Learning Annealing Schedules for Channel Routing

Todd W. Neller and David C. Hettlinger

Abstract—**The choice of a good annealing schedule is necessary for good performance of simulated annealing for channel routing and other combinatorial optimization problems. In this paper, we propose a new means of controlling annealing temperatures by posing the simulated annealing task as an optimal control problem and applying the techniques of reinforcement learning. Although many means of automating control of annealing temperatures have been proposed, this technique requires no specific knowledge of the problem and provides a natural means of expressing time versus quality tradeoffs.**

Index Terms—**learning systems, optimization methods, routing, simulated annealing.**

I. Introduction

SIMULATED annealing (SA) [6] is a technique for combinatorial optimization that is inspired by analogy to the cooling and freezing of liquids. A slowly cooled metal reaches a more orderly, lower-energy state than one that is cooled rapidly or "quenched". With SA, search in a state space varies with a "temperature" parameter from a freely random walk to a stochastic local optimization. In this paper, we focus on the how one can automatically learn control of the temperature parameter. For further background reading on SA, we recommend [6] and [9]. The basic SA algorithm can be described as follows:

Pick an initial state s.
While cooling temperature T according to a given schedule:

1. Generate a next state s'.
2. Compute $\Delta E = E(s') - E(s)$, the change in energy.
3. If $\Delta E < 0$, accept the next state ($s = s'$).
4. Otherwise, accept the next state with probability $e^{-\Delta E/kT}$, where k is Boltzmann's constant. [1]

Return the state s^* that minimized E.

To apply SA, the user must make four design decisions: (1) the state representation, (2) the next state generation function, (3) the energy (or objective) function, and (4) the *cooling* or *annealing schedule*. The successful application of SA often depends on good choices for each of these. Although (1)-(3) present interesting challenges of their own, we here focus on (4), the annealing schedule. In the more general case where the temperature is controlled dynamically, we refer to this as annealing control.

In their paper [6], Kirkpatrick et al. note the importance of the annealing schedule. They define a "specific heat" function for their circuit layout problem, noting that it "…can be useful in practice as a means for determining the temperature ranges in which the important rearrangements in the design are occurring, where slower cooling [will] be helpful." Indeed, the paper goes on to show that different combinatorial optimization problems call for different annealing schedules that change the rate of cooling in specific temperature ranges.

Many annealing schedules are still chosen through trial and error, but the space of possible annealing schedules is vast, and a user's search is generally limited and tedious. It would be preferable to automate annealing control and focus attention on the other dimensions of design. To this end, researchers have proposed numerous ways to automate annealing control.

For example, Otten and van Ginneken [8] developed a theory for annealing in the framework of Markov chains. Unfortunately, the application of their theory requires the user to know the size of the state space and the number of global optima. While the state space size is difficult to estimate for most problems, one rarely expects to have knowledge of the number of global optima. Techniques such as these require too much knowledge of the state space.

Many means of controlling annealing (e.g. [1], [2], [5], [7]) provide ways to speed cooling and trade off the quality of the result for computational time. This tradeoff of time versus quality of result is fundamental to the application of SA. However, no method to date deals explicitly with the time versus quality tradeoff.

In this paper, we introduce a means of controlling annealing that requires no knowledge of the state space, and allows the user to be explicit about time versus quality tradeoffs. This is accomplished by first posing SA as an optimal control problem, and then applying the techniques of reinforcement learning [11] to approximate a solution to the Bellman optimality equations. In effect, the algorithm "learns" annealing control through annealing experience. Our initial

T. W. Neller is with the Department of Computer Science, Gettysburg College, Gettysburg, PA 17325 USA (phone: 717-337-6643; fax: 717-337-6638; e-mail: tneller@gettysburg.edu).

D. C. Hettlinger is a student of the Department of Computer Science, Gettysburg College (e-mail: hettda01@gettysburg.edu) and was supported by a Gettysburg College Research Grant.

[1] In practice, Boltzmann's constant is often omitted from the implementation.

results demonstrate the practicality of this technique in the context of VLSI channel routing.

