D. R. Thompson and G. L. Bilbro, “Sample-sort simulated annealing,” *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, vol. 35, no. 3, pp. 625-632, June 2005.

Sample-Sort Simulated Annealing

Dale R. Thompson, *Senior Member, IEEE,* and Griff L. Bilbro, *Senior Member, IEEE*

***Abstract*—A simulated annealing (SA) algorithm called Sample-Sort that is artificially extended across an array of samplers is proposed. The sequence of temperatures for a serial SA algorithm is replaced with an array of samplers operating at static temperatures and the single stochastic sampler is replaced with a set of samplers. The set of samplers uses a biased generator to sample the same distribution of a serial SA algorithm to maintain the same convergence property. Sample-Sort was compared to SA by applying both to a set of global optimization problems and found to be comparable if the number of iterations per sampler was sufficient. If the evaluation phase dominates the computational requirements, Sample-Sort could take advantage of parallel processing.**

***Index Terms*—Optimization methods, parallel algorithm, simulated annealing (SA).**

I. Introduction

SIMULATED ANNEALING (SA) was first described by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller as a modified Monte Carlo integration method [1]. Then, Kirkpatrick, Gelatt, and Vecchi applied it to combinatorial optimization including the traveling salesman problem (TSP) and computer design [2]. SA is a randomized gradient descent algorithm that permits uphill moves with some probability so that it can escape local minima. It is commonly used to solve combinatorial problems [2], [3], but has also been used for functional optimization [4].

In this paper, all problems are assumed to be minimization problems. SA begins with an initial solution x_0 generated randomly or with a heuristic that has cost $f(x_0)$. Then, it performs the following steps for a number of iterations.

1) Given the current solution $x \in S$, generate a candidate solution $y \in S$ in the neighborhood of x, with the conditional probability $\gamma(y|x)$.
2) Evaluate the cost of the candidate solution $f(y)$.
3) Decide whether to accept y with probability

$$\min\left(1, \frac{\pi(y)}{\pi(x)}\right) \tag{1}$$

where $\pi(x)$ is the probability of being in state x and of the form $\pi(x) \propto \exp(-f(x)/T)$ and T is a parameter known as temperature.

The serial SA algorithm is shown in Fig. 1. The acceptance criteria in the algorithm of Fig. 1 is based on the Metropolis sampling procedure for a target distribution $\pi(x)$ [2]. The probability in (1) is implemented by the *if* statement that accepts y if it is less than x, or accepts a higher cost y with probability $\exp(-\Delta E/T)$, where $\Delta E = f(y) - f(x)$. The function random() returns a number in the range [0,1) with a uniform distribution. The parameter T begins at a high temperature and is slowly lowered. At higher values of T, the probability of accepting a higher cost solution is higher than at lower values of T. The control parameter T controls how sharply $\pi(x)$ is peaked at the minima of f [4].

```
x ← initial_solution
f(x) ← evaluate(x)
while (T > T_final)
  for i = 1 to iterations(T)
    y ← generate(x)
    f(y) ← evaluate(y)
    ΔE ← f(y) − f(x)
    /* decide */
    if[ΔE ≤ 0 OR exp(−ΔE/T) > random()]
      /* accept candidate solution */
      x ← y
      f(x) ← f(y)
    endif
  endfor
  T ← next_temp(T)
endwhile
```

Fig. 1. Serial SA algorithm.

The initial temperature, the number of iterations at each temperature, the method used to decrease the temperature, and the final temperature is called the *annealing schedule* [5], [6]. The quality of the final solution is dependent on the annealing schedule. If the temperature is lowered slowly, then the quality of the solution is better. One theoretical optimal cooling schedule by Hajek is shown in (2), where k is the iteration number and c is a problem specific constant [5]. The constant c is the depth of the maximum local minima that is not the global minimum. This gives SA the ability to climb out of the largest local minima

$$T_k = \frac{c}{\log(1+k)}. \tag{2}$$

The annealing schedule presented in (2) asymptotically approaches the global minimum as $k \to \infty$. In practice, (2) typically requires too much computational time. Therefore, several nonoptimal cooling schedules that work well in practice are commonly used. For example, the cooling schedule commonly is of the following form:

$$T \leftarrow \alpha T \tag{3}$$

Manuscript received June 29, 2004; revised January 5, 2005. This paper was recommended by Associate Editor Y. Pan.

D. R. Thompson is with the Computer Science and Computer Engineering Department, University of Arkansas, Fayetteville, AR 72701 USA (e-mail: d.r.thompson@ieee.org).

G. L. Bilbro is with the Electrical and Computer Engineering Department, North Carolina State University, Raleigh, NC 27695-7115 USA (e-mail: griff_bilbro@ncsu.edu).

Digital Object Identifier 10.1109/TSMCB.2005.843972

where α is called the cooling coefficient with $0.80 \leq \alpha < 1.0$ [7].

The SA algorithm proposed by Kirkpatrick *et al.* is serial in nature and requires a large amount of computational time [2]. Each generate, evaluate, and decide cycle of the serial SA is performed for several iterations at a constant temperature and then the temperature is lowered. In this paper, a new algorithm is presented that is based on an algorithm presented in [4] that has m *samplers* performing the generate, evaluate, and decide cycle each at a different temperature. The algorithm, called *Sample-Sort*, could be implemented on a computer with multiple processors to reduce the wall-clock time although it is not necessary and is not the focus of this paper. Since Sample-Sort does use multiple samplers it can be considered a parallel SA technique and shares the same theoretically treatment as parallel SA algorithms [8], [9]. The main contribution of this paper is a biased generator that artificially extends serial SA over an array of samplers. If the evaluation phase dominates the computational requirements, Sample-Sort could take advantage of parallel processing to reduce the wall-clock time.

