Pergamon

Computers Math. Applic. Vol. 28, No. 6, pp. 1–8, 1994

0898-1221/94 $7.00 + 0.00

0898-1221(94)00147-2

Simulated Annealing and Tabu Search in the Long Run: A Comparison on QAP Tasks

R. BATTITI*
Dipartimento di Matematica, Università di Trento and INFN
Via Sommarive, 38050 Povo (Trento), Italy
battiti@science.unitn.it

G. TECCHIOLLI
Istituto per la Ricerca Scientifica e Tecnologica and INFN
Via Sommarive, 38050 Povo (Trento), Italy
tec@irst.it

(Received and accepted March 1994)

Abstract—Simulated Annealing (SA) and Tabu Search (TS) are compared on the Quadratic Assignment Problem. A recent work on the same benchmark suite argued that SA could achieve a reasonable solution quality with fewer function evaluations than TS. The discussion is extended by showing that the conclusions must be changed if the task is hard or a very good approximation of the optimal solution is desired, or if CPU time is the relevant parameter. In addition, a recently proposed version of TS (the Reactive Tabu Search) solves the problem of finding the proper *list size* with an automatic memory-based reaction mechanism.

Keywords—Combinatorial optimization, Heuristic search, Tabu search, Simulated annealing, Quadratic assignment problem.

1. INTRODUCTION

The competitive performance of "general-purpose" combinatorial optimization algorithms is still an open issue. In particular, [1] argues that randomized local search (also called "repeated local minima search" in the present work, or RLMS) provides better asymptotical results than the Simulated Annealing algorithm of [2]. On the other hand, by considering the experimental results of [3] on the Quadratic Assignment Problem, one observes that SA produces satisfactory results and uses less function evaluations than the Tabu Search technique used in [4]. Tabu Search schemes are designed to use memory during the search process and therefore to beat RLMS, see [5,6] for two seminal papers and [7] for a brief description of the general TS algorithm.

Hence, we investigate into the matter in detail, particularly to find possible evidence of the poor asymptotic performance of SA on the cited QAP tasks (see Section 3 for a brief description of QAP). This work presents a general discussion about some fundamental differences between SA and TS (Section 2), the output of our numerical experiments in terms of function evaluations (Section 3), and the comparison of the actual CPU times on a specific workstation (Section 4).

We are pleased to acknowledge fruitful discussions with F. Glover and the kind collaboration of J. Paulli for making available a digital copy of the QAP benchmark suite that he used and of S. Struthers for her courteous assistance. Some of the cited preprints are available by anonymous `ftp` at the archive `volterra.science.unitn.it` (`130.186.34.16`), see the `README` file for instructions.
*Author to whom correspondence should be sent.

2. MARKOVIAN VERSUS MEMORY-BASED SEARCH

In our opinion, the *leitmotif* of [3] that "fast and cursory SA is better than thorough and slow TS" does not capture the main difference between SA (that moves after checking one neighbor) and TS (that moves after evaluating the entire neighborhood). In fact, there is no problem in designing "fast and cursory" versions of TS: for instance, a *partial evaluation of the neighborhood* was used in our previous work about training neural nets with the Reactive Tabu Search [8]. For that application, the choice of evaluating a small and randomly-extracted subset of the neighborhood reduced CPU times and maintained satisfactory performance.

The "quantum leap" of TS is caused by its intense use of memory, and therefore of learning, during the search process. Some memory-based heuristics were already considered in the sixties (see the historical discussion about local search in [9]), but only recently the availability of cheap RAM components permits efficient search schemes based on intense memory usage, like the RTS algorithm of [10].

On the contrary, SA is a Markov chain Monte Carlo method, and therefore, memoryless. By definition, the distribution of the random variable $X(t+1)$ in a Markov chain is mediated entirely by the value of $X(t)$: the past history *does not* influence the current move. Recently, [1] demonstrated that the asymptotic performance of SA is worse than that of Repeated Local Minima Search (repeated generation of random starting points and greedy search of the nearest local minimum by exhaustive neighborhood evaluation). This result is not surprising: when the "temperature" parameter has been decreased so that it is much smaller than the height of the barrier around the current "attraction basin," either the basin contains the global optimum, or SA will spend an enormous amount of time before escaping, and therefore, it will be surpassed even by the simple RLMS. The main problem is that SA will continue to jump up and down without noticing that the movement is confined. The "convergence theorems" of SA (for example with the logarithmic schedule $T_t \propto T_1/\log(t)$) are of dubious practical interest. Citing from [11], "these results say that SA ... retains enough traces of exhaustive search to guarantee asymptotic convergence, but if exhaustive search were a realistic option we would not be using SA anyway." This does not mean that SA will not solve some relatively simple tasks (see also Section 3).

