A HYBRID ALGORITHM TECHNIQUE

Miroslaw Malek, Mohan Guruswamy,
Howard Owens, and Minhir Pandya*

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-89-06 March 1989

* All of the authors are affiliated with: The Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas 78712. Telephone: 512-471-5704, ARPANET: malek@emx.utexas.edu

Abstract

A new hybrid algorithm technique(HAT) based on the idea of mixing two or more algorithms is proposed. Though the algorithm is general and may be applied to the majority of optimization problems, a hybrid algorithm search technique(HAST) is the focus of this paper. As an example of HAST, this paper describes mixing of simulated annealing and tabu search algorithms into a new hybrid search algorithm applied to the traveling salesman problem. A brief introduction to the simulated annealing and tabu search algorithms is given followed by a description of how we mixed these algorithms to form a new parallel hybrid search technique. Comparison of our algorithm mixer with simulated annealing and tabu search indicates consistently better results. Examples include 33, 42, 50, 57, 75, and 100 city problems from the literature. Solutions for the 50 and 75 city problems outperform best known published to date results.

Key Words : Algorithm, search techniques, simulated annealing, tabu search, traveling salesman problem.

1 INTRODUCTION

The idea of combining two or more different algorithms into a single hybrid algorithm was inspired by the possibility of this new algorithm performing better than any of its component algorithms individually. The result is a new class of algorithms under the umbrella of hybrid algorithms techniques(HAT). The hybrid algorithm combines the strengths of the individual algorithms so that the resulting algorithm provides a combination of the following advantages:

1. can produce better solutions,
2. and/or produce solutions in less time.
3. can effectively handle problems with larger input sizes, especially with respect to NP problems.

These advantages seem to be gained without major new disadvantages.

Within HAT, the algorithms we are interested are different search techniques applicable to solving combinatorial optimization problems, and are therefore called hybrid algorithm search techniques(HAST). In order to guarantee the optimum solution, all possible solutions must be considered. Unfortunately, many of these problems fall into the class of NP-complete, and therefore the set of all possible solutions is too large to consider. Heuristics are therefore used to test only more promising subsets of the possible solutions. The existing algorithms cannot, therefore, assure the optimum solution will be found.

Several algorithms to solve combinatorial optimization problems exist. Hybridization of some of these algorithms is intended to combine the strengths of their respective heuristic techniques into a better algorithm. This new algorithm should produce solutions more near optimal, or in less time, or both. The algorithm which produces satisfactory results in less time can also be applied to larger problems.

We expect our new hybrid algorithm technique to be general and applicable to the majority of optimization problems. Some examples of these problems where HAST could be applied are in computer-aided design (e.g., integrated circuit or printed circuit board placement and routing), scheduling, resource allocation, test generation, integer programming and a number of graph heuristic algorithms such as coloring and partitioning. To demonstrate viability of our hypothesis of increased performance we have chosen the Traveling Salesman Problem (TSP) which is an easily defined problem

in combinatorial optimization research. The problem consists of finding the shortest Hamiltonian circuit (circuit that includes every node) in a complete graph. The nodes of the graph represent cities of a map and the edges are weighted with the distance between each pair of cities. We will define and test our algorithms with respect to the traveling salesman problem.

Our objective is to implement two different combinatorial optimization algorithms such that they may execute in parallel and exchange data periodically. The goal is to study the time efficiency and cost of mixing the simulated annealing[1] and tabu search[2] algorithms into a new parallel hybrid search algorithm as compared to these algorithms executing independently. These three search algorithms are tested on the move of the 2-opt heuristic which is based on swapping pairs of edges [3]. Experiments are conducted on six well known problems from the literature, namely, the 33 city, 42 city, 50 city, 57 city, 75 city, and 100 city problems. The 50 city and 75 city problems have no known optimal solutions while the others do.

2 DESCRIPTION OF ALGORITHMS

2.1 SIMULATED ANNEALING

Simulated annealing uses the analogy of annealing to guide the use of moves which increase cost[4]. The main body of the algorithm (see Figure 1)consists of two loops, where the inner loop is nested within an outer loop. The **inner loop** runs till an equilibrium is reached. In this loop, a possible move is generated using the 2-opt exchange and the decision of accepting the chosen move is made using an *accept* function. If the move is accepted, it is applied to the current tour to generate the next tour state. Equilibrium is reached when large swings in energy (miles) no longer occur.