Many parallel SA algorithms have been proposed because of the large amount of computational time of a serial SA [8]–[15]. Casotto, Romeo, and Sangiovanni-Vincentelli used a parallel SA for the placement of macrocells [10]. Witte, Chamberlain, and Franklin used speculative computation to create a parallel SA [11]. Future moves and evaluations were computed that may later be rejected by the decide cycle. Nabhan and Zomaya improve [11] by decreasing the amount of communications overhead [12]. Another way to make a parallel SA algorithm is to combine it with an algorithm that is naturally parallel. For example, Mahfoud and Goldberg developed the parallel recombinative simulated annealing (PRSA) algorithm that combines a genetic algorithm (GA) with SA [16]. It uses a Boltzmann trial for selection like SA. Kurbel, Schneider, and Singh use PRSA to solve the cell placement problem in VLSI [17]. Li and Jiang integrate SA, GA, and the chemotaxis algorithm and apply it to production scheduling, the traveling salesman problem, the training of artificial neural networks, and a set of common test functions [18], and Delport integrates SA with an evolutionary selection algorithm to create a parallel SA algorithm [19].

Lee and Lee provided one of the first classifications of parallel SAs [9]. Parallel SAs were classified as either acting on a single Markov chain or multiple Markov chains. Similar classifications are presented in [13] and [14]. Some of the classifications are the same, but others are different. These different classifications are merged and presented here to provide organization to the different types of parallel SAs. Onbaşoğlu and Özdamar classify parallel SAs into two general categories: application dependent and problem independent [13]. They focus on problem independent algorithms and develop five parallel SAs and compare them on an extensive set of functions. Under the classification presented in this paper, the algorithms are classified as being one of type 1, three of type 5, and one of type 2. They also present an improved type 2. As in [13], the focus of this paper is on algorithms that are problem independent.

Parallel SAs that are problem independent are classified into the following categories:

1) data parallel;
2) parallel moves;
3) speculative computation;
4) asynchronous parallel SA;
5) simultaneous periodically interacting;
6) Sample-Sort.

A data parallel algorithm simultaneously performs moves on chunks of data. It is used for problems that can be naturally divided and evaluated in pieces. The parallel moves algorithm follows a single Markov chain. In the classic form, a configuration is sent to multiple processors. Each processor performs a move and evaluate on the configuration and the best is chosen. In the speculative computation algorithm, several candidate future moves and evaluations are performed on a single Markov chain [11], [12]. Many of these calculations are rejected later depending upon the decide cycle. An asynchronous parallel SA consists of several simultaneous independent searches of different Markov chains. Then, the best solution is chosen from among the independent Markov chains. A simultaneous periodically interacting algorithm also consists of several simultaneous searches. But, occasionally the different Markov chains interact and exchange solutions. Finally, there is the Sample-Sort algorithm presented in this paper which is a special type of simultaneous periodically interacting algorithm. It will be discussed below.

A concern of parallel SAs is that they have the same convergence properties of a serial SA. The set of moves and computations done in parallel must be *serializable* to guarantee convergence [15]. The computation of the parallel moves should be equal to serial execution. A parallel algorithm that is not serializable may adversely affect the convergence [8]. In the data parallel algorithm, the problem can be naturally divided. Unfortunately, many problems cannot be naturally divided. Pao, Lam, and Fong use the concurrency control theory of databases to enforce the serializable property in their parallel SA [15]. In the parallel moves algorithm, a single Markov chain is followed and multiple moves are performed by different processors and the best is chosen. The synchronization can cause inefficiencies in the algorithm [9]. Lee and Lee [9] relax the constraint of following one Markov chain and obtained good results as do many others. Bevilacqua overcomes the synchronization inefficiency by dynamically switching between the serial SA and parallel SA depending upon the ratio between accepted and rejected moves [14]. At higher temperatures of SA, there are many accepted moves so synchronization overhead dominates. Therefore, it may be more efficient to use a serial SA. At lower temperatures, there are many rejections so parallel SA is more efficient. The speculative computation algorithm strictly enforces serialization by forcing it to follow a single Markov chain [11], [12]. An asynchronous parallel SA completely avoids the problem by using independent Markov chains. An *unmodified* simultaneous periodically interacting algorithm usually is not serializable, and, therefore, convergence may be adversely affected. In fact, if the simultaneous periodically interacting algorithm is performed with the processors having a *common* cooling schedule, then Azencott proved that it is asymptotically *less efficient* than performing independent searches as in the asynchronous parallel SA [8]. In other words,

it is better to perform independent noninteracting searches and choose the best answer from among the independent Markov chains. This can also be seen in the results of [9].

In this paper, we will focus on a modified simultaneous periodically interacting algorithm that we call *Sample-Sort*. There are multiple simultaneous periodically interacting searches being performed by different *samplers*. The algorithm that most resembles the new Sample-Sort algorithm is presented by Azencott [8]. In Azencott's algorithm, there are a fixed set of samplers each operating at different temperatures. Each sampler performs the generate, evaluate, and decide cycle at a different temperature. A solution that costs less is propagated from the higher temperature sampler to the neighboring sampler that is operating at a lower temperature. Therefore, the best solution at a given time is propagated to all samplers operating at a lower temperature. The performance of this algorithm is of the same order as a sequential SA at the lowest temperature T and the rate of convergence is quicker, but not computable according to them.