In section II, we pose SA as an optimal control problem. Next, we discuss possible designs for the annealing agent in section III. The empirical study of section IV compares the performance of one annealing agent's control to two other manually-tuned annealing controllers. Finally, we seek to place the significance of the results in a broader perspective.

II. Optimal Annealing

At a metalevel, we view simulated annealing (SA) design itself as an optimal control problem where the annealing controller receives a control input signal describing the SA process and transmits a control output signal prescribing how the process should proceed. The performance index or objective function to be optimized *for the controller* is the *expected utility of the SA result* (i.e. the returned state). Although there are other design decisions that affect the performance of SA, we only consider the choice of this *annealing controller.* Our goal is to seek an annealing schedule/controller that varies temperature in such a way as to maximize the expected utility of the SA result.

The utility of the SA result to the user may in actuality be very complex and impossible to predict. However, we note that two central considerations are (1) the utility of the quality of the result of SA (e.g. in terms of conserved space, money, etc.), and (2) the utility of the time it takes to compute such a result. Put simply, *each iteration of simulated annealing offers a potential gain of quality with a definite loss of time*.

The measure of result quality, called the *intrinsic utility function* [10], should be considered distinct from the state energy (a.k.a. objective) function of SA. While intrinsic utility may be a function of energy, they serve two different purposes. A result of SA may not have any intrinsic utility until its energy crosses a certain threshold. The energy function, by contrast, can and should provide guidance in searching the state space. We denote the energy and intrinsic utility of the search state s to be $E(s)$ and $U_I(s)$ respectively. The time cost function is denoted $TC(t)$ where t is elapsed SA time. Let s_t be the current search state of a SA process at time t. Let $s_t^* = \arg\min_{s_{t'}, 0 \le t' \le t} E(s_{t'})$ denote the minimal energy state s_t^* visited up to time t. Then the *net utility* $U(s_t^*, t)$ of a SA process at time t is $U_I(s_t^*) - TC(t)$. If we compare the net utility of a SA process at two different times t_1 and t_2, then the utility of the SA process from t_1 to t_2 is $U(s_{t_2}^*, t_2) - U(s_{t_1}^*, t_1)$.

The annealing agent may choose between iterations how to proceed with annealing. We call each of the annealing agent control signals an *annealing action.* An annealing action may dictate a temperature control policy and the duration (i.e. number of iterations) of that policy. Another important annealing action is to *terminate* the SA process. An annealing agent monitors the state of the SA process and directs its progress or termination.

Optimal control of annealing would then be an agent producing a sequence of annealing actions that maximize expected utility. Thus, learning optimal control of annealing entails some form of search among mappings from annealing states to annealing actions for one that approximates optimal control. This invites us to employ policy evaluation and improvement techniques of reinforcement learning [11].

III. The Annealing Agent

The annealing agent is essentially an adaptive mapping from the annealing states to annealing actions. Over time, the agent seeks to maximize its expected reward. We consider possibilities for each of these in turn.

A. States

One could represent the SA process as a Markov decision procedure (MDP) based on each search state s_t. To condition our decisions on actual states of the search space requires knowledge of the state space (i.e. domain-specific knowledge) that would make our technique less general. We instead opt, as with other automatic annealing controllers, to approximate optimal control by use of an approximating MDP. In other words, we base our MDP on an abstraction of the current state of the SA process itself and consider only the most important domain-independent features such as time t, temperature T, or energy $E(s_t)$. Many other measures such as an approximation of specific heat at a given temperature or various Markov chain statistics could also summarize important features of the SA process.

B. Actions

Between SA iterations, the annealing agent may choose whether and how to proceed with annealing. This includes both an annealing policy and the duration of that policy. For example, an agent may decide to perform 1000 iterations of annealing at a specific fixed temperature. It may instead decide to perform 100 iterations with a fixed temperature decay rate. The agent may also decide to raise the temperature for a period[2], or may decide to terminate the SA process.

While it is possible to have the annealing agent choose a temperature for each iteration of simulated annealing, this is unnecessary and introduces significant computational overhead. This is decision-making at a fine-grained extreme where all annealing schedules can be expressed. At the same time, we have a vast space of potential SA state-action mappings to consider, posing problems for learning. As we make our decision-making more coarse-grained, we reduce both the difficulty of learning and the quality of the approximation.