The outer loop checks for the stopping condition to be met. Each time the inner loop is completed, the temperature [1](T) is updated using an *update temperature* function and the stopping criterion is checked again. This continues until the stopping criterion is met.

The accept function

IF ($\Delta C < 0$) RETURN(TRUE)
ELSEIF ($e^{-\frac{\Delta C}{T}} > random(0,1)$) RETURN(TRUE)

[1]Temperature, by analogy to physical annealing process, represents the control variable for accepting uphill moves.

ELSE RETURN(FALSE)

assigns a probability of accepting a move based on the current temperature and the change in cost(ΔC) which would result in the tour if the move is accepted. If ΔC is negative, meaning the cost would go down, a probability of one is assigned to acceptance. Otherwise the probability of acceptance is assigned the value

$$probability\ of\ acceptance = exp(-\Delta C/T).$$

A randomly generated number is used to test whether the move is accepted.

In our implementation of the simulated annealing algorithm we chose the stopping criteria to be a set temperature such that the probability of accepting an uphill move is very close to zero. After a fixed number of iterations in our algorithm we assume equilibrium is reached. Finally, to update temperature following equilibrium, we simply multiply the current temperature by a constant ALPHA, where ALPHA is less than one, to obtain the new temperature. Thus our implementation of the simulated annealing algorithms has as inputs the initial temperature, the number of iterations to simulate equilibrium, and ALPHA. These parameters allow tuning of the algorithm for the TSP problem.

2.2 Tabu Search

Tabu search is another optimization technique for solving permutation problems [2]. In this technique, we start with an arbitrary permutation and make a succession of moves to transform this permutation into an optimal one (or

```
WHILE (stopping criterion not met)
        WHILE (equilibrium not reached)
                Generate-next-move()
                IF (Accept(Temperature, change-in-cost))
                    Update-tour()
        ENDWHILE
        Calculate-new-temperature()
ENDWHILE
```

Figure 1: Simulated annealing algorithm.

as close to the optimum as possible). In determining the shortest tour for a given set of cities, the tabu search procedure starts with a randomly generated tour and makes a succession of 2-opt exchanges that reduces the cost. At each step all the possible 2-opt moves are examined and the one which gives the best improvement in tour cost is chosen. In order to prevent the process from being trapped at a local optimum, this algorithm allows moves that increases the tour cost (*uphill* moves). It is most likely that the moves following the uphill moves will reverse it and retrace the path back to the local optimum. This results in cycling. To avoid this, the procedure maintains a history of recent moves and classifies moves that reverse these as tabu. This enables the search process to escape the local optima and explore new areas of the solution state space.

Creating a tabu classification for the moves involves the identification of the *swap attributes* which could be one of the following :

- the cities involved in the swap, or
- the *positions* they occupy before/after the swap, or
- the direction in which the cities move in the swap.

The tour of the cities is represented in a one-dimensional array format, with the array index denoting the *position* of the city in the tour. If the city moves from a lower index to a higher index during a swap, then it is said to move right. Conversely, if it moves from a higher index to a lower one, then it is said to move left.

We also need to identify the tabu classifications based on the attributes so that we can specify a set of moves as tabu. These attributes are discussed in detail in the next section. Figure 2 shows the tabu search strategy superimposed on the hill climbing heuristic.

The algorithm examines all the swaps of the current tour and keeps track of the *best-swap-value*, however, those that are classified as tabu are rejected if they do not satisfy the *aspiration criteria* [2]. In other words, we restrict the set of available swaps. The tabu status of the move is overridden and the move is accepted if the swap-value satisfies the aspiration level. The *best-swap* among all the available swaps for the current tour is obtained at the exit of the inner loop. In the hill climbing method, the best-swap-value

[2]This is a concept based on the observation that it may be advantageous to override a tabu restriction if it promises a better solution or a new search space.