The Sample-Sort algorithm also has a fixed set of samplers each operating at different temperatures. It propagates a less-cost solution to other samplers, but does it *probabilistically* by permitting the samplers to exchange solutions with neighboring samplers. The samplers are lined up in a row and exchange solutions with samplers that are one or more hops away. The number of hops is called the neighborhood. Also, it adjusts the probability of accepting a higher cost solution dependent on the temperature of the neighboring sampler. Therefore, it overcomes the problem of maintaining serializable moves in an easier way than the previously proposed simultaneously periodically interacting parallel SA algorithms. Sample-Sort artificially extends the Markov chain across an array of samplers.

II. Biased Generators

Generating a candidate solution can be biased to a particular region of the solution space S by a heuristic method [4]. If the generate step is biased instead of uniform distribution, then the acceptance criteria must be modified by dividing $\pi(y)/\pi(x)$ by $\gamma(y|x)/\gamma(x|y)$ to preserve convergence [4], [20]–[22], as shown in (4)

$$\min\left(1, \frac{\gamma(x|y)\pi(y)}{\gamma(y|x)\pi(x)}\right). \tag{4}$$

III. Sample-Sort

The Sample-Sort algorithm is shown in Fig. 2. It consists of an array of m samplers, each operating at a different temperature T_k for $k = 1, 2, \cdots, m$. Assume T_k to be monotonically increasing. Also, Sample-Sort interacts with neighboring samplers N hops away. At each iteration of the algorithm, each sampler tests and possibly updates its current state twice: first with the state of the neighboring samplers as determined by the parameter N, then with a candidate solution generated uniformly in a neighborhood around its current state. Note that the sort starts at the sampler with the highest temperature and proceeds toward the sampler with the lowest temperature. This slightly decreased the time to converge compared to sorting from the lowest temperature to the highest temperature. When the state x_j of a neighboring sampler j is used as a candidate y for sampler i, the acceptance probability must account for the bias due to the generator, assuming the jth sampler has reached equilibrium. In (4), let $\gamma(y|x) = \pi_{T_j}(y)$ and $\gamma(x|y) = \pi_{T_j}(x)$. Let $\pi(x) = \pi_{T_i}(x)$ and $\pi(y) = \pi_{T_i}(y)$. Equation (4) prescribes that the present state of sampler j be accepted with probability

```
for x = 1 to n_iterations
  /* Metropolis Sort in temperatures */
  for i = m to 1
    for j = max(1, i − N) to min(n_s, i + N)
      accept y = x_j with probability a_1(y | x_i)
    end j
  end i
  /* Metropolis Sample in solution space */
  for i = m to 1
    accept y = x_i + δ, with probability a_2(y | x_i)
  end i
end x
```

Fig. 2. Sample-Sort SA algorithm.

$$a_1(x_j|x_i) = \min\left[1, \exp\left(-\left(f_{x_j} - f_{x_i}\right)\left(\frac{1}{T_{x_i}} - \frac{1}{T_{x_j}}\right)\right)\right]. \tag{5}$$

For the second kind of update, the generator obeys $\gamma(y|x_i) = \gamma(x_i|y)$ so that the acceptance probability of (4) reduces to the usual Metropolis criterion

$$a_2(y|x_i) = \min\left[1, \exp\left(-\frac{f_y - f_{x_i}}{T_i}\right)\right] \tag{6}$$

where y is a candidate generated uniformly in a region of specified size around the current state x_i. In Fig. 2, the parameter δ specifies the size of the region.

IV. Proof

The Sample-Sort algorithm is proved using a Markov chain for a neighborhood N equal to one in this section and can be extended to any neighborhood size. One of the original serial SA algorithm proofs requires that the transition from a state must be reversible to prove convergence [5]. This is one of the conditions that Sample-Sort enforces by modifying the acceptance criteria. To maintain convergence in Sample-Sort the probability of being in any of the samplers must be equal. In other words, the probability of being in any of the samplers must be $1/m$ for m samplers.

A. Neighborhood of One

The Markov chain of Sample-Sort with m samplers and a neighborhood $N = 1$ is shown in Fig. 3. The array of m samplers each operate at a different temperature T_k for $k = 1, 2, \cdots, m$. Assume T_k to be monotonically increasing. Let x be the initial state and y be the destination state of a valid transition. For a neighborhood $N = 1$, a valid transition is only with a neighboring sampler one hop away. Let $\pi(f, t) \propto \exp(-f/t)$ where f is the present value of the sampler and t is the fixed temperature of the sampler. The probability of being in state x and y is $\pi(x) = \pi(f_x, t_x)$ and

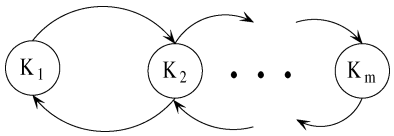

Fig. 3. Neighborhood equal to one.

$\pi(y) = \pi(f_y, t_x)$, respectively. The probability of going to state y given you are in state x is $\gamma(y|x) = \pi(f_y, t_y)$. The probability of going to state x given you are in state y is $\gamma(x|y) = \pi(f_x, t_y)$.

