cicirello, V.A.: Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In M. Keijzer et al, ed.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'06). Volume 2., ACM Press (8-12 July 2006) 1125–1131