Learning to Recognize (Un)Promising Simulated Annealing Runs: Efficient Search Procedures for Job Shop Scheduling and Vehicle Routing

Norman M. Sadeh
Carnegie Mellon University

Yoichiro Nakakuki
NEC

Sam R. Thangiah
Slippery Rock University

Abstract:
Simulated Annealing (SA) procedures can potentially yield near-optimal solutions to many difficult combinatorial optimization problems, though often at the expense of intensive computational efforts. The single most significant source of inefficiency in SA search is the inherent stochasticity of the procedure, typically requiring that the procedure be rerun a large number of times before a near-optimal solution is found. This paper describes a mechanism that attempts to learn the structure of the search space over multiple SA runs on a given problem. Specifically, probability distributions are dynamically updated over multiple runs to estimate at different checkpoints how promising a SA run appears to be. Based on this mechanism, two types of criteria are developed that aim at increasing search efficiency: (1) a *cutoff criterion* used to determine when to abandon unpromising runs and (2) *restart criteria* used to determine whether to start a fresh SA run or restart search in the middle of an earlier run. Experimental results obtained on a class of complex job shop scheduling problems show (1) that SA can produce high quality solutions for this class of problems, if run a large number of times, and (2) that our learning mechanism can significantly reduce the computation time required to find high quality solutions to these problems. The results also indicate that, the closer one wants to be to the optimum, the larger the speedups. Similar results obtained on a smaller set of benchmark Vehicle Routing Problems with Time Windows (VRPTW) suggest that our learning mechanisms should help improve the efficiency of SA in a number of different domains.
Key Words: Simulated annealing, learning, scheduling, vehicle routing.

1   Introduction

Simulated Annealing (SA) is a general-purpose search procedure that generalizes iterative improvement approaches to combinatorial optimization by sometimes accepting transitions to lower quality solutions to avoid getting trapped in local minima [12, 3]. SA procedures have been successfully applied to a variety of combinatorial optimization problems, including Traveling Salesman Problems [3], Graph Partitioning Problems [10], Graph Coloring Problems [11], Vehicle Routing Problems [19], Design of Integrated Circuits, Minimum Makespan Scheduling Problems [14, 18, 31] as well as other complex scheduling problems [35], often producing near-optimal solutions, though at the expense of intensive computational efforts.

The single most significant source of inefficiency in SA search is the inherent stochasticity of the procedure, typically requiring that the procedure be rerun a large number of times before a near-optimal solution is found. Glover et al. developed a set of "Tabu" mechanisms that can increase the efficiency of SA and other neighborhood search procedures by maintaining a selective history of search states encountered earlier during the *same run* [7]. This history is then used to dynamically derive "tabu restrictions" or "aspirations", that guide search, preventing it, for instance, from revisiting areas of the search space it just explored. This paper describes a complementary mechanism that attempts to learn the structure of the search space *over multiple runs* of SA on a given problem. Specifically, we introduce a mechanism that attempts to predict how (un)promising a SA run is likely to be, based on probability distributions that are refined ("learned") over multiple runs. The distributions, which are built at different checkpoints, each corresponding to a different value of the temperature parameter used in the procedure, approximate the cost reductions that one can expect if the SA run is continued below these temperatures. Two types of criteria are developed that aim at increasing search efficiency by exploiting these distributions:

- *A Cutoff Criterion*: This criterion is used to detect runs that are unlikely to result in an improvement of the best solution found so far and, hence, should be abandoned;

- *Restart Criteria*: When completing a run or abandoning an unpromising one, these criteria help determine whether to

start a fresh SA run or restart search in the middle of an earlier promising run.

The techniques presented in this paper have been applied to a class of complex job shop scheduling problems first described in [24]. Problems in this class require scheduling a set of jobs that each need to be completed by a possibly different due date. The objective is to minimize the sum of tardiness and inventory costs incurred by all the jobs. This class of problems is known to be NP-complete and is representative of a large number of actual scheduling problems, including Just-In-Time factory scheduling problems [24, 25]. Experimental results indicate (1) that SA can produce high quality solutions for this class of problems, if run a large number of times, and (2) that our learning mechanism can yield significant reductions in computation time. The results further indicate that, the closer one wants to be to the optimum, the larger the speedups. To test the generality of our learning mechanism, we also ran similar experiments with a SA procedure developed for the Vehicle Routing Problem with Time Windows (VRPTW). Results on this second class of problems suggest that, indeed, our learning mechanism can help improve the efficiency of SA search in different domains.

The balance of this paper is organized as follows. Section 2 quickly reviews fundamentals of SA search. Section 3 analyzes the behavior of typical SA runs and introduces a mechanism that aims at learning to recognize (un)promising runs on a given problem, using the concept of *Expected Cost Improvement Distributions* ($ECID$). In Section 4, we use $ECID$ distributions to develop a *cutoff criterion* to determine when to abandon unpromising runs. Section 5 presents three *restart criteria* based $ECID$ distributions. Experiments obtained on a set of benchmark job shop scheduling problems with tardiness and inventory costs and a smaller set of complex VRPTW benchmarks are reported in Section 6. A summary is provided in Section 7.