The observation that "TS uses the same amount of information to perform one move as simulated annealing uses in N moves" [3], N being the neighborhood size of the QAP, is not accurate both because it is possible to apply a partial neighborhood evaluation to TS, and because the information obtained from N points in the neighborhood usually is *smaller* than the information obtained from N points on the search trajectory. For structured problems one expects that the function values of the neighboring points will be correlated with the current value: guessing a value in the neighborhood is easier than guessing an arbitrary value.[1] In the QAP problem, one does indeed obtain less information per point in the neighborhood than for uncorrelated configurations, but one also needs fewer operations (if n is the QAP size, $O(n^2)$ operations are required for evaluating a single random point, while $O(n^2)$ operations are sufficient for the entire neighborhood if efficient schemes are used, see for example [4,13]). In addition, when already-visited points are found again during the search, no new information is obtained. In fact, this is an underlying motivation for prohibiting moves that would revisit configurations in a version of Tabu Search.

[1] A quantity that measures the average amount of information gained after observing the value of a stochastic variable x with probability density function $p(x)$ is the *entropy* $H = -\int_{-\infty}^{+\infty} p(x)\log p(x)\,dx$. Now, if the value of the zero-mean function $f(t)$ along the search trajectory is modeled by an autoregressive process of order one: $f(t) = \alpha\, f(t-1) + Z(t)$, with $0 < \alpha < 1$ and $Z(t)$ a zero-mean random variable, the variances are related by $\sigma_Z^2 = \sigma_f^2\,(1-\alpha^2)$. Points in the neighborhood will have a smaller spread than points on the trajectory, but a smaller spread means a smaller entropy ($H_Z = H_f - \log(1/\sqrt{(1-\alpha^2)})$). The uncertainty, and therefore the information, for points on the trajectory is larger than that for points in the neighborhood. Autoregressive models for the QAP task are considered in [12].

The statement that "the most difficult part of applying TS is finding the right list size" [3] is not appropriate anymore. In the RTS scheme of [10] and [14], the list size is adapted in an automated way by *reacting* when configurations are repeated along the search trajectory. The term "Reactive Tabu Search" derives from this property. All solutions found during the search are stored and the discrete dynamical system that generates the search trajectory is regulated by the entire past history. The asymptotic space-time requirements can be reduced to one bit and a small number of CPU cycles per RTS iteration by using *hashing* strategies. The criterion for prohibiting moves is the same that was used in [4]: an exchange is prohibited if it places both units into locations that they had occupied within the last T iterations. The basic RTS reaction is that T increases if a configuration along the trajectory is repeated, and decreases if no repetitions occur in a suitable time interval. If a configuration is repeated more than once, a diversification phase based on a *random walk* is activated ("escape" mechanism). The following tests use the algorithm that is described and discussed in detail in [10] (REPT = 2, CHAOS = 1, the memory is never cleaned during the search).

3. COMPARING FUNCTION EVALUATIONS

Let us briefly summarize the notation and the SA algorithm. The QAP problem dimension is n, the function to be minimized is:

$$f(\phi) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, b_{\phi(i)\phi(j)}, \qquad \phi : \text{ permutation of } \{1, 2, \ldots, n\}.$$

For a concrete application, n units must be assigned to n locations such that the sum of products distance $\times$ flow is minimized. The number of neighbors obtained by exchanging two arbitrary units is $N = n(n-1)/2$, the number of function evaluations is M.

The Simulated Annealing version used in [3] is considered for an extensive comparison on the same QAP tasks adopted here. Before the SA run is started, one selects $N/4$ random neighbors of the starting configuration and determines $\Delta_{\max}$ and $\Delta_{\min}$, the maximal and minimal increase in the objective function value. The initial value of the temperature T is then set equal to $\Delta_{\min} + 0.1\ (\Delta_{\max} - \Delta_{\min})$. During the run, T is decreased after each step by $0.1(\Delta_{\max} - \Delta_{\min})/M$, so that T equals $\Delta_{\min}$ at the end of the search.