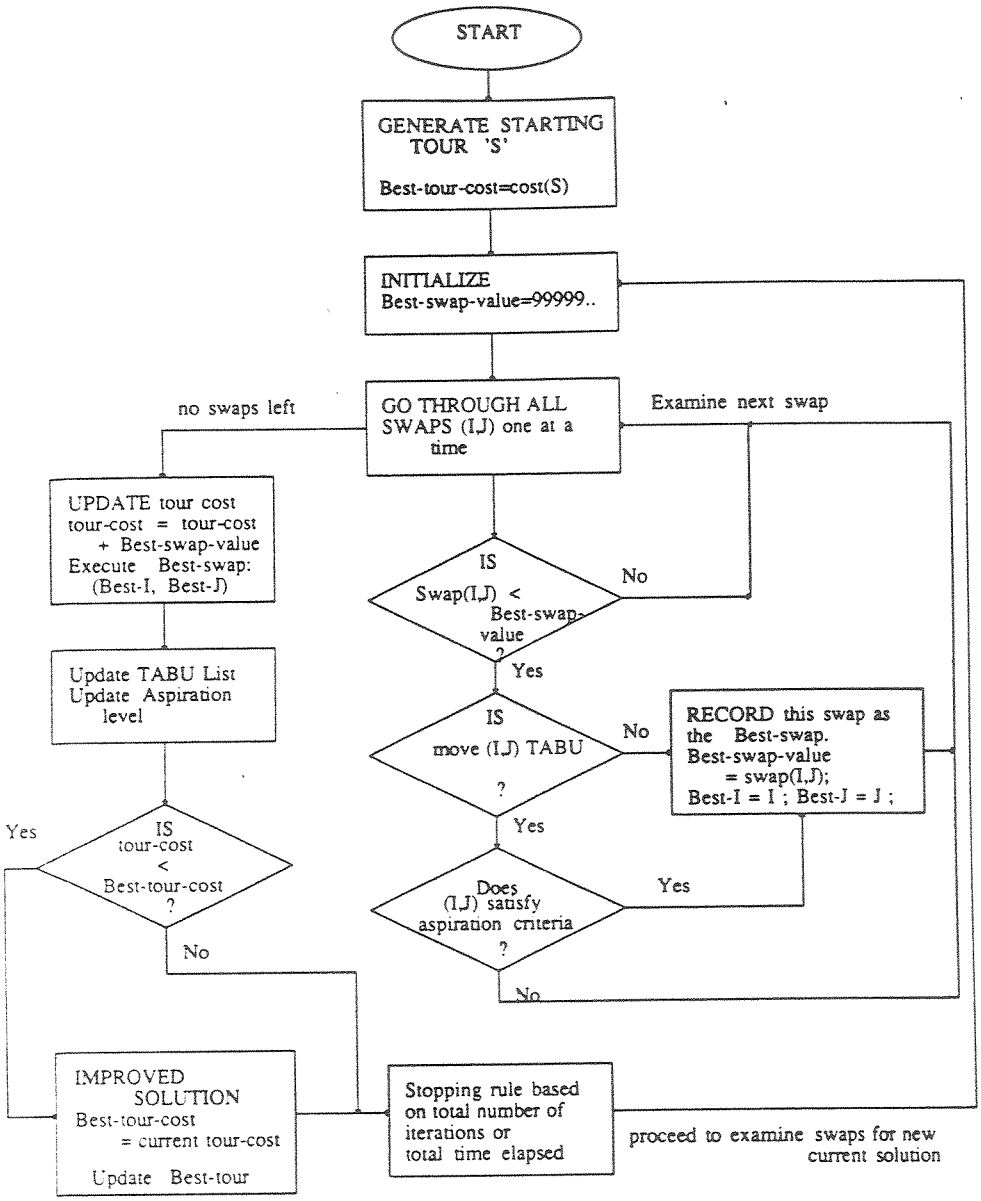

Figure 2: Tabu search.

is usually negative indicating a reduction of the current tour cost. When it becomes positive, the process has reached its terminating condition.

In tabu search, the best-swap is executed regardless of the sign of the best-swap-value. The best swap from the inner loop is accepted even if it results in a higher tour cost. This helps the process to climb out of local optima. The outer loop keeps track of the *best-tour* and its cost. The tabu list is also updated by including the current move made. The stopping criteria is usually a fixed set of iterations or a fixed computation time specified in the input.

2.2.1 Tabu conditions

This section presents examples of the move attributes and the tabu restrictions based on these attributes. In our implementation we select only one tabu condition for a specific tabu search process.

1. **Vector (I, J, POSITION(I), POSITION(J))** -
   this vector is maintained to prevent any swap in the future from resulting in a tour with city I and city J occupying POSITION(I) **and** POSITION(J) respectively.

2. **Vector (I, J, POSITION(I), POSITION(J))** -
   the same vector to prevent a swap resulting in city I occupying POSITION(I) **or** city J occupying POSITION(J).

3. **Vector (I, POSITION(I))** -
   to prevent city I from returning to POSITION(I).

4. **City I** -
   to prevent city I from moving LEFT of current position.

5. **City I** -
   to prevent I from moving in any direction.

6. **Vector (J, POSITION(J))** -
   to prevent city J from returning to POSITION(J).