Using the previously defined values and (4), the probability of transition from sampler K_k to K_{k+1} is

$$\min\left(1, \frac{\pi(f_k, t_{k+1})\pi(f_{k+1}, t_k)}{\pi(f_{k+1}, t_{k+1})\pi(f_k, t_k)}\right). \tag{7}$$

Similarly, the probability of transition from K_{k+1} to K_k is

$$\min\left(1, \frac{\pi(f_{k+1}, t_k)\pi(f_k, t_{k+1})}{\pi(f_k, t_k)\pi(f_{k+1}, t_{k+1})}\right). \tag{8}$$

Let P_k and P_{k+1} be the probability of being in samplers K_k and K_{k+1}, respectively. At steady state, P_k times (7) is equal to P_{k+1} times (8)

$$P_k \times \min\left(1, \frac{\pi(f_k, t_{k+1})\pi(f_{k+1}, t_k)}{\pi(f_{k+1}, t_{k+1})\pi(f_k, t_k)}\right) = P_{k+1}$$
$$\times \min\left(1, \frac{\pi(f_{k+1}, t_k)\pi(f_k, t_{k+1})}{\pi(f_k, t_k)\pi(f_{k+1}, t_{k+1})}\right) \quad k = 1, 2, \cdots, m-1. \tag{9}$$

Therefore, the probability of being in sampler K_k is equal to the probability of being in K_{k+1}

$$P_k = P_{k+1}. \tag{10}$$

Also, the sum of the probabilities of being in each of the samplers must be equal to 1

$$\sum_{k=1}^{m} P_k = 1. \tag{11}$$

Therefore, the probability of being in any state is equal to

$$P_k = \frac{1}{m}. \tag{12}$$

V. Standard Global Optimization Problems

The proposed Sample-Sort algorithm was compared with the serial SA algorithm and shown to be approximately equivalent if a sufficient number of iterations per sampler was performed during the sampling phase of the algorithm. The algorithms were applied to a set of global optimization problems of varying degree of difficulty as suggested by Torn, Ali, and Viitanen [23]. Torn *et al.* ranked global optimization problems into different categories of complexity including unimodal, easy of type one and two (E1 and E2), moderate difficulty of type one and two (M1 and M2), and difficult of type one and two (D1 and D2) [23]. Torn *et al.* suggested that algorithms should be compared by applying them to problems taken from each of the categories [23]. Therefore, Sample-Sort was compared with SA over nine test problems from categories E1, E2, and M2. Each problem consists of an objective function f and a set of limits of the space to search for minima.

A. Function Branin (E1)

The function Branin is a two-dimensional (2-D) problem due to Branin with three global minima in the search space $-5 < x_1 < 15$, $0 < x_2 < 15$, at $(x_1, x_2) \approx (-3.143\,12.28)$, $(x_1, x_2) \approx (3.143, 2.275)$, and $(x_1, x_2) \approx (9.42, 2.475)$, all with value $f \approx 0.3979$ [24].

It is implemented by first defining the following variables:

$$\begin{aligned} a &= 1.0 \\ b &= \frac{5.1}{4\pi^2} \\ c &= \frac{5.0}{\pi} \\ d &= 6.0 \\ e &= 10.0 \\ g &= \frac{1}{8\pi}. \end{aligned} \tag{13}$$

Then, Branin is defined as (14)

$$f = a\left(x_2 - bx_1^2 + cx_1 - d\right)^2 + e(1.0 - g)\cos(x_1) + e. \tag{14}$$

B. Function GoldPrice (E1)

The function GoldPrice is a 2-D problem due to Goldstein and Price [24]. In the search space $-2 < x_1, x_2 < 2$, it has four local minima and one global minimum at $(x_1, x_2) = (0, -1)$ with a minimal value of $f = 3$

$$\begin{aligned} f = &\left[1+(x_1+x_2+1)^2\right. \\ &\left.\times\left(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2\right)\right] \\ &\times\left[30+(2x_1-3x_2)^2\right. \\ &\left.\times\left(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2\right)\right]. \end{aligned} \tag{15}$$

C. Function Shekel5 (E1)

The function Shekel5 is a four-dimensional (4-D) problem in the Shekel family and is the sum of five terms [24]. In the search space $0 < x_1, x_2, x_3, x_4 < 10$, it has a global minimum at $(x_1, x_2, x_3, x_4) \approx (4.0, 3.998, 3.991, 3.996)$, with value $f \approx -10.14$. It is implemented by first defining g as (16)

$$\begin{aligned} g(a_1, a_2, a_3, a_4, c) = &(x_1 - a_1)^2 + (x_2 - a_2)^2 \\ &+(x_3 - a_3)^2 + (x_4 - a_4)^2 + c. \end{aligned} \tag{16}$$

Then, Shekel5 is defined as (17)

$$f = -\left[\frac{1.0}{g(4.0,4.0,4.0,4.0,0.1)} + \frac{1.0}{g(1.0,1.0,1.0,1.0,0.2)} + \frac{1.0}{g(8.0,8.0,8.0,8.0,0.2)} + \frac{1.0}{g(6.0,6.0,6.0,6.0,0.4)} + \frac{1.0}{g(3.0,7.0,3.0,7.0,0.4)}\right]. \quad (17)$$

D. Function Shekel7 (E1)

The function Shekel7 is a 4-D problem in the Shekel family and is the sum of seven terms [24]. In the search space $0 < x_1, x_2, x_3, x_4 < 10$, it has a global minimum at $(x_1, x_2, x_3, x_4) \approx (4.009, 4.001, 4.000, 4.007)$, with value $f \approx -10.39$. It is implemented by first defining g as (16).