The maximum problem dimension considered in [3] is 50, the maximum number of TS iterations is 1,600. For RTS, these values correspond to some seconds of CPU time on current workstations and it is doubtful that the results measure the potential of TS for solving "hard" tasks. It was then decided to investigate what happens for a larger number of search steps. The number of allowed iterations was increased to 30,000 for RTS and $30{,}000 \times N$ for SA. Some of the tasks considered in [3] are solved exactly in a much lower number of iterations. Both for this reason and for brevity, results are presented only for the most difficult tasks: `nug30`, proposed in [15], `ste1` and `ste3`, proposed in [16], `ran3` and `ran6` proposed in [3].

Let us start from what appears to be the most difficult task in Paulli's paper: `ste3`, the task proposed by Steinberg. Figure 1 reports the results obtained with RTS, RLMS, and SA. SA was used in a "fast" version ($M = 1{,}600 \times N$), and in a "slow" version ($M = 30{,}000 \times N$). Let us call these two options FAST-SA and SLOW-SA. The average deviations in percent from the best known solution, and the standard deviation σ of the performance distribution are reported. The optimal values found during our tests always coincide with those listed in [3]. Averages are over 100 runs with different random initial configurations; therefore, the statistical error on the average deviation is approximately $\sigma/10$. The bottom plots in Figure 1 have the Y-axis rescaled so that the details in the 3% range can be seen better.

At $M = 1600 \times N$ function evaluations on our runs of SA gives: %Dev = **2.1**, with $\sigma = 2.3$, while RTS obtains: %Dev = **5.7**, $\sigma = 3.3$. The performance approximately duplicates that shown in [3], which considers the TS version of [4].