7. **City J** -
   to prevent city J from moving RIGHT of current position.

8. **City J** -
   to prevent J from moving in any direction.

9. **Cities I and J** -
   to prevent both from moving.

Conditions 3 through 9 have been established by assuming that cities I and J are identified such that POSITION(I) < POSITION(J). It is obvious that condition 1 is the least restrictive and 9 is the most restrictive. Conditions 3, 4 and 5 have increasing restrictiveness.

2.3 SIMULATED ANNEALING/TABU SEARCH HYBRID (SATH)

As can be seen from the previous two sections, simulated annealing and tabu search use very different approaches to search for optimal solutions to combinatorial optimization problems. Although both of these algorithms provide good results on some problems, neither can guarantee the optimal solution will be found in real time. This, of course, leaves room for improved algorithms. We have therefore developed a hybrid algorithm in attempt to produce better performance.

SATH is a simulated annealing/tabu search hybrid algorithm, the first in a new class of easily parallelizable hybrid algorithms. SATH incorporates both simulated annealing and tabu search as low level algorithms with a high level algorithm to mix the results from each. The idea is to execute each low level algorithm for some specified amount of time, the results of which are evaluated by the high level algorithm. The low level routines are than restarted in a more promising area of the solution space. This process is repeated as many times as is necessary or desired.

The SATH algorithm can be realized with the simulated annealing and tabu search portions implemented as subroutines. These subroutines could be executed, one after the other, followed by analysis of the results by a higher level routine. However, one of the most important features of this hybrid algorithm is the ease with which it may be executed in parallel. Each low level algorithm can be executed in parallel with a supervising process to synchronize execution and analyze results. This opens up the possiblility of executing several low level algorithms in parallel, any number of which may be instances of simulated annealing or tabu search with different operating parameters. Interprocess communication is minimal and only occurs between a low level algorithm and the single high level algorithm. Speedup can therefore be linear with the number of processors as long as the number of processors does not exceed the number of low level algorithms.

3 IMPLEMENTATION OF ALGORITHMS

3.1 Implementation of simulated annealing

To implement the simulated annealing algorithm, as described earlier, requires specifying the stopping and equilibrium criteria, and the update temperature rule. The stopping criteria chosen for the algorithm is for the temperature to reach a specified value. This stopping temperature is chosen such that the probability of accepting an uphill move is very close to 0. We make the typical assumption that the equilibrium is reached after a fixed number of iterations. The update temperature rule is

$$\textit{new-temperature} = \alpha * \textit{temperature},$$

where α is a constant less than one.

The consequence of choosing these parameters to be simple constants is some increase in computation time. It forces the choice of α and the number of iterations to be tuned for the critical regions of temperature. This critical region requires a slow annealing rate. However, at high and low temperatures it is possible to anneal at a faster rate.

The simulated annealing algorithm implemented has as inputs the initial temperature, the number of iterations to simulate equilibrium, and α. These parameters allow the algorithm to be tuned for any TSP problem.

3.2 Implementation of Tabu search

3.2.1 Data structures

In order to determine the tabu status of a move and update the tabu list efficiently, we need well-designed data structures. As an example, the tabu identification and tabu-list update for one of the tabu conditions(condition-4) is described below.

There are two lists, *tabu-left* and *tabu-list*. The first list, *tabu-left*, indicates which cities are prevented from moving left of their current position. The second list, *tabu-list*, contains a fixed number of cities that had been moved to the right in the last k iterations (k is the *tabu-list size* which is an input parameter).

Updating the tabu-list with a new city i which was moved right is done by incrementing an index(*ring-index*) to the tabu-list and overwriting the city i at this new index position. This automatically removes the tabu status of the city which was residing in the new position of the index. In order for the index to stay in range of the list, the incrementing is done using a **mod**

operator : new-ring-index = (ring-index + 1) **mod** tabu-size. Similar data structures have been implemented for other tabu conditions.

3.2.2 Aspiration criterion

The aspiration criterion we have used is straightforward. Any tabu move that reduces the current tour-cost to a value below the best-tour-cost obtained by the process so far is accepted. When the move results in a tour-cost lower than the best-tour-cost, it indicates a new path not previously visited and so the move can no longer be termed as tabu. This simple aspiration criterion is :

tour-cost + swap-value(I,J) < best-tour-cost

3.2.3 Tabu list size

This parameter needs experimental tuning. It can be observed that for highly restrictive tabu conditions the tabu list size has to be smaller than for lesser restrictive conditions. If the tabu list size is small, cycling phenomenon will be evident, whereas, if it is large, the process might be driven away from the vicinity of global optimum. The optimum tabu list size will be the one which is long enough to prevent cycling but small enough to explore a continuum of solution space. Experimental results with the 42 and 57 city problems have shown that for tabu conditions 1, 2, 3 and 6, list sizes of the order of the size of the problem is good. For conditions 4, 5, 7 and 8, the list size ranging from 7 to 30 gave us good results.