2 Simulated Annealing Search

Figure 1 outlines the main steps of a SA procedure designed to find a solution $x \in S$ that minimizes a real-valued function, $cost(x)$. The procedure starts from an initial solution x_0 (randomly drawn from S) and iteratively moves to other neighboring solutions, as deter-

mined by a neighborhood function, $neighbor(x)$, while remembering
the best solution found so far (denoted by s). Typically, the proce-
dure only moves to neighboring solutions that are better than the
current one. However, the probability of moving from a solution
x to an inferior solution x' is greater than zero, thereby allowing
the procedure to escape from local minima. $rand()$ is a function
that randomly draws a number from a uniform distribution on the
interval $[0, 1]$. The so-called temperature, T, of the procedure is a
parameter controlling the probability of accepting a transition to a
lower quality solution. It is initially set at a high value, T_0, thereby
frequently allowing such transitions. If, after N iterations, the best
solution found by the procedure has not improved, the tempera-
ture parameter T is decremented by a factor α ($0 < \alpha < 1$). One
motivation for progressively lowering the temperature is to obtain
convergence. Additionally, as the procedure slowly moves towards
globally better solutions, accepting transitions to lower quality so-
lutions becomes increasingly less attractive. When the temperature
drops below a preset level T_1, the procedure stops and s is returned
(not shown in Figure 1).

```
T = T₀; x = x₀ (∈ S); min = ∞;
while (T > T₁)  {
        for i = 1, N   {
                x' = neighbor(x);
                if (cost(x') < cost(x))   x = x';
                else if (rand() < exp{(cost(x) − cost(x'))/T})   x = x';
                if(cost(x) < min)   min = cost(x), s = x;
        }
        if (Min was not modified in the above loop)   T = T * α;
}
```

Fig. 1 Basic Simulated Annealing Procedure.

Fig. 2 depicts the cost distribution of the best solutions returned
by 300 SA runs on a typical combinatorial optimization problem
— a job shop scheduling problem from a set of benchmarks to be

described in Section 6.
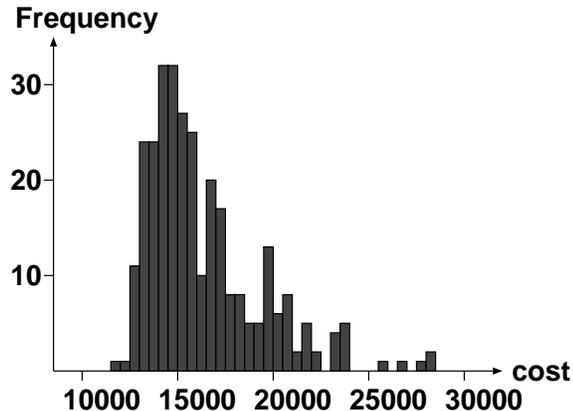
Fig. 2 Cost Distribution of the Best Solutions Found by 300 SA Runs.

The optimal solution for this problem is believed to have a cost around $11,500$ — the value in itself is of no importance here. Figure 2 indicates that, if run a large number of times, SA is likely to eventually find an optimal solution to this problem. It also shows that, in many runs, SA gets trapped in local minima with costs much higher than the global minimum. For instance, 60% of the runs produce solutions with a cost at least 30% above the global minimum. This suggests that, if rather than completing all these unsuccessful runs, one could somehow predict when a run is likely to lead to a highly sub-optimal solution and abandon it, the efficiency of SA could be greatly enhanced. The following section further analyzes the behavior of typical SA runs and proposes a mechanism which, given a problem, aims at learning to recognize (un)promising SA runs.

3   Learning To Recognize (Un)promising SA Runs

Figure 3 depicts the behavior of a SA procedure on two different scheduling problems (from the set of benchmarks used in Section 6). For each problem, the figure depicts five SA runs, plotting the cost of the best solution, s, as the temperature of the procedure is progressively lowered — temperatures are shown in log scale, which is almost equivalent to computation time in linear scale. SA behaves

very differently on these two problems. For instance, in Problem #1, the range of final solutions is relatively narrow, while in Problem #2 it is much wider. Another differentiating factor is the behavior of the procedure at low temperatures. It seems that for Problem #1, the quality of a run can already be estimated quite accurately at $T = 50$ (e.g. the best run at $T = 50$ remains best at lower temperatures), while this is less so for Problem #2.
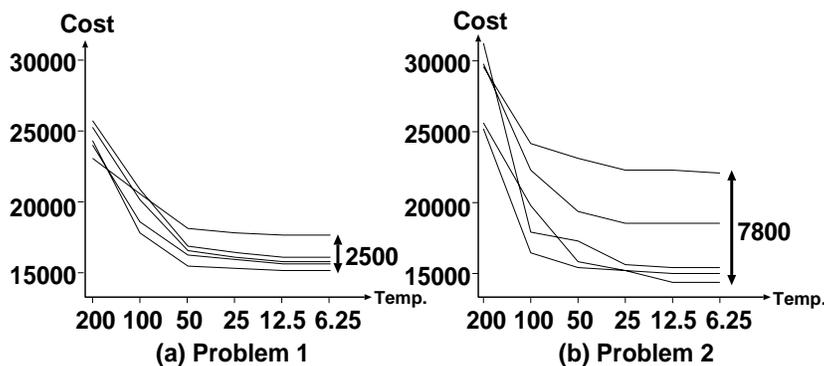
Fig. 3 Cost reductions in five SA runs on two different problems.

Clearly, such properties are not intrinsic to a problem itself. They could change if a different neighborhood structure or a different cooling profile was selected, as these parameters can affect the types of local optima encountered by the procedure and the chance that the procedure extricates itself from these local optima below a given temperature. While, in general, it may be impossible to find a SA procedure that reliably converges to near-optimal solutions on a wide class of problems, we can try to design adaptive SA procedures which, given a problem, can learn to recognize (un)promising runs and improve their performance over time. Below, we present a mechanism, which, given a problem, attempts to "learn" at different checkpoint temperatures the distribution of cost improvements that one can hope to achieve by continuing search below these temperatures.

Specifically, we postulate that, given a problem and a checkpoint temperature $T = t$, the distribution of the cost improvement that

is likely to be achieved by continuing a run below t can be approximated by a normal distribution. Using performance data gathered over earlier runs on a same problem, it is possible to approximate these *Expected Cost Improvement Distributions (ECID)* for a set C of checkpoint temperatures and use these distributions to identify (un)promising runs.