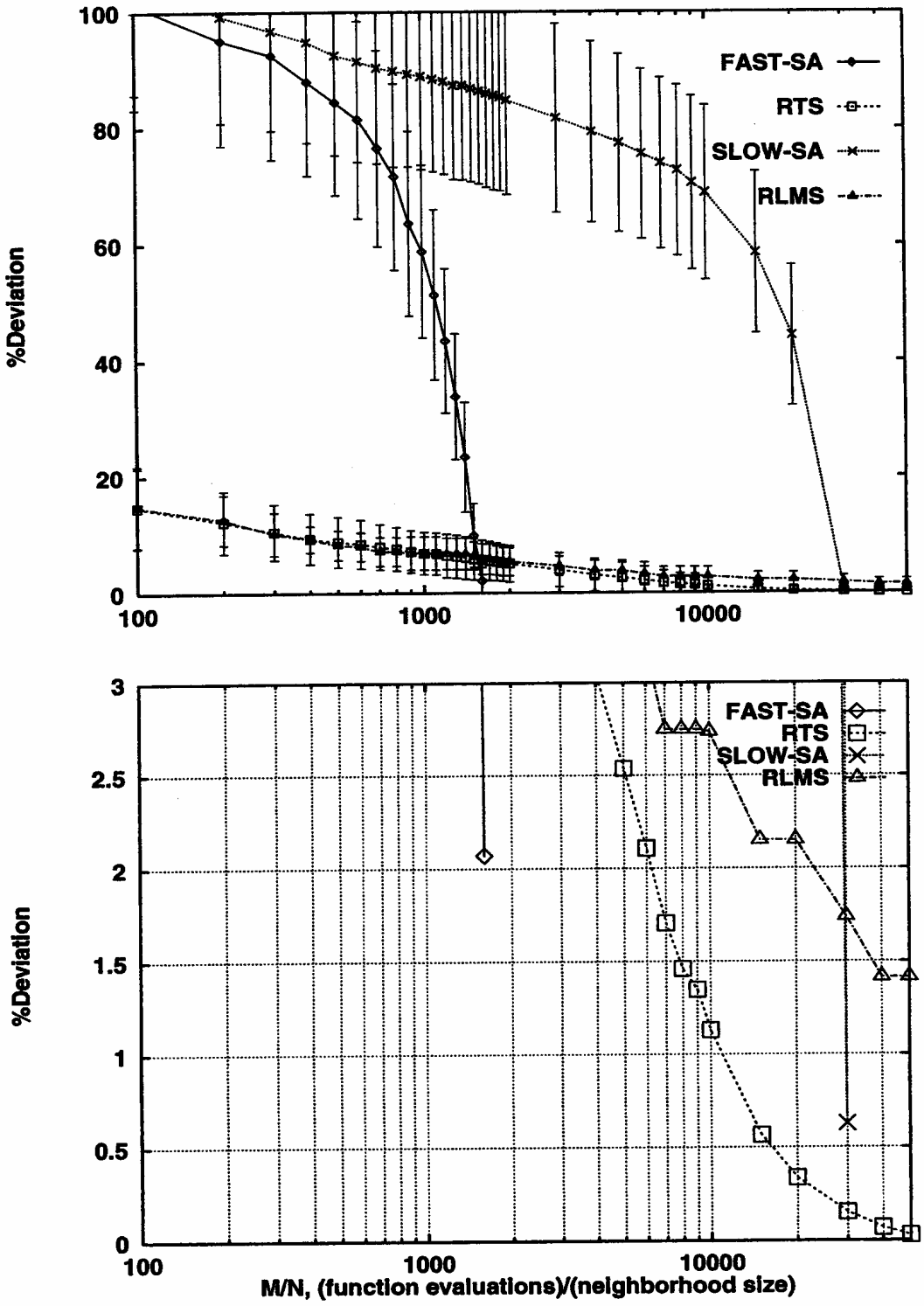

Figure 1. Steinberg task `ste3`: RTS (first 50,000 iterations) versus SA and RLMS. Average deviation from best known solution with $\pm\sigma$ bars (top), Y-axis rescaled (bottom).

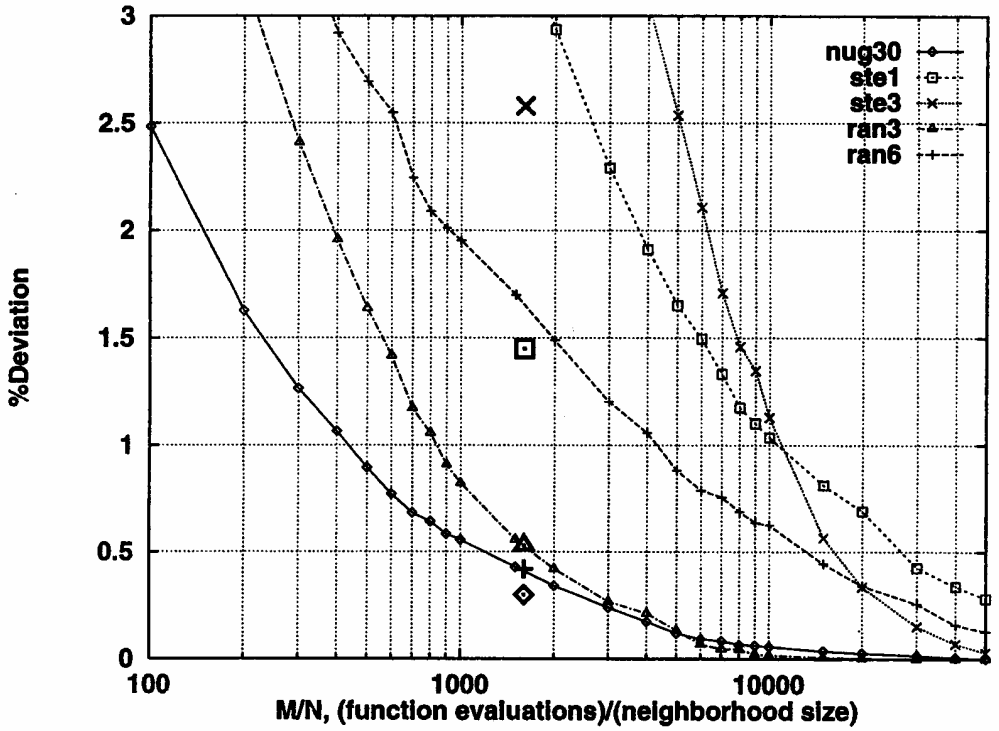

Figure 2. Evolution of average RTS results during the first 50,000 iterations (curves for different tasks). The SA performance at $M/N = 1,600$ is shown with thick points.

But RTS easily overcomes FAST-SA: at 10,000 iterations RTS gives %Dev = **1.12**, σ = 1.53, at 30,000 it gives %Dev = **0.15**, σ = 0.46. For a fair comparison, SLOW-SA at M = 30,000 × N obtains %Dev = **0.59**, σ = 0.97: the expected quality of the solution is about four times worse than that obtained by RTS, for the same number of function evaluations. The results are surprising if one compares the trajectory length: 18,900,000 iterations in SLOW-SA (630 × 30,000), only 30,000 in RTS.