3.2.4 Long term memory

This is a function designed to enable the process to explore new areas of the solution state space. Applying tabu search from several different starting points is more likely to perform better than exploring from one starting point. Instead of starting at randomly chosen starting points, if we can provide a purposeful alternate starting point for search, we might be able to reach the optimum faster. In our approach for an alternate starting point, we use a long term history function that maintains the edges visited by the process and generates a starting point consisting of edges that have been visited the least. We maintain a two dimensional array of occurrence of each edge. After each 2-opt move, the entries corresponding to all the edges

in the new tour are incremented. After a specified number of iterations, a new starting tour is generated based on the edges that occur least frequently. The results obtained using this memory function are very encouraging. We were able to reach the optimum results for the 42 and 57 city problems consistently from different random starting points.

3.3 Implementation of SATH

We implemented our SATH algorithm by allocating a separate process for each part of the algorithm. The basic implementation includes one main process and two child processes. When the program is executed, a main process is generated which reads in the problem definition. The main process then creates a set of child processes, one of which is a simulated annealing process, the other of which is a tabu search process. After specified time intervals, the child processes are halted and the main process compares their results. It selects a *good* solution for the child processes to continue with. A *good* solution might be the one with the least cost. In case the tour with the least cost had already been given to the child processes, passing the same tour again will result in cycling. To prevent this from happening, the tour with the next to least tour (if not previously encountered) is made the common starting point for the child processes.

Implemented in this fashion the SATH algorithm can be executed on a single processor or on multiple processors with very little effort. The algorithm is also expandable by adding additional simulated annealing and tabu search processes executing with different search parameters. The algorithm can be expanded in this way until there is a process for every available processor.

To execute the SATH algorithm on a single processor requires that only one processor is available to execute the processes. In addition, synchronization was added to assure one process executes to completion before the next one begins.

In our SATH algorithm each simulated annealing process executes with a different annealing schedule. The schedules are chosen as in the accelerated simulated annealing algorithm described in [3]. When the SATH algorithm had multiple tabu search processes, each process had a different tabu condition and a corresponding tabu list size to distribute the search in the solution space.

4 TSP EXPERIMENTS

The experiments on the TSP were performed on a multiprocessor system, the Sequent Balance 8000 computer running the DYNIX[3] operating system, a version of UNIX 4.2bsd[4]. The Sequent Balance 8000 has ten homogeneous 32-bit processors. Our programs are written in PPL (Parallel Program Language)[5] , which is a set of extensions to the C programming language[6]. PPL allows easy parallelization of the SATH algorithm.

The experiments were conducted on the 33, 42, 57 and 100 city problems, each with known optimum solutions. In addition, experiments were conducted on the 50 and 75 city problems, each with unknown optimum solutions. For each problem, six tours were generated randomly and this set was given as starting tours to all algorithms. The SATH algorithm was executed on one, two, four and eight processors, with half the processors executing simulated annealing and the other half executing tabu search. It was found that one processor implementation of mixing the algorithms gave the same results as the parallel version in which both processes were run in parallel on two processors. The time of execution and the best time for one processor implementation were roughly twice that of the two processor parallel implementation. Hence further experiments were conducted on mutiple processors with all processes executing in parallel.

The parameters of the algorithm were tuned for each problem and then these parameters were used for all the six starting tours. The parameters were not changed for each starting tour. The simulated annealing algorithm is supposed to give the best result at the end of the annealing process and so the time at which the best result was obtained is the same as the execution time for the algorithm. Tabu search algorithm maintains a monitor which records the best result as it is generated. The best time is the instance of time at which the best result was generated. In SATH, a monitor keeps track of the best tours at the instances of comparison and the best time refers to the time at which the main process encountered the best result while comparing and passing back tours. Hence in tabu search and SATH, the best time is not necessarily the execution time of the process.

[3]DYNIX is a trademark of Sequent Computer, Inc.