C. Rewards

A reinforcement learning agent seeks to maximize its expected reward over time. This reward is received iteratively after each action. After choosing an action a in state s at time t_1, the agent receives at time t_2 a new state s' and an immediate reward r.

[2] Allowing what is called *simulated tempering*

The annealing agent's reward simply reflects its gain in utility: $r = U(s^*_{t_2}, t_2) - U(s^*_{t_1}, t_1)$. In reinforcement learning, the agent will thus seek to maximize the expected utility of SA over time. How well it is able to do so depends on the choice of state representation, actions, and algorithms applied to learning the optimal behavior of the given MDP.

IV. Experimental Results

We now compare the performance of the optimal annealing schedule found by our annealing agent with two manually-tuned schedules: a geometric schedule and Boyan's modified Lam schedule. All three schedules were tested with Deutsch' s difficult example (DDE) [4], a challenging, well-known benchmark problem. We represent channel routing state and energy according to [13] with an assumption of restricted doglegging. Initially each net (i.e. set of connected pins) is assigned to its own horizontal track or *group*. Next states are generated by randomly reassigning portions of these nets to different groups. All SA trials in our experiments thus share the same initial state and search stochastically for a minimum energy state.

For our measure of net utility $U(s,t)$, we must define $U_I(s)$ and $TC(t)$. For simplicity, we let $U_I(s) = -E(s)$, i.e. the number of horizontal tracks negated, and $TC(t) = .0005 * iter(t)$, where $iter(t)$ is the number of iterations completed at time t. This latter choice allowed us to compare results from machines with different CPU clock speeds. Note that one is not limited to such simple choices in our framework.

A. Geometric Schedule

The first annealing schedule we experimented with was the geometric schedule where the temperature is multiplied by a fixed decay rate after each iteration. Thus the schedule is an exponential decay to temperature 0 specified by the initial temperature, the decay rate, and the number of iterations. This schedule apportions its temperatures evenly across orders of magnitude. Without knowledge of important temperature regions, this is a reasonable naïve schedule.

A good choice for an initial temperature can be computed by sampling initial states and computing the standard deviation σ of the ΔE's. Then, according to [5] and [12], the initial temperature $T_0 = 20\sigma$ allows acceptance of almost all ΔE one encounters in the first iterations of annealing. Since all of these tests are performed with the same initial state, we sampled random walks from the initial state before each run. A typical T_0 value was 60.

Beginning with 10000 iterations and a decay rate of .95, we manually tuned these parameters using an experimental technique best described as direction-set optimization without direction updates: vary and optimize each parameter in turn. Since this is a stochastic function, we performed 50 trials for each parameter setting. After some initial experimentation, it was determined that 7000 iterations and a decay rate of .75 were good choices. This is commonly referred to as *simulated quenching* since the temperature drops quickly. For Figure 1, we fixed the number of iterations at 7000 and varied decay, and ran 50 trials for each decay rate. All figures show means and 90% confidence intervals from bootstrapping with 10000 resamples.

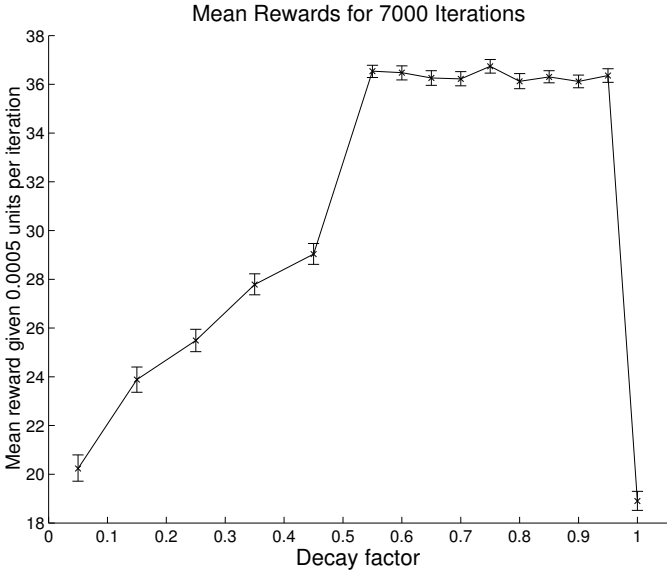

Fig 1. Geometric schedule net utility varying decay rate.