Formally, given a combinatorial optimization problem and a SA procedure for that problem, we define c_i^t as the cost of the best solution, s, at check point t in the i-th run and c_i^0 as the cost of the best solution obtained at temperature $T = T_1$ in the i-th execution. When the $(n+1)$-th run reaches a checkpoint temperature t, the *ECID* below t is approximated as a normal distribution $N[\mu_n^t, \sigma_n^t]$, whose average, μ_n^t, and standard deviation, σ_n^t, are given by:

$$\mu_n^t = \frac{\sum_{i=1}^n (c_i^t - c_i^0)}{n}, \quad \sigma_n^t = \sqrt{\frac{\sum_{i=1}^n \{(c_i^t - c_i^0) - \mu_n^t\}^2}{n-1}}$$

By incrementally refining these estimators over multiple runs, this mechanism can in essence "learn" to recognize (un)promising SA runs. The following sections successively describe a cutoff criterion and three restart criteria based on *ECID* distributions.

4   A Cutoff Criterion

Suppose that, in a sixth run on Problem #1, the best solution obtained at checkpoint $T = 100$ is solution **A** $-$ Figure 4(a). At this checkpoint, the distribution of c_6^0 $-$ the cost of the best solution that will have been found if the run is completed $-$ can be approximated by the normal distribution $N[c_6^{100} - \mu_5^{100}, \sigma_5^{100}]$. This distribution, represented in Fig. 4(a), suggests that, if continued, the current run has a good chance of improving the current best solution, **x**. Suppose that based on this analysis, the run continues until the next checkpoint, $T = 50$, and that the best solution found by the run when it reaches that temperature is **A'**. At this point, a new distribution of c_6^0 can be computed to check how the run is doing. This distribution, $N[c_6^{50} - \mu_5^{50}, \sigma_5^{50}]$ is shown in Figure 4(b). It appears much less promising than the one at $T = 100$. Now, the chances of improving the current best solution, **x**, appear remote:
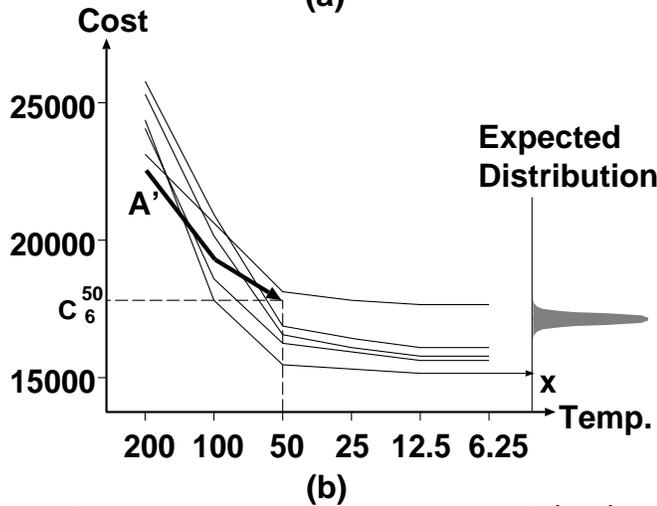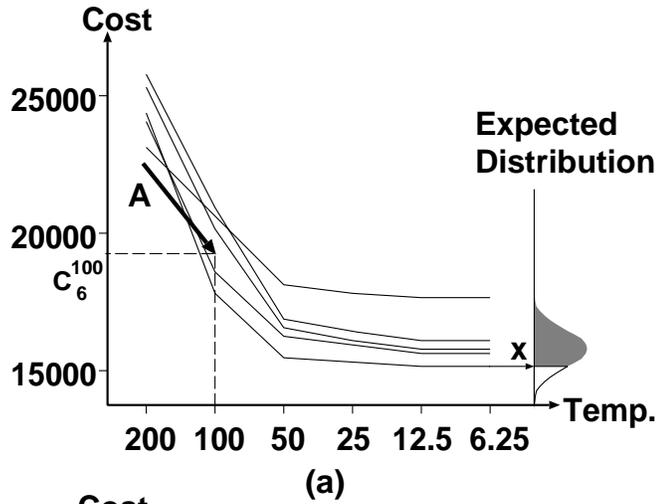
it probably does not make sense to continue this run.

Fig. 4 Expected Cost Improvement Distributions at
T=100 and T=50.

Formally, when the $(n+1)$-th run reaches a checkpoint temperature t, a cutoff criterion is used to determine whether or not to continue this run. In the study reported in Section 6, we use a cutoff criterion of the form:

$$\frac{(c_{n+1}^t - \mu_n^t) - x_n}{\sigma_n^t} > threshold$$

where x_n is the cost of the best solution found during the previous n runs and $threshold$ is a threshold value. If the inequality holds,

the current run is abandoned. For example, if $threshold = 3$ (the value used in our experiments) and the cutoff inequality holds at a given checkpoint temperature t, the probability of improving x_n by continuing the run below t is expected to be less than 1% [1].

5 Three Restart Criteria

Whenever a run is completed or abandoned, two options are available: either start a fresh new annealing run *or*, instead, restart an earlier (promising) run, using a different sequence of random numbers ("reannealing"). In total, if reannealing is constrained to start from one of the checkpoint temperatures, there are up to $n \cdot |C| + 1$ possible options, where n is the number of earlier runs and $|C|$ the number of checkpoints in set C. Below, we describe three "restart criteria" that aim at selecting among these options so as to maximize the chances of quickly converging to a near-optimal solution.

5.1 Maximum Cost Reduction Rate Criterion

When considering several points from which to restart search, two factors need to be taken into account: (1) the likelihood that restarting search from a given point will lead to an improvement of the current best solution and (2) the time that it will take to complete a run from that point. Restarting from a low temperature will generally bring about moderate solution improvements, if any, while requiring little CPU time. Starting fresh new runs or restarting from higher temperatures can lead to more significant improvements, though less consistently and at the cost of more CPU time. In general, the cost improvements that can be expected from different temperatures will vary from one problem to another, as illustrated in Figure 3 (and as formalized by $ECID$ distributions).