Let us note that the difference between RTS and RLMS is negligible up to 1,000 iterations, but it becomes substantial at large iteration numbers: at 30,000 iterations RLMS gives only %Dev = **1.7**, σ = 1.1. This competitive advantage is precisely what an intelligent use of memory is supposed to accomplish. RLMS tends to repeat previously found local minima with a higher frequency than RTS, which builds upon the diversification enforced by the prohibition period and by the "escape" mechanism. Some additional observations can be derived from Figure 1:

- SA produces interesting configurations only in the very last moments of its life, when the "temperature" is small (i.e., when SA accepts almost only downward moves).
- The "life span" of SA is crucial. If SA is too fast, one is left with a result that can be insufficient, but if SA is slow, one has to wait until the end of the search before obtaining good approximations of the global optimum.
- On the contrary, the "life span" of RTS is not crucial. No parameters have to be chosen by the user for specific tasks: the search continues until either the maximum allowed time elapses or until an acceptable result is encountered. Naturally the best values obtained are recorded as soon as they are found and listed at the end of the search.

How does one choose the proper number of search steps (and therefore, the "annealing schedule") for SA? In the absence of specific prescriptions, one executes a trial-and-error process until the results are satisfactory. This can be easily done for simple tasks, but is difficult and time-consuming for large and complex tasks.

A conclusion of [3] was that "at the time TS catches up with SA the solution quality is usually within 0.5–1% of the best known solution." Now, it can happen that requiring only about 99% of the global optimum simplifies some tasks that would otherwise be difficult. We find different results in tests on the Knapsack problem [14]: the number of instances out of 100 that are solved by SA at a given number of function evaluations remains substantially lower with respect to the number solved by RTS. SA was used with time-dependent penalty to take the constraints into account. For large Knapsack tasks ($n = 500$, with the same number of constraints), SA beats RTS during the first phase but consistently loses for large numbers of function evaluations, although different annealing schedules were tried.

Table 1. Comparison of RTS versus SA at $30,000 \times N$ function evaluations. Averages of 100 runs.

Task	Size	Best sol.	**SA** % Dev.	(σ)	**RTS** % Dev.	(σ)
nug30	30	6124	**0.056**	(0.11)	**0.015**	(0.03)
ste1	36	4763	**0.35**	(0.40)	**0.42**	(0.41)
ste3	36	7926	**0.59**	(0.97)	**0.15**	(0.46)
ran3	30	34574	**0.025**	(0.039)	**0**	(0)
ran6	50	149358	**0.08**	(0.10)	**0.25**	(0.31)

Table 1 lists the performance comparison for the most difficult tasks at $30,000 \times N$ function evaluations. RTS obtains comparable or better solutions for all tasks apart from `ran6`, in which SA is still significantly better. The evolution of the average RTS performance as a function of the number of iterations is shown in Figure 2.

4. COMPARING CPU TIMES

End users of Combinatorial Optimization algorithms are interested in solving tasks with minimum time and effort. The total effort is difficult to evaluate, but certainly it includes contributions from setting parameters appropriately. In what follows, our consideration is limited to the CPU time and we answer to the following question: Which algorithm is expected to provide the best performance if the same CPU time is allotted?

To fix a benchmark, the CPU time considered is that required by a run of SA for $1,600 \times N$ iterations. If we are interested in the asymptotic behavior, SA requires $O(n)$ cycles to compute a QAP function value on the trajectory, while RTS requires only $O(1)$ cycles per point in the neighborhood. In fact, the entire neighborhood can be evaluated in $O(n^2)$ time, see for example [13].

The asymptotic CPU times measured on our workstation[2] are approximated by:

$$CPU_t(SA) \approx 2\mu s \;\; t \;\; n, \qquad CPU_t(RTS) \approx 4\mu s \;\; t \;\; n^2,$$

where t is the number of iterations, i.e., the length of the search trajectory. Starting from the above measurements, the number of RTS iterations corresponding to $t(SA) = 1,600 \times N$ iterations is:

$$t(RTS) = \frac{2\mu s \;\; t(SA) \; n}{4\mu s \;\; n^2} = 400 \; (n - 1).$$

In Table 2, the average results at equivalent CPU times are shown for the same tasks considered in Table 1. To facilitate the comparison, the SA performance derived from [3] is also listed.

At equivalent CPU times, RTS provides significantly better results for all tasks. While the absolute time constants in the above formulas depend on the machine, language, and compiler, the different asymptotic dependence on n will probably remain, at least for general-purpose sequential processors. Let us note that the points in the neighborhood can be evaluated in

[2]Spark station 10 mod. 41 from SUN Microsystems Inc., SunOS Release 4.1.3, GNU gcc C compiler.