Then, Shekel7 is defined as (18)

$$f = -\left[\frac{1.0}{g(4.0,4.0,4.0,4.0,0.1)} + \frac{1.0}{g(1.0,1.0,1.0,1.0,0.2)} + \frac{1.0}{g(8.0,8.0,8.0,8.0,0.2)} + \frac{1.0}{g(6.0,6.0,6.0,6.0,0.4)} + \frac{1.0}{g(3.0,7.0,3.0,7.0,0.4)} + \frac{1.0}{g(2.0,9.0,2.0,9.0,0.6)} + \frac{1.0}{g(5.0,5.0,3.0,3.0,0.3)}\right]. \quad (18)$$

E. Function Hartman3 (E1)

The function Hartman3 is the first of two problems in the Hartman family and is the sum of four Gaussians in a three-dimensional (3-D) space [24]. In the search space $0 < x_1, x_2, x_3 < 1$, it has a global minimum at $(x_1, x_2, x_3) \approx (0.115, 0.555, 0.852)$ with value $f \approx -3.86$. It is implemented by first defining g as (19)

$$g(a_1,a_2,a_3,p_1,p_2,p_3) = a_1(x_1-p_1)^2 + a_2(x_2-p_2)^2 + a_3(x_3-p_3)^2. \quad (19)$$

Then, Hartman3 is defined as the sum of the four Gaussian terms in (20)

$$\begin{aligned} f = &-1.0\exp\left[-g(3.0,10,30.0,0.3689,0.1170,0.2673)\right] \\ &-1.2\exp\left[-g(0.1,10,35.0,0.4699,0.4387,0.7470)\right] \\ &-3.0\exp\left[-g(3.0,10,30.0,0.1091,0.8732,0.5547)\right] \\ &-3.2\exp\left[-g(0.1,10,35.0,0.038\,15,0.5743,0.8828)\right]. \end{aligned} \quad (20)$$

F. Function Hartman6 (E1)

The function Hartman6 is the second of the two problems in the Hartman family and is the sum of four Gaussians in a six-dimensional space [24]. In the search space $0 < x_1, x_2, x_3, x_4, x_5, x_6 < 1$, it has a global minimum at $(x_1, x_2, x_3, x_4, x_5, x_6) \approx (0.201, 0.150, 0.477, 0.275, 0.312, 0.657)$ with value $f \approx -3.32$. It is implemented by first defining g as (21)

$$\begin{aligned} &g(a_1,a_2,a_3,a_4,a_5,a_6,p_1,p_2,p_3,p_4,p_5,p_6) \\ &\quad = a_1(x_1-p_1)^2 + a_2(x_2-p_2)^2 + a_3(x_3-p_3)^2 \\ &\qquad + a_4(x_4-p_4)^2 + a_5(x_5-p_5)^2 + a_6(x_6-p_6)^2. \end{aligned} \quad (21)$$

Then, Hartman6 is defined as the sum of the four Gaussian terms in (22)

$$\begin{aligned} f = &-1.0\exp\left[-g(10.0,3.0,17.0,3.5,1.7,8.0,0.1312,\right. \\ &\qquad \left.0.1696,0.5569,0.0124,0.8283,0.5886)\right] \\ &-1.2\exp\left[-g(0.05,10.0,17.0,0.1,8.0,14.0,0.2329,\right. \\ &\qquad \left.0.4135,0.8307,0.3736,0.1004,0.9991)\right] \\ &-3.0\exp\left[-g(3.00,3.5,1.70,10.0,17.0,8.0,0.2348,\right. \\ &\qquad \left.0.1451,0.3522,0.2883,0.3047,0.6650)\right] \\ &-3.2\exp\left[-g(17.0,8.0,0.05,10.0,0.1,14.0,0.4047,\right. \\ &\qquad \left.0.8828,0.8732,0.5743,0.1091,0.0381)\right]. \end{aligned} \quad (22)$$

G. Function Schubert3 (E2)

The function Schubert3 is a 3-D problem that has roughly 5^3 local minima [25]. In the search space $-10 < x_1, x_2, x_3 < 10$, it has a global minimum at $(x_1, x_2, x_3) = (-1, -1, -1)$ with value $f = 0$. It is implemented by first defining u as (23)

$$u(x_i,a,k,m) = \begin{cases} k(x_i-a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i-a)^m, & x_i < -a. \end{cases} \quad (23)$$

Then, Schubert3 is defined as (24)

$$\begin{aligned} f = &\left(\frac{\pi}{n}\right)\left\{k_1\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-k_2)^2 \right. \\ &\qquad \left. \times\left[1+k_1\sin^2(\pi y_{i-1})\right] + (y_n-k_2)^2\right\} \\ &+\sum_{i=1}^{n} u(x_i,10,100,4) \\ &n = 3,\ k_1 = 10,\ k_2 = 1,\ y_i = 1+\frac{1}{4}(x_i+1). \end{aligned} \quad (24)$$

H. Function Griewank2 (M2)

The function Griewank2 is a 2-D problem that has 500 local minima [24]. In the search space $-100 < x_1, x_2 < 100$, it has a global minimum at $(x_1, x_2) = (0, 0)$ with value $f = 0$.