A natural restart criterion is one that picks the restart point expected to maximize the rate at which the cost of the current best solution will improve. For each restart candidate O_k (fresh annealing or reannealing), this can be approximated as the expected cost reduction (in the best solution), if search is restarted from O_k, divided by the expected CPU time required to complete a run from that restart point. Below, we use $R(O_k)$ to denote this approximation of the expected cost reduction rate, if search is restarted from

O_k:

$$R(O_k) = \frac{expected\text{-}reduction(O_k)}{expected\text{-}CPU(O_k)}$$

where $expected\text{-}reduction(O_k)$ is the expected cost reduction at the
end of a run starting from O_k and $expected\text{-}CPU(O_k)$ is the CPU
time that this run is expected to require. $expected\text{-}CPU(O_k)$ can be
approximated as the average time required to complete earier runs
from O_k's temperature. $expected\text{-}reduction(O_k)$ can be evaluated
using $ECID$ distributions, as detailed below.

Given a reannealing point O_k at checkpoint temperature t and n
earlier SA runs completed from t or above, $expected\text{-}reduction(O_k)$
can be approximated as:

$$expected\text{-}reduction(O_k) = \int_{LB}^{x_n} \{P_{nk}^t(x) \cdot (x_n - x)\}dx$$

where $P_{nk}^t(x)$ is the density function of the normal distribution
$N[c_k - \mu_n^t, \sigma_n^t]$, c_k is the cost of O_k's best solution[1], x_n is the cost
of the best solution obtained over the first n runs, and LB is a
lower-bound on the optimal solution[2]

Similarly, if O_k is a fresh SA run, $expected\text{-}reduction(O_k)$ can
be approximated as:

$$expected\text{-}reduction(O_k) = \int_{LB}^{x_n} \{P_n(x) \cdot (x_n - x)\}dx$$

where $P_n(x)$ is the density function of the normal distribution $N[\mu_n^0, \sigma_n^0]$,
with

$$\mu_n^0 = \frac{\sum_{i=1}^n c_i^0}{n}, \quad \sigma_n^0 = \sqrt{\frac{\sum_{i=1}^n \{c_i^0 - \mu_n^0\}^2}{n-1}}.$$

5.2   Randomized Criterion

One possible problem with the above criterion is its high sensitivity
to possible inaccuracies in approximations of $ECID$ distributions.
This can be a problem when the number of earlier runs is still
small. When inaccurate $ECID$ distributions lead the criterion to
choose a poor restart point, the procedure may take a long time
before it improves the quality of the current best solution. In the

---

[1]To be consistent, if O_k correponds to the i-th SA run, $c_k = c_i^t$, as defined
in Section 3.

[2]In the experiments reported in this paper, LB was simply set to 0.

meantime, it may keep on coming back to the same poor restart point. For this reason, it is tempting to use a randomized version of the criterion. One such variation involves randomly picking from a set of promising restart points, $H = \{O_l | R(O_l) > \beta \cdot Max\{R(O_k)\}\}$, while assuming that each element in H has the same probability, $1/|H|$, of being selected. β is a constant whose value is between 0 and 1.

5.3 Hybrid Criterion

A third alternative involves keeping some level of stochasticity in the restart criterion, while ensuring that more promising restart points have a higher chance of being selected. This is done by selecting restart points in H according to a Boltzmann distribution that assigns to each element $O_l \in H$ a probability

$$p(O_l) = \frac{exp(R(O_l)/\tau)}{\sum_{O_k \in H} exp(R(O_k)/\tau))}$$

Here, τ is a positive constant. If τ is very large, this method becomes equivalent to the randomized criterion described in subsection 5.2. If $\tau \approx 0$, this criterion becomes similar to the criterion of subsection 5.1. A similar distribution is used in the Q-learning algorithm described in [33].

6 Performance Evaluation

To evaluate performance of our cutoff and restart criteria, we consider two classes of particularly challenging combinatorial optimization problems: the Just-In-Time Job Shop Scheduling Problem and the Vehicle Routing Problem with Time Windows. Below, we successively provide a succinct description of each domain, the base SA procedure we developed for it and its performance on benchmark problems, when run with and without our cutoff and restart criteria.

6.1 The Job Shop Scheduling Problem with Tardiness and Inventory Costs

The Just-In-Time Job Shop Scheduling Problem or Job Shop Scheduling Problem with Tardiness and Inventory Costs was first introduced in [24]. The problem assumes a factory, in which a set of

10

jobs, $J = \{j_1, j_2, \cdots, j_n\}$, has to be scheduled on a set of resources, $RES = \{R_1, R_2, \cdots, R_m\}$. Each job requires performing a set of operations $O^l = \{O_1^l, O_2^l, \cdots O_{n_l}^l\}$ and, ideally, should be completed by a given due date, dd_l, for delivery to a customer. Precedence constraints specify a complete order in which operations in each job have to be performed. By convention, it is assumed that operation O_i^l has to be completed before operation O_{i+1}^l can start ($i = 1, 2, \cdots, n_l - 1$). Each operation O_i^l has a deterministic duration du_i^l and requires a resource $R_i^l \in RES$. Resources cannot be assigned to more than one operation at a time. The problem is to find a feasible schedule that minimizes the sum of tardiness and inventory costs of all the jobs ("Just-In-Time" objective). This problem is known to be NP-complete [24] and is representative of a large number of actual factory scheduling problems where the objective is to meet customer demand in a timely yet cost effective manner. Additional details on this model can be found in [24].

Experimental results reported below suggest that a good neighborhood function for this problem can be obtained by randomly applying one of the following three operators to the current schedule[3]:

- SHIFT-RIGHT: randomly select a "right-shiftable" operation and increase its start time by one time unit[4].

- SHIFT-LEFT (mirror image of SHIFT-RIGHT): randomly select a "left-shiftable" operation and decrease its start time by one time unit.