From this data, we observe a plateau indicating that there is little difference in the quality of decay rates from .55 through .95. The best of these was .75.

For Figure 2, we varied the maximum number of iterations for the decay rates .75 and .95, and ran at least 350 trials for each decay and number of iterations.

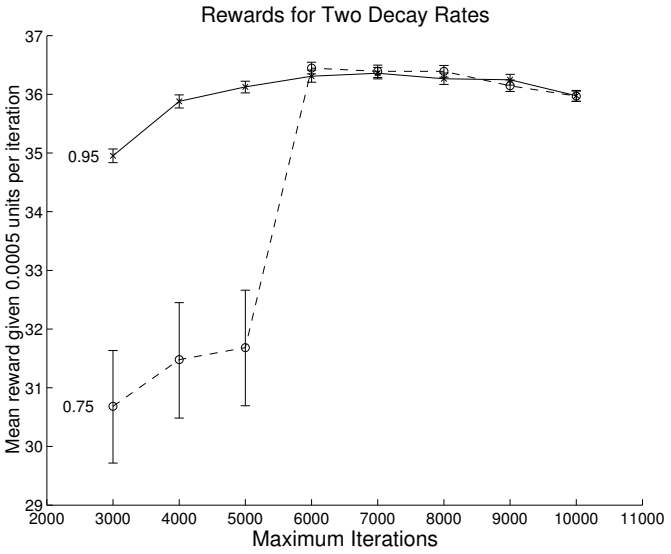

Fig. 2. Geometric schedule net utility varying maximum iterations.

From the data of Figure 2, we observe that both decay rates perform best from 6000 to 8000 iterations, peaking slightly at 6000 iterations. The performance difference for the two decay rates in this range is statistically insignificant.

Using a decay rate of .75 with 6000 iterations for 500 trials, the mean net utility of SA was 35.64 with a 90% confidence interval from 35.55 to 35.70.

B. Modified Lam Schedule

The Modified Lam (MLam) schedule described in Boyan's thesis [2] dynamically adjusts the temperature in order to track with a target acceptance ratio function. This ratio starts at 1.0 and decreases to .44 after 15% of the iterations and holds at .44 until 65% of the iterations. Then it drops exponentially to 0.

The Boyan thesis experiments used 500,000 maximum iterations but this would guarantee negative reward values given our cost per iteration. Thus, we varied the number of iterations logarithmically and gave each value 50 annealing runs. Figure 3 demonstrates that the optimal number of annealing iterations peaks near 10,000 iterations.

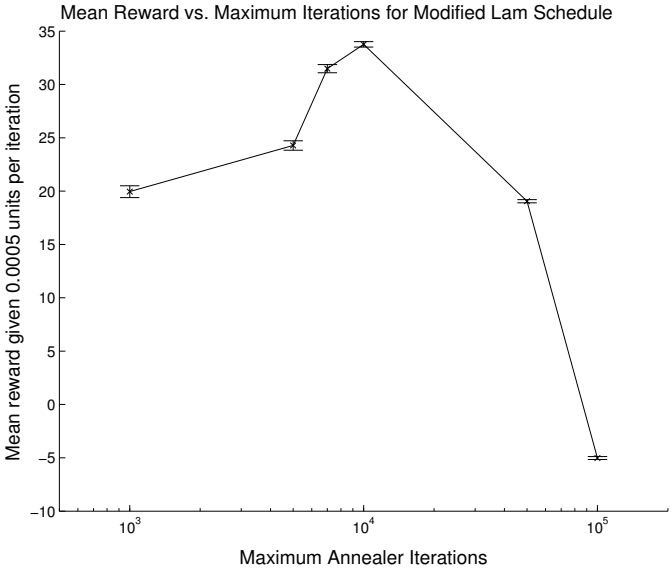

Fig 3. MLam schedule net utility varying iterations.

Using the MLam adaptive schedule for 10000 iterations for 500 trials, the mean net utility of SA was 33.18 with a 90% confidence interval from 33.09 to 33.27.