Table 2. Comparison of RTS versus SA at the same CPU time (corresponding to $1,600 \times N$ SA iterations). Averages of 100 runs.

Task	Size	Best sol.	**SA** % Dev.	**RTS** % Dev.
nug30	30	6124	**0.30**	**0.04**
ste1	36	4763	**1.45**	**0.81**
ste3	36	7926	**2.58**	**0.61**
ran3	30	34574	**0.53**	**0.009**
ran6	50	149358	**0.42**	**0.35**

parallel, if a parallel processor or a dedicated VLSI chip is used. In this case, the real time will be proportional to the number of RTS steps, and therefore, to M/N, while it is proportional to M for SA.

5. CONCLUSIONS

It is not surprising that SA beats TS when a limited number of iterations are executed on relatively simple tasks. At the very beginning of the search SA "jumps around" in the search space with an almost *random walk* (if the initial "temperature" is not too small with respect to the typical function differences), and then converges in a stochastic way to the bottom of an attraction basin with a sufficiently large probability measure, while TS is busy evaluating neighboring points. But, if the final solution provided by SA is not acceptable, the user must start again, usually with a slower annealing schedule, until the run produces acceptable results. No such tuning is required by RTS, which uses all information acquired during an optimization run to diversify the search in an automatic way. For the tasks considered, RTS needs less CPU time than SA to reach average results in the 1% area.

This paper argued that a fast evaluation of the neighborhood can be executed also in the Tabu Search framework [8] and that the main difference is given by the intense memory usage of TS during the search trajectory generation. It is far from our intention to claim that SA is not appropriate as an optimization algorithm, although other competitors do exist for simple tasks, like RLMS or stochastic versions of it. What this paper claims is that the small overhead caused by the efficient memory usage and adaptation mechanisms of RTS is, in some cases, repaid by avoiding possible "traps" in the search process, such as attraction basins with large probability measure but severely suboptimal function values.

REFERENCES

1. A.G. Ferreira and J. Zerovnik, Bounding the probability of success of stochastic methods for global optimization, *Computers Math. Applic.* **25**, 1–8 (1993).
2. S. Kirkpatrick, C.D. Gelatt, Jr. and M.P. Vecchi, Optimization by simulated annealing, *Science* **220**, 671–680 (May 1983).
3. J. Paulli, Information utilization in simulated annealing and Tabu search, *COAL Bulletin* **22**, 28–34 (1993).
4. E. Taillard, Robust Tabu search for the quadratic assignment problem, *Parallel Computing* **17**, 443–455 (1991).
5. F. Glover, Tabu search—Part I, *ORSA Journal on Computing* **1**, 190–206 (1989).
6. F. Glover, Tabu search—Part II, *ORSA Journal on Computing* **2**, 4–32 (1990).
7. N. Dubois and D. de Werra, EPCOT: An efficient procedure for coloring optimally with Tabu search, *Computers Math. Applic.* **25** (10/11), 35–46 (1993).
8. R. Battiti and G. Tecchiolli, Training neural nets with the reactive Tabu search, Technical Report UTM 421, Univ. of Trento, Italy, (November 1993).
9. C.H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, Englewood Cliffs, NJ, (1982).
10. R. Battiti and G. Tecchiolli, The reactive Tabu search, Technical Report 9302-13, IRTS, Trento, Italy, (1992); *ORSA Journal on Computing* (to appear).
11. R.M. Neal, Probabilistic inference using Markov chain Monte Carlo methods, Technical Report CRG-TR-93-1, Univ. of Toronto, (September 1993).

12. R. Battiti and G. Tecchiolli, Parallel biased search for combinatorial optimization: Genetic algorithms and TABU, *Microprocessors and Microsystems* **16**, 351–367 (1992).
13. A.M. Frieze, J. Yadegar, S. El-Horbaty and D. Parkinson, Algorithms for assignment problems on an array processor, *Parallel Computing* **11**, 151–162 (1989).
14. R. Battiti and G. Tecchiolli, Local search with memory: Benchmarking RTS, Technical Report, Math. Dept., Univ. of Trento, Italy, (October 1993).
15. C.E. Nugent, T.E. Vollmann and J. Ruml, An experimental comparison of techniques for the assignment of facilities to locations, *Journal of Operations Research* **16**, 150–173 (1968).
16. L. Steinberg, The backboard wiring problem: A placement algorithm, *SIAM Review* **3**, 37–50 (1960).