- EXCHANGE: randomly select a pair of adjacent operations on a given resource and permute the order in which they are processed by that resource. Specifically, given two consecutive operations, A and B on a resource R, with A preceding B in the current solution, the exchange operator sets the new start time of B to the old start time of A and the new end time of

---

[3]In the scheduling jargon, the Just-In-Time objective considered in this study is known to be irregular[15]. Prior applications of SA to job shop scheduling have only considered regular objectives such as Minimum Makespan. It can be shown that the neighborhoods used in these earlier studies are not adequate to deal with irregular objectives such as the one considered here [26].

[4]An operation is said to be "right(left)-shiftable" if its start time can be increased (decreased) by one time unit without overlapping with another operation.

A to the old end time of B [5].

In our experiments, the probability of picking the EXCHANGE operator was empirically set to 3/7 while the probabilities of picking SHIFT-RIGHT or SHIFT-LEFT were each set to 2/7. Additionally, the values of parameters in the SA procedure (see Figure 1) were set as follows: $T_0 = 700$, $T_1 = 6.25$, $N = 200,000$ and $\alpha = 0.85$.

The performance of this SA procedure has been evaluated in a comparison against 39 combinations of well-regarded dispatch rules and release policies previously used to assess the performance of the Sched-Star [16] and Micro-Boss [24, 25] systems on a set of 40 benchmark problems similar to the ones described in [24, 25]. The 40 benchmarks consisted of 8 problem sets obtained by adjusting three parameters to cover a wide range of scheduling condition: an average due date parameter (tight versus loose average due date), a due date range parameter (narrow versus wide range of due dates), and a parameter controlling the number of major bottlenecks (in this case one or two). For each parameter combination, a set of 5 scheduling problems was randomly generated, thereby resulting in a total of 40 problems. Each problem involved 20 jobs and 5 resources for a total of 100 operations. On average, when compared against the best solution found on each problem by the 39 combinations of dispatch rules and release policies, SA reduced schedule cost by 15% (average over 10 SA runs). When comparing the best solution obtained in 10 SA runs against the best solution obtained on each problem by the 39 combinations of dispatch rules and release policies, SA produced schedules that were 34% better. However, while running all 39 combinations of dispatch rules and release policies takes a few CPU seconds on a problem, a single SA run takes about 3 minutes on a DECstation 5000/200 running C. Additional details on these experiments can be found in [26].

6.1.1 Empirical Evaluation of Cutoff and Restart Criteria

We now turn to the evaluation of the cutoff and restart criteria presented in this paper and compare the performance of five variations of the SA procedure presented in 6.1:

---

[5]In our implementation, exchanging two operations is allowed even if a precedence constraint is violated in the process. Precedence constraint violations are handled using large artificial costs that force the SA procedure to quickly get rid of them [26].

- **N-SA**: regular SA, as described in 6.1 (no learning).

- **P-SA**: SA with cutoff criterion.

- **B-SA**: SA with cutoff and Maximum Cost Reduction Rate restart criteria.

- **R-SA**: SA with cutoff and randomized restart criteria ($\beta = 0.5$).

- **H-SA**: SA with cutoff and hybrid restart criteria ($\beta = 0.5$ and $\tau = 1$).

When running P-SA, B-SA, R-SA, and H-SA, the cutoff and/or restart criteria were only activated after 5 complete SA runs to allow for the construction of meaningful $ECID$ distributions. All four of these procedures used the same set of checkpoints, $C = \{200, 100, 50, 25, 12.5\}$.

The five procedures were compared on the same 40 benchmark problems described in subsection 6.1[6]. Each SA procedure was run for 2 hours on each benchmark problem. Furthermore, to eliminate possible noise effects, each two-hour experiment was repeated a total of 15 times. The results presented here were obtained by averaging performance of these 15 runs of each procedure on each problem.

Fig. 5 depicts the performance of the five SA procedures on a typical benchmark problem. The first 15 minutes are not represented, as they correspond to the first 5 runs when the cutoff and restart criteria have not yet been activated.

---
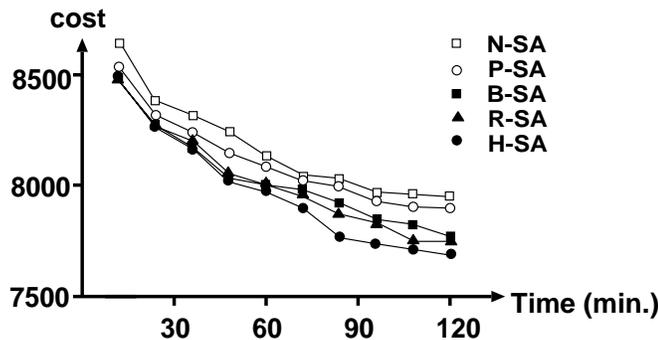
[6]At the present time, only a subset of the problems in each of the 8 problem sets have been completed. Complete results will be presented in the final version of the paper.

Fig. 5 Improvement of the best solution over time.

The figure shows that throughout its run, N-SA was dominated by the other four procedures. It also indicates that both the cutoff criterion and the restart criteria contributed to this performance improvement. Among the three restart criteria, H-SA appears to perform best. Figure 5 further suggests that the restart criterion in H-SA improves performance through the entire run, as the gap between H-SA and N-SA widens over time. These observations are confirmed by results obtained on the 8 problem sets of the study, as depicted in Figure 6. Fig. 6(a) shows the average cost reductions yielded by P-SA, B-SA, R-SA and H-SA over N-SA at the end of the two-hour runs. Figure 6(b) gives the average reduction in the CPU time required by each of these four procedures to find a solution of equal or better quality than the best solution found by N-SA in two hours. It can be seen that H-SA requires between 30% and 70% less CPU time than N-SA.