C. LSTD Annealing Agent Schedule

The LSTD Annealing Agent is based on Boyan's least-squares temporal-difference (LSTD(λ)) learning algorithm of [3] with $\lambda = .5$. One of LSTD learning's strengths is that it makes relatively efficient use of its learning experiences. This strength is particularly important for our application, where gaining experience is computationally expensive.

The agent has two types of actions: choose a decay rate for the next 1000 iterations, or terminate annealing. Decay rates were limited to the range [.9, .9999]. The agent was ε-greedy with $\varepsilon = .1$. That is, the agent took a random action 10% of the time and pursued what it had learned to be the best action 90% of the time. This is a simple means of balancing exploration versus exploitation. The best "greedy" action was chosen by first querying its internal state-action utility estimation function given the current state and each of four decay rates distributed logarithmically (i.e. {.9, .99, .999, .9999}). If the best state-action estimation is negative, the agent believes it has reached a point of diminishing returns, and the termination action is chosen. When taking random actions, termination is chosen 20% of the time, with a random logarithmically distributed decay rate chosen 80% of the time.

The state of the SA process was represented simply as the current temperature and iteration number. Thus, the agent learns to map temperatures and iterations to decay rates/termination. The state-action utility estimation, or *Q*-function, is implemented as a tile-coding memory [11] with 5 tilings. Each tiling had 5 partitions each for temperatures, iterations, and decay rates. Temperatures and decay rates were partitioned logarithmically.

Learning consisted of running the agent for over 4000 SA runs. Periodically, the state of the tile-coding memory was logged. Afterwards, we ran 500 trials with the agent acting 100% greedy at each stage of its learning, thus showing the progression of the agent's performance. Results are shown in Figure 4.

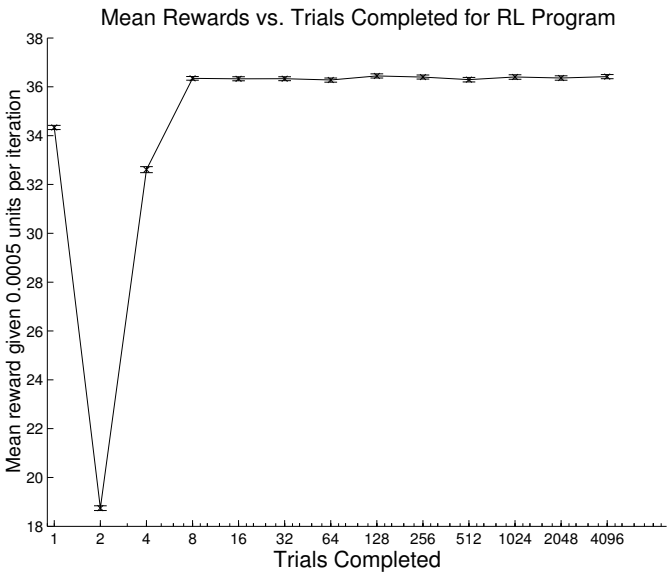

Fig. 4. LSTD agent schedule net utility as learning progresses.

The initial run of annealing happens to largely quench and gives an early bias towards this behavior. Recall that the agent issues a random annealing action 10% of the time. After only 8 runs of SA that explore a variety of schedules, the LSTD agent prefers a quenching policy. At that point the agent prefers to quench maximally with a decay rate of .9 for 8000 iterations. After 64 runs, it has refined its policy so as to maximally quench for 6000 iterations. Performance varies only slightly across the remainder of the runs as the agent gains new experiences and modifies its utility estimates for various annealing schedules. After 4096 learning trials, the LSTD agent had learned an annealing schedule with a decay of .9 for 6000 iterations. Using this schedule for 500 trials, the mean net utility of SA was 36.42 with a 90% confidence interval from 36.33 to 36.50.

V. Discussion

The LSTD agent had the highest mean net utility of 36.42. The manually-tuned geometric schedule's performance was a close second with a mean net utility of 35.64. What is of particular interest is the fact that the LSTD agent learned

essentially the same qualitative behavior with a small fraction of the experimentation.