Griewank2 is defined as (25)

$$f = \sum_{i=1}^{n} \frac{x_i^2}{d} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}} + 1\right), \quad n = 2,\ d = 200. \quad (25)$$

I. Function Schubert5 (M2)

The function Schubert5 is a five-dimensional problem that has roughly 15^3 local minima [25]. In the search space $-5 < x_1, x_2, x_3, x_4, x_5 < 5$, it has a global minimum at $(x_1, x_2, x_3, x_4, x_5) = (1, 1, 1, 1, 1)$ with value $f = 0$. It is implemented by first defining u as (23).

Then, Schubert5 is defined as (26)

$$\begin{aligned} f = k_3 \Bigg\{ & \sin^2(\pi k_4 x_1) + \sum_{i=1}^{n-1} (x_i - k_5)^2 \\ & \times \left[1 + k_6 \sin^2(\pi k_4 x_{i+1})\right] + (x_n - k_5)^2 \\ & \times \left[1 + k_6 \sin^2(\pi k_7 x_n)\right] \Bigg\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4) \\ n = 5,\ k_3 = 0.1,\ k_4 = 3,\ & k_5 = 1,\ k_6 = 1,\ k_7 = 2. \quad (26) \end{aligned}$$

VI. Experimental Results

Sample-Sort and SA were compared by applying each to a set of global optimization problems. Both algorithms require an initial temperature T_H, a final temperature T_L, a cooling schedule, and the size of the region δ around the current state to search for a candidate solution. The parameters T_H, T_L, and the cooling schedule were set automatically at run-time by sampling the given problem.

SA starts at T_H and multiplies the temperature by a factor to decrease T_H to T_L within a given number of iterations. On the other hand, Sample-Sort assigns temperatures statically to each of the individual samplers. In Sample-Sort, the temperature of the first sampler K_1 was set to T_L and the temperature of the last sampler K_m was set to T_H. The temperature difference between the samplers is usually more coarse than the temperature difference between iterations in SA. For example, Sample-Sort with ten samplers has only ten temperatures.

The quality of the solution for Sample-Sort and SA is dependent upon T_H, T_L, and the cooling schedule. Both Sample-Sort and SA automatically set these parameters at run-time by generating 100 random solutions in the search space and calculating the cost of each solution. Then, the method from [16] was used to determine T_H and T_L.

The temperature T_H was set by defining the energy $\Delta E = |\bar{f}| + \sigma$, where $\bar{f}$ is the average cost of the random solutions and σ is the standard deviation. The probability p of accepting a higher cost solution was set to 0.75. Then, (27) was solved for T to find T_H

$$p = \exp\left(-\frac{\Delta E}{T}\right). \quad (27)$$

The temperature T_L was determined by sorting the random solutions from lowest to highest cost and setting ΔE to the difference in cost between the lowest-cost solution and the next higher cost solution. The probability of accepting a higher cost solution p was set to 0.01. Then (27) was solved for T to find T_L.

The geometric cooling schedule $T \leftarrow \alpha T$ was used to set the static temperatures of the samplers in Sample-Sort and for decreasing the temperature in SA at each iteration. In Sample-Sort, the number of samplers m was fixed and the temperatures of samplers K_1 and K_m were set to T_L and T_H, respectively. Then (28) was solved for the cooling coefficient α with $T_k = T_L$ and $k = 1$. Using T_H and the calculated α, the temperatures of the samplers T_k for $1 \le k \le m$ were set to (28)

$$T_k = T_H \alpha^{m-k} \quad (28)$$

where $\alpha < 1$. For SA, (28) was solved for α with $T_k = T_L$ and $(m - k)$ equal to the number of iterations minus one.

A candidate solution in the sampling phase of Sample-Sort and in SA was determined by first randomly picking one of the function variables with equal probability. Then, this variable was modified by adding a value equal to $\pm\delta$, where δ was chosen uniformly from $\pm 1\%$ of the range of the variable.

Sample-Sort and SA were compared by fixing the number of iterations per dimension, applying both algorithms 100 times to each problem, and calculating the probability that the algorithm reached a solution that was within 5% of the known global optimum [23]. This was done over a range of iterations per dimension as suggested in [23]. The range of iterations per dimension was 1000, 10 000, and 100 000.

For Sample-Sort, the number of iterations per *sampler* was set to the number of iterations per dimension divided by the number of samplers so that both Sample-Sort and SA performed the same number of sampling operations and, therefore, the same number of function evaluations. The computational requirements of the sorting were ignored, but will be discussed in the conclusions. The number of samplers m for Sample-Sort was set to 10, the neighborhood N was set to 1, and the results are in Table I.

The difference between the probability of Sample-Sort and SA finding a solution within 5% of the known global optimum is shown for each function. A value greater than zero means that Sample-Sort had a higher probability of finding a solution, while a value less than zero means that SA had a higher probability. An overall average score was given for each number of iterations per dimension by summing the scores of the individual functions and dividing by the number of functions.

As seen in Table I, Sample-Sort has a lower overall score at 1000 iterations per dimension and a slightly higher score at 10 000 and 100 000 iterations per dimension. Recall that Sample-Sort with ten samplers and 1000 iterations per dimension will perform 100 function evaluations per dimension per *sampler*, while at 10 000 iterations per dimension each sampler performs 1000 function evaluations per dimension per sampler.