(a) Cost reduction (computation time: fixed)   (b) Speedup (cost: fixed)
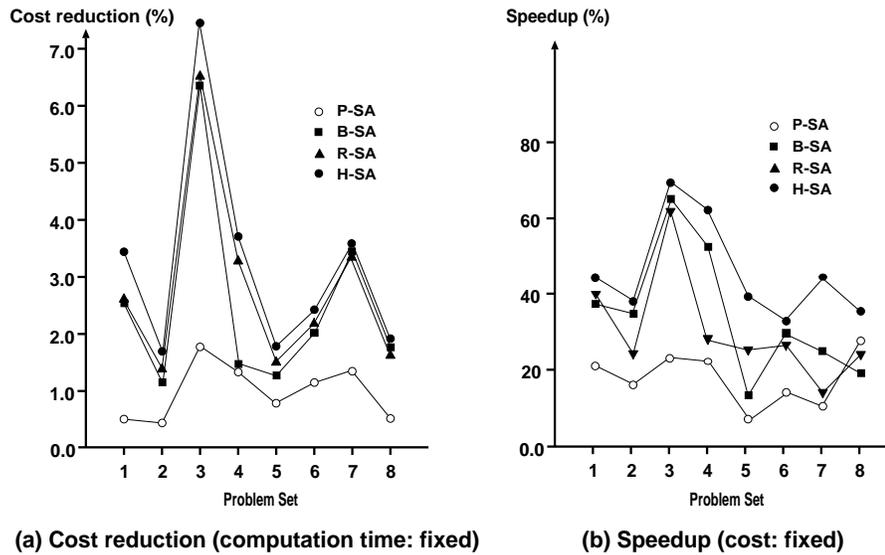
Fig. 6 Empirical comparison.

A finer analysis indicates that performance improvements produced by our cutoff and restart criteria increase as one requires higher quality solutions. Figure 7, compares the average CPU time of each of the five procedures as the required quality of solutions is increased. While all five procedures take about as long to find

a solution with cost below 9000 or 8800, the time required to find
a solution below 8500 varies significantly (e.g. H-SA can find such
a solution in 3500 seconds while N-SA requires close to 10,000 sec-
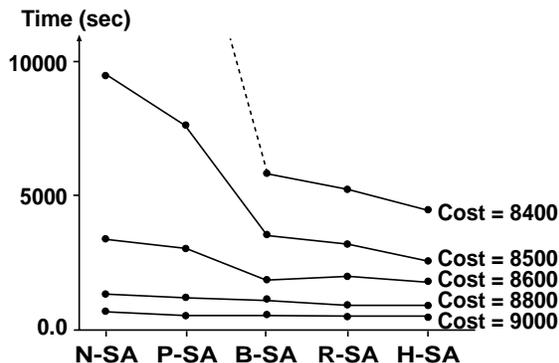onds).

Fig. 7 Speedups as a function of required solution quality.

As already indicated in Section 5, the difference in performance
between B-SA, R-SA and H-SA suggests that a deterministic use
of $ECID$ distributions to decide where to restart search can be
tricky, as these distributions may not be accurate, especially when
only a small number of runs has been completed. By injecting non-
determinism in the restart criterion, R-SA and H-SA ensure that
the procedure will not always restart from the same point. The
procedure is forced to sample a wider area and in the process gets
a chance to refine $ECID$ distributions. From this point of view,
B-SA is a procedure that places more emphasis on using existing
knowledge of the search space than acquiring new one, while R-SA
places more emphasis on learning and less on exploiting already
acquired information. H-SA appears to provide the best balance
between these two requirements.

Finally, it should be obvious that the CPU time and memory
overheads of our cutoff and restart criteria are very moderate. All
in all, in our experiments, the CPU time required to learn $ECID$
distributions and apply the cutoff and restart criteria was well under
1% of total CPU time.

6.2   The Vehicle Routing Problem With Time Windows

The Vehicle Routing Problem with Time Windows (VRTW) is an
extension of the classical vehicle routing problem (VRP) [2, 8, 13].
In the VRPTW, a set of capacitated vehicles are to be routed from a

central depot to pickup goods at different customer locations. Each customer location has a time window with an earliest release time and a latest delivery time. The vehicles are required to service every customer after their release times and before the latest delivery times. A service time is incurred at each customer location visited by the vehicle. Vehicles that reach a customer before the earliest release time have to wait until the release time is realized. Vehicles that reach the customer after the latest delivery time are considered to be tardy. In [27], it is shown that finding a feasible solution to the Traveling Salesman Problem with Time Windows (TSPTW) is an NP-complete problem. As a result, it can be seen that finding a feasible solution to the VRPTW with a fixed number of vehicles is also NP-complete. Practical applications of the VRPTW are many (e.g. bank deliveries, postal deliveries, industrial refuse collection, school bus routing and scheduling, etc.) and research on this problem has resulted in a number of publications (e.g. [28], [13], [21], [4], [29], [30]).

In the procedure considered in this study, an initial solution is generated using a Genetic Algorithm [29]. The solution is then improved using a SA procedure where neighborhood solutions are generated with λ-interchange operators similar to the ones introduced in [19] for the capacitated clustering problem. λ-interchange operators rely on the interchange of customers between pairs of routes. Given a solution $S = \{R_1, \cdots, R_p, \cdots, R_q, \cdots, R_k\}$, where each route R_i represents the set of customers serviced by $vehicle_i$, a λ-interchange between a pair of routes R_p and R_q is the replacement of a subset of customers $S_1 \subseteq R_p$ ($\mid S_1 \mid \leq \lambda$) by another subset $S_2 \subseteq R_q$ ($\mid S_2 \mid \leq \lambda$) to produce a neighboring solution $S' = \{R_1, \cdots, R'_p, \cdots, R'_q, \cdots, R_k\}$, where $R'_p = (R_p \setminus S_1) \cup S_2$ and $R'_q = (R_q \setminus S_2) \cup S_1$. Given a value of λ, a cycle in the SA procedure (equivalent of the "for i=1,N" loop in Fig. 1) involves the systematic examination of all neighboring solutions one can reach through λ-interchange. Given a solution $S = \{R_1, \cdots, R_p, \cdots, R_q, \cdots, R_k\}$, our procedure systematically considers each pair of routes:

$$(R_1, R_2), (R_1, R_3), \cdots, (R_1, R_k), (R_2, R_3), \cdots, (R_2, R_k), \cdots, (R_{k-1}, R_k) \tag{1}$$

and considers all possible λ-interchanges between these routes. The result is a search cycle that examines a total of $\frac{k \times (k-1)}{2}$ pairs of routes. To avoid too long of a cycle, the SA procedure considered

in this study restricts itself to the exchange of at most two customers between routes ($\lambda = 2$). Specifically, given a pair of routes R_p and R_q, neighboring solutions are systematically generated by applying each one of the following operators: (0,1), (1,0), (1,1), (0,2), (2,0), (2,1), (1,2) and (2,2), where (0,1) denotes the transfer of one customer from route q to route p, (0,2) the transfer of two customers, (1,1) the exchange of one customer between route p and q, etc. Given a pair of routes and an operator, neighboring solutions are generated by visiting customers along each route in sequential order. For each λ-interchange, the cost of the newly obtained solution is computed by updating the costs of routes R_p and R_q. Based on these costs and the current temperature, the SA procedure determines whether or not to accept the transition. The SA procedure considered in this study allows for transitions to solutions that violate vehicle capacity (overloading violation) and/or tardiness constraints. This is accounted for using costs penalties proportional to the constraint violations. The cost associated with a given vehicle route R_k is of the form:

$$D_k + \phi W_k + \eta O_k + \kappa T_k \qquad (2)$$

where D_k is the total distance traveled by vehicle k, W_k is the total waiting time on route k, O_k is the total overload for vehicle route k and T_k is the total tardiness on vehicle route k. In the experiments reported below, the weight factor ϕ was set to one percent of the total distance D_k. The penalty factor ϕ was set to ten percent of D_k and κ to one percent of D_k, thereby creating penalties that are proportional to the total distance traveled by the vehicle. The result is a SA procedure that oscillates between feasible and infeasible parts of the solution space. The amplitude of the oscillation is determined by the current value of the temperature parameter. As temperature decreases, the amplitude is reduced with search concentrating mainly on the feasible region. The final solution reported by the procedure is the best feasible solution obtained over the entire search.

6.2.1 Empirical Evaluation of Cutoff and Restart Criteria

Performance of this SA procedure both with and without our cutoff and restart criteria was studied on an initial set of VRPTW benchmarks. The 16 problems, each consisting of 100 customers, were

obtained from the R1 and RC1 data sets introduced in [28]. These problems have been used since then by a number of researchers to evaluate the performance of their procedures. The R1 data set consists of 100 uniformly distributed customers and the RC1 set consists of 100 customers that are both clustered and uniformly distributed. Each problem requires a minimum number of vehicles ranging between 10 and 20. The experiments reported in this subsection, were conducted using a set of checkpoint temperatures $C = 150, 100, 50, 25, 12.5$ [7].

On each problem, five initial SA runs were executed to obtain meaningful ECID distributions (i.e. train the system).

[7]In general, optimum sets of checkpoint temperatures are expected to vary from one domain to another. Experiments reported in this section are preliminary and it is expected that better results could be obtained by further fine tuning checkpoint temperatures and other parameters.

Results obtained with different variations of our SA procedure

for each of the sixteen problems are presented in Table 1 and 2, where they are compared with the performance of other procedures described in the literature [28, 5, 21, 29, 23]. On each problem, each variation of our SA procedure ($SA - N$, $SA - B$, $SA - R$, and $SA - H$) was applied for a total of ten runs, not including the five runs to obtain initial ECID distributions. CPU time was obtained on an Intel486/100 Mhz machine running C.

While performance of our procedures could benefit from additional fine tuning (e.g. checkpoint temperatures), the results clearly that:

1. Our basic SA procedure (i.e. procedure without cutoff and restart criteria), denoted $SA - N$, produces results that are consistently within 5 to 10 percent of the optimal solution or best known solution on data set R1-8. The results on data set RC1-8 are generally within 12 percent or so, with the exception of RC3 and RC4.

2. Performance of the procedure consistently improves with the addition of our cutoff and restart criteria, reducing CPU time and often producing better solutions. In addition, the resulting SA-H procedure appears competitive, when considering its CPU requirements.

Overall, the SA procedure with cutoff and randomized restart (SA-R) and the SA procedure with cutoff and hybrid restart (SA-H) perform consistently better than the other three SA methods. Between SA-R and SA-H, the SA-H method is more consistent in the amount of time required to explore its neighborhood and obtains solutions that are usually better than all of the other SA methods. In other words, while more preliminary, these results seem to confirm those observed with just-in-time job shop scheduling problems, as reported earlier.

7 Summary

In summary, we have developed a mechanism that learns to recognize (un)promising SA runs by refining "Expected Cost Improvement Distributions" ($ECID$s) over multiple SA runs, and have developed search cutoff and restart criteria that exploit these distributions. These mechanisms are domain independent and have been validated on two domains: the just-in-time job shop scheduling domain with tardiness and inventory costs and the vehicle routing problem with time windows. Experimental results show that in both domains these mechanisms can dramatically reduce the computational requirements of competitive SA procedures. Experiments presented in this paper further indicate that the closer one seeks to be to the optimum, the larger the speedups.

8 Acknowledgements

References

[1] Beyer, W.H. "CRC Standard Mathematical Tables, 28th Edition," CRC Press, Inc., Boca Raton, Florida, 1987.