As we view this result critically, we note that this is only one learning trial. However, two other learning trials that terminated prematurely due to various technical difficulties all showed the same qualitative behavior. Perhaps the extreme behavior of optimal quenching may be easier to learn. Even so, it is very interesting that the agent managed to tune the duration of annealing so well so quickly. It should be noted that the agent's control actions could approximate a wide variety of optimal annealing schedules.

In comparing the LSTD agent with the MLam schedule, we see that it is clearly superior. The parameters of MLam's target acceptance rate made it unsuitable for quenching. While we trust it is, as reported, a very good schedule for a variety of combinatorial optimization problems run for hundreds of thousands of iterations, it does not seem to provide the best schedule when computational time costs are high.

These preliminary results show great hope for the automated learning of different annealing schedules for different classes of optimization problems under different assumptions of overall utility.

We believe that this approach also presents promise for the wider class of optimization methods that rely on various forms of *relaxation*. When an optimization process must gradually change its behavior over time, the question inevitably arises regarding how quickly the behavior should change. Often experimentation and tuning are left as arcane "black arts" and thus prone to misapplication. We hope that such tuning will increasingly be performed autonomously so that we can fully realize the potential of such techniques.

VI. Conclusions

The LSTD Annealing Agent was able to autonomously find an annealing schedule in less than an hour of computation that exceeded the performance of manually-tuned schedules from over 30 hours of experimental computation. As an initial result, this is very encouraging, but further experimentation is required to understand its performance for different problems and utility assumptions.

What is perhaps the most important contribution of this work is a new approach to the design of annealing schedules. Currently, practitioners will often perform tedious, ad hoc experiments, inevitably introducing human error and bias. By placing this burden of experimentation onto an untiring, principled, exact, and unbiased autonomous agent, we make simulated annealing that much easier to apply.

Further, most adaptive, autonomous annealing methods provide no explicit means of expressing the utility of time versus quality of result. To pose annealing as an optimal control problem where time and quality are explicitly placed in tension is, we believe, a better way to frame the problem.

References

[1] E. H. L. Aarts and P. J. M. van Laarhoven, "A new polynomial time cooling schedule," in *Proc. ICCAD-85: IEEE Int. Conf. on Computer Aided Design*, 1985, pp. 206–208.

[2] J. A. Boyan, "Learning Evaluation Functions for Global Optimization," Ph.D. thesis, Comp. Sci. Dept., Carnegie Mellon Univ., Pittsburgh, PA, August 1998, Appendix B. Available: Carnegie Mellon Tech. Report CMU-CS-98-152.

[3] J. A. Boyan, "Least-squares temporal difference learning," *Machine Learning*, vol. 49, nos. 2/3, pp. 233–246, Nov./Dec. 2002.

[4] D. N. Deutsch, "A 'Dogleg' channel router," in *Proc. of the 13th Design Automation Conference*, San Francisco, CA, 1976, pp. 425–433.

[5] M. D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An efficient general cooling schedule for simulated annealing," in *Proc. ICCAD-86: IEEE Int. Conf. on Computer Aided Design*, Santa Clara, CA, 1986, pp. 381–384.

[6] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 13 May 1983.

[7] J. Lam and J.-M. Delosme, "Performance of a new annealing schedule," in *Proc. of the 25th ACM/IEEE Design Automation Conference*, Anaheim, CA, 1988, pp. 306–311.

[8] R. H. J. M. Otten and L. P. P. P. van Ginneken, *The Annealing Algorithm*, Dordrecht: Kluwer Academic Publishers, 1989.

[9] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes in C: the art of scientific computing,* 2nd ed., Cambridge: Cambridge University Press, 1988, pp. 444–455.

[10] S. J. Russell and E. Wefald, *Do the Right Thing: studies in limited rationality*, Cambridge, MA: MIT Press, 1991.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: an introduction*, Cambridge, MA: MIT Press, 1998.

[12] S. R. White, "Concept of scale in simulated annealing," in *Proc. ICCD-84: IEEE Int. Conf. on Computer Design*, Port Chester, New York, 1984, pp. 646–651.

[13] D. F. Wong, H. W. Leong, and C. L. Liu, *Simulated Annealing for VLSI Design*, Dordrecht: Kluwer Academic Publishers, 1988.