Then, the number of samplers m for Sample-Sort was set to 100 anticipating more samplers with less temperature difference between adjacent samplers would improve the quality of the solutions. The results are in Table II. As seen in Table II, the overall score *decreases* for Sample-Sort when the number of samplers is increased at a given number of iterations per dimension. Increasing the number of samplers requires decreasing the number

TABLE I
SAMPLE-SORT VERSUS SA: PROBABILITY WITHIN 5% OF OPTIMUM WITH $m = 10$ AND $N = 1$ FOR 1000, 10 000, AND 100 000 ITERATIONS PER DIMENSION OVER 100 RUNS

Iterations per Dimension			1 000			10 000			100 000		
Problem	Difficulty	Dim.	SS	SA	SS−SA	SS	SA	SS−SA	SS	SA	SS−SA
Branin	E1	2	0.20	0.59	-0.39	0.24	1.00	-0.76	0.44	1.00	-0.56
GoldPrice	E1	2	0.34	0.01	0.33	0.82	0.06	0.76	0.75	0.25	0.50
Shekel5	E1	4	0.03	0.60	-0.57	0.91	0.66	0.25	0.95	0.82	0.13
Shekel7	E1	4	0.05	0.66	-0.61	0.92	0.90	0.02	0.97	0.97	0.00
Hartman3	E1	3	0.54	0.91	-0.37	1.00	0.99	0.01	1.00	1.00	0.00
Hartman6	E1	6	0.01	0.99	-0.98	1.00	1.00	0.00	1.00	0.26	0.74
Schubert3	E2	3	0.20	0.92	-0.72	0.93	1.00	-0.07	0.93	1.00	-0.07
Griewank2	M2	2	0.00	0.10	-0.10	0.14	0.23	-0.09	0.23	0.37	-0.14
Schubert5	M2	5	0.00	0.06	-0.06	0.69	0.47	0.22	1.00	0.87	0.13
Average Score					-0.39			0.04			0.08

TABLE II
SAMPLE-SORT VERSUS SA: PROBABILITY WITHIN 5% OF OPTIMUM WITH $m = 100$ AND $N = 1$ FOR 1000, 10 000, AND 100 000 ITERATIONS PER DIMENSION OVER 100 RUNS

Iterations per Dimension			1 000			10 000			100 000		
Problem	Difficulty	Dim.	SS	SA	SS−SA	SS	SA	SS−SA	SS	SA	SS−SA
Branin	E1	2	0.13	0.59	-0.46	0.72	1.00	-0.28	0.79	1.00	-0.21
GoldPrice	E1	2	0.08	0.01	0.07	0.92	0.06	0.86	0.88	0.25	0.63
Shekel5	E1	4	0.00	0.60	-0.60	0.33	0.66	-0.33	1.00	0.82	0.18
Shekel7	E1	4	0.00	0.66	-0.66	0.35	0.90	-0.55	1.00	0.97	0.03
Hartman3	E1	3	0.62	0.91	-0.29	1.00	0.99	0.01	1.00	1.00	0.00
Hartman6	E1	6	0.00	0.99	-0.99	0.37	1.00	-0.63	1.00	0.26	0.74
Schubert3	E2	3	0.02	0.92	-0.90	0.78	1.00	-0.22	0.90	1.00	-0.10
Griewank2	M2	2	0.02	0.10	-0.08	0.02	0.23	-0.21	0.34	0.37	-0.03
Schubert5	M2	5	0.00	0.06	-0.06	0.07	0.47	-0.40	1.00	0.87	0.13
Average Score					-0.44			-0.19			0.15

of function evaluations per dimension per sampler to provide a fair comparison with SA. Notice that the Sample-Sort score in Table I for ten samplers at 10 000 iterations per dimension is positive like in Table II for 100 samplers at 100 000 iterations per dimension. In both cases, Sample-Sort is performing 1000 function evaluations per dimension per sampler. Therefore, as in SA, it is important that each *sampler* perform a sufficient number of function evaluations. This is analogous to performing a sufficient number of evaluations at each temperature in SA.

Sample-Sort was also tested using neighborhoods N of 3 and 5 with $m = 10$, but the results are not presented here. Varying the neighborhood did not significantly change the results when compared with $N = 1$.

VII. CONCLUSION

A new algorithm called Sample-Sort has been introduced, theoretically analyzed, and experimentally validated on a set of global optimization problems. It has m samplers performing the generate, evaluate, and decide cycle each at a different temperature. The parameters for Sample-Sort include the number of samplers m, the size of the region δ around the current state to search for a candidate solution, and the cooling schedule.

Sample-Sort and SA were found to be comparable if the number of iterations per sampler was sufficient for Sample-Sort. For example, when the number of iterations per dimension *per sampler* was 100 then Sample-Sort did not perform as well as SA. But, when the number of iterations per dimension per sampler was 1000 Sample-Sort had a slightly higher score, which corresponds to a higher probability of finding a solution within 5% of the global optimum. In the future, adaptive techniques would further improve the performance. A disadvantage of Sample-Sort is the extra computational requirements of exchanging solutions with neighboring samplers during the sort phase.

The main advantage of Sample-Sort is not that it is better than SA, but that it permits SA to be performed across an array of samplers. The proposed algorithm can be viewed as an SA algorithm that is artificially extended across an array of samplers. The sequence of temperatures for serial SA is replaced with an array of samplers operating at static temperatures and the single stochastic sampler is replaced with a set of samplers. The set of samplers use a biased generator to sample the same distribution of a serial SA algorithm to maintain the same convergence property. If the evaluation phase dominates the computational requirements, Sample-Sort could take advantage of parallel processing while maintaining the same convergence property of SA.