[2] Bodin, L., B. L. Golden, A. A. Assad and M. Ball "The State of the Art in the Routing and Scheduling of Vehicles and Crews" *Computers and Operations Research* Vol. 10 no. 2, 1983.

[3] Cerny, V., "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *J. Opt. Theory Appl.,* Vol. 45, pp. 41–51, 1985.

[4] Chiang, W. C. and R. Russell "Simulated Annealing Metaheuristics for the Vehicle Routing Problem with Time Windows" Working Paper, Department of Quantitative Methods University of Tulsa, Tulsa, OK 74104. 1993.

[5] Desrochers, M., J. Desrosiers and M. M. Solomon "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows" newblock *Operations Research* Vol. 40 no. 2, 1992.

[6] Garey, M. R. and Johnson, D. S. "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman and Co., 1979

[7] Glover, F. and Laguna M. "Tabu Search," Chapter in *Modern Heuristic Techniques for Combinatorial Problems*, June 1992.

[8] Golden B. L. and A. A. Assad (eds.) "Vehicle Routing: Methods and Studies," North Holland, Amsterdam 1988.

[9] Graves, S. C. "A Review of Production Scheduling," *Operations Research* Vol. 29 no. 4, pp. 646–675, 1981.

[10] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part I, Graph Partitioning" *Operations Research* Vol. 37 no. 6, pp. 865–892, 1989.

[11] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part II, Graph Coloring and Number Partitioning" *Operations Research* Vol. 39 no. 3, pp. 378–406, 1991.

[12] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science,* Vol. 220, pp. 671–680, 1983.

[13] Laporte, G. "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms," *European Journal of Operational Research,* Vol. 59, pp. 345–358, 1992.

[14] Matsuo H., C.J. Suh, and R.S. Sullivan "A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem" Technical Report, "Department of Management, The University of Texas at Austin Austin, TX, Working Paper, 1988. 1992.

[15] Morton, T.E. and Pentico, D.W. "Heuristic Scheduling Systems," *Wiley Series in Engineering and Technology Management,* 1993

[16] Morton, T.E. "SCHED-STAR: A Price-Based Shop Scheduling Module," *Journal of Manufacturing and Operations Management,* pp. 131–181, 1988.

[17] Nakakuki,Y. and Sadeh,N. "Increasing the Efficiency of Simulated Annealing Search by Learning to Recognize (Un)Promising Runs," *Proceedings of the Twelfth National Conference on Artificial Intelligence,*, pp. 1316–1322, 1994.

[18] Osman,I.H., and Potts, C.N., "Simulated Annealing for Permutation Flow-Shop Scheduling," *OMEGA Int. J. of Mgmt Sci.,* Vol. 17, pp. 551–557, 1989.

[19] Osman, I.H. "Meta-Strategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem," *Annals of Operations Research,* Vol. 41, pp. 421–451, 1993.

[20] Palay, A. "Searching With Probabilities" PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, July 1983.

[21] Potvin, J. Y. and S. Bengio. "A Genetic Approach to the Vehicle Routing Problem with Time Windows" Technical Report, CRT-953, Centre de Recherche sur les Transports, Universite de Montreal, Canada, 1993.

[22] Potvin, J. Y. and J. M. Rousseau "A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows" *European Journal of Operational Research*, Vol. 66, pp. 331-340, 1993.

[23] Rochat, Y. and E. D. Taillard "Probabilisitic Diversification and Intensification in Local Search for Vehicle Routing " *European Journal of Operational Research*, Vol. 66, pp. 331-340, 1995.

[24] Sadeh, N. "Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling" PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, March 1991.

[25] Sadeh, N. "Micro-Opportunistic Scheduling: The Micro-Boss Factory scheduler" Chapter in *Intelligent Scheduling*, Zweben and Fox (eds), Morgan Kaufmann Publishers, 1994.

[26] Sadeh, N. and Nakakuki Y. "Focused Simulated Annealing Search: An Application to Job Shop Scheduling" To appear in *Annals of Operations Research*, Special issue on Metaheuristics in Combinatorial Optimization, Also available as technical report CMU-RI-TR-94-30 at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.

[27] Savelsbergh M. W. P. "Local Search for Routing Problems with Time Windows," *Annals of Operations Research*, Vol. 4, pp. 285–305, 1985.

[28] Solomon, M. M. "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints." *Operations Research*, Vol. 35, no. 2 pp. 254–265, 1987.

[29] Thangiah, S. R. "Vehicle Routing with Time Windows using Genetic Algorithms." Chapter in *Application Handbook of Genetic Algorithms: New Frontiers*, CRC Press, 1994.

[30] Thangiah, S. R. , I. H. Osman, T. Sun "Metaheuristics for Vehicle Routing Problems with Time Windows." Technical Report *Artificial Intelligence and Robotics Laboratory*, Computer Science Department, Slippery Rock University, Slippery Rock, PA 16057, 1994.

[31] Van Laarhoven, P.J., Aarts, E.H.L., and J.K. Lenstra "Job Shop Scheduling by Simulated Annealing," *Operations Research,* Vol. 40, No. 1, pp. 113–125, 1992.

[32] Vepsalainen,A.P.J. and Morton, T.E. "Priority Rules for Job Shops with Weighted Tardiness Costs," *Management Science,* Vol. 33, No. 8, pp. 1035–1047, 1987.

[33] Watkins, C. J. C. M., "Learning with Delayed Rewards" PhD thesis, Cambridge University, Psychology Department, 1989.

[34] Wefald, E.H. and Russel, S.J. "Adaptive Learning of Decision-Theoretic Search Control Knowledge" *Sixth International Workshop on Machine Learning*, Ithaca, NY, 1989.

[35] Zweben, M., Davis E., Daun B., Deale M., "Rescheduling with Iterative Repair" Technical Report FIA-92-15, NASA Ames Research Center, Artificial Intelligence Research Branch, Moffett Field, CA 94025 April, 1992.