REFERENCES

[1] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, pp. 1087–1092, 1953.
[2] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[3] D. R. Thompson and G. L. Bilbro, "Comparison of a genetic algorithm with a simulated annealing algorithm for the design of an ATM network," *IEEE Commun. Lett.*, vol. 4, no. 8, pp. 267–269, Aug. 2000.
[4] G. L. Bilbro, "Fast stochastic global optimization," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, no. 4, pp. 684–689, Apr. 1994.
[5] B. Hajek, "Cooling schedules for optimal annealing," *Math. Oper. Res.*, vol. 13, no. 2, pp. 311–329, 1988.

[6] F. Romeo and A. Sangiovanni-Vincentelli, "A theoretical framework for simulated annealing," *Algorithmica*, vol. 6, no. 5, pp. 302–345, 1991.
[7] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. New York: Wiley, 1989.
[8] R. Azencott, *Simulated Annealing: Parallelization Techniques*. New York: Wiley, 1992.
[9] S.-Y. Lee and K. G. Lee, "Synchronous and asynchronous parallel simulated annealing with multiple Markov chains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 10, pp. 993–1008, Oct. 1996.
[10] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, "A parallel simulated annealing algorithm for the placement of macro-cells," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, no. Sep., pp. 838–847, 1987.
[11] E. E. Witte, R. D. Chamberlain, and M. A. Franklin, "Parallel simulated annealing using speculative computation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 2, no. 4, pp. 483–494, Apr. 1991.
[12] T. M. Nabhan and A. Y. Zomaya, "A parallel simulated annealing with low communication overhead," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 12, pp. 1226–1233, Dec. 1995.
[13] E. Onbaşoğlu and L. Özdamar, "Parallel simulated annealing algorithms in global optimization," *J. Global Optim.*, vol. 19, pp. 27–50, 2001.
[14] A. Bevilacqua, "A methodological approach to parallel simulated annealing on an SMP system," *J. Parallel Distrib. Comput.*, vol. 62, pp. 1548–1570, 2002.
[15] D. C. W. Pao, S. P. Lam, and A. S. Fong, "Parallel implementation of simulated annealing using transaction processing," *Proc. Inst. Elect. Eng.—Comput. Digit. Tech.*, vol. 146, no. 2, pp. 107–113, Mar. 1999.
[16] S. W. Mahfoud and D. E. Goldberg, "Parallel recombinative simulated annealing: a genetic algorithm," *Parallel Comput.*, vol. 21, no. 1, pp. 1–28, 1995.
[17] K. Kurbel, B. Schneider, and K. Singh, "Solving optimization problems by parallel recombinative simulated annealing on a parallel computer—an application to standard cell placement in VLSI design," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 28, no. 3, pp. 454–461, Jun. 1998.
[18] B. Li and W. Jiang, "A novel stochastic optimization algorithm," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 30, no. 1, pp. 193–198, Feb. 2000.
[19] V. Delport, "Parallel simulated annealing and evolutionary selection for combinatorial optimization," *Electron. Lett.*, vol. 34, no. 8, pp. 758–759, 1998.
[20] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970.
[21] G. L. Bilbro and W. E. Snyder, "Optimization of functions with many minima," *IEEE Trans. Syst., Man, Cybern.*, vol. 21, no. 4, pp. 840–849, Jul./Aug. 1991.
[22] G. L. Bilbro, M. B. Steer, R. J. Trew, C.-R. Chang, and S. G. Skaggs, "Extraction of the parameters of equivalent circuits of microwave transistors using tree annealing," *IEEE Trans. Microwave Theory Tech.*, vol. 38, no. 11, pp. 1711–1718, Nov. 1990.
[23] A. Törn, M. M. Ali, and S. Viitanen, "Stochastic global optimization: problem classes and solution techniques," *J. Global Optim.*, vol. 14, pp. 437–447, 1999.
[24] A. Törn and A. Žilanskas, *Global Optimization*. Berlin, Germany: Springer-Verlag, 1989.
[25] A. Dekkers and E. Aarts, "Global optimization and simulated annealing," *Math. Programm.*, vol. 50, pp. 367–393, 1991.

Dale R. Thompson (M'00) received the B.S. and M.S. degrees in electrical engineering from Mississippi State University, Mississippi State, and the Ph.D. degree in electrical engineering from North Carolina State University, Raleigh, in 1990, 1992, and 2000, respectively.

He is an Assistant Professor at the University of Arkansas, Fayetteville. He was an Electronics Engineer in the Communications Group at the U.S. Army Engineer Research and Development Center, Vicksburg, MS, from 1992 to 2000. He has been with the Department of Computer Science and Computer Engineering at the University of Arkansas since 2000. His research interests are network design, survivable networks, and grid computing.

Griff L. Bilbro (SM'94) received the B.S. degree in physics from Case Western Reserve University, Cleveland, OH, and the Ph.D. degree in physics from the University of Illinois, Urbana-Champaign.

Before joining North Carolina State University (NCSU), Raleigh, he designed computer models of complex systems in a contract research laboratory and was a Systems Programmer in a software startup company. He is now a Professor with the Department of Electrical and Computer Engineering, NCSU. He has published in device physics, image analysis, global optimization, neural networks, and circuits. His current interests include device physics, vacuum electronics, and microwave device modeling.