[4]UNIX 4.2bsd is a Berkeley Software Distribution version of UNIX, UNIX is a trademark of AT& T

4.1 Simulated Annealing

Each execution of the simulated annealing algorithm takes as input parameters the number of iterations to approximate equilibrium, the starting temperature, and the cooling rate α. These input parameters allow the algorithm to be tuned for a specific problem. Experiments were performed for the 33, 42, 57 and 100 city problems with proven optimum solution and for 50 and 75 city problems where only best solutions to date are known[7].

For each problem the first step in the experimentation was to tune the algorithm for the problem. During tuning of the algorithm we varied the input parameters throughout a wide range. The number of iterations to simulate equilibrium ranged from 4 to 20. The starting temperatures varied from 10 all the way up to 10,000. α values of .98 down to .50 were tried. For each of these problems a different input parameter set was chosen from the tuning process.

Results from applying our simulated annealing algorithm on the 33 city problem was very good, reaching the known optimal solution from all but one starting tour. The the published optimum solution has a tour cost of 10861 miles. The time required for execution of the problem was 115 seconds.

The process of tuning the algorithm for the 42 city problem resulted in several input parameter sets reaching the same minimum cost solution. The parameter set which found the minimum cost in shortest time took 185 seconds to execute. Using this input parameter set, resulting tour costs ranged from 699 miles to 705 miles. Roughly half of the results had a tour cost of 699 miles for our set of randomly generated starting tours. The optimal solution to the 42 city problem is published to be 699 miles [8]. We observed this tour to be (0 1 2 ... 40 41).

The 57 city problem proved to be more difficult. After tuning the algorithm, we reached the optimum solution of 12955 [9] miles only once on our set of randomly generated starting tours. However, all non-optimal results were within one percent of optimum. Resulting tour costs found ranged from 12955 miles to 13042 miles, requiring 430 seconds to execute.

The 100 city problem was the most difficult. The published optimal solution is 21282 miles when the problem is executed using floating point numbers. This corresponds to a tour cost of 21247 miles when the elements of the cost matrix are truncated, as we did in our experiments for the 100 city problem, for the same tour. While tuning our algorithm for this problem we chose cooling schedules which took up to 10 hours to complete. However,

Table 1: Results of simulated annealing algorithm.

Problem size	Average cost	Average % above best known solution	Average time	Best cost
33	10905	0.4	113	10861
42	701	0.3	185	699
50	432	1.6	280	425
57	12999	0.34	430	12955
75	546	2.1	570	535
100	21500	1.19	1242	21267

we never reached the optimal solution. The cooling schedule we finally chose required 1242 seconds to execute. Using this schedule our results averaged 21500 miles, with the best result of 21267 miles, on our randomly generated set of starting tours.

The final two problems we considered have no proven optimal solutions. We reported the best to date solution to the 50 city problem as 425 miles and 535 miles for the 75 city problem [3]. Executing our simulated annealing algorithm on the 50 city problem resulted in one solution equalling our reported best solution for our set of randomly generated starting tours. Our results ranged from 425 miles to 438 miles with an average cost of 432 miles. The time to execute our algorithm on this problem was 280 seconds.

For the 75 city problem our algorithm resulted in one solution of equal quality to our previously reported results on our set of randomly generated stating tours. The range of results was from 535 miles to 559 miles, with an average tour cost of 546 miles. The time to execute our algorithm on this problem was 315 seconds.

Table 1 shows the results of the simulated annealing algorithm for the six problems studied. Columns 2 and 4 show the average tour cost and average execution time for six randomly selected starting tours. The third column represents the percentage by which the average tour cost is inferior to the optimum or the best known cost. The best cost obtained in the six runs is shown in the last column.

4.2 Tabu search

The first stage in developing the tabu search algorithm was the implementation of the hill climbing heuristic. The hill climbing algorithm was then transformed into the tabu search algorithm using the nine different tabu conditions discussed earlier. The tabu search process has the following input parameters :

- tabu condition,
- tabu list size and
- total number of iterations.

The tabu condition and the tabu list size are two interdependant parameters and the algorithm is very sensitive to both of them. A smaller tabu list size for a weaker tabu condition will result in cycling, whereas, a larger tabu list size for a stronger tabu condition could drive the search process away from the global optimum. A compromise had to be reached and experiments were conducted for all the nine tabu conditions to find out a reasonable range of tabu list sizes for each of these conditions. Generally, tabu list sizes from one-fourth to one-third the number of cities for conditions 4 and 7, and about one-fifth for conditions 5, 8 and 9 gave the best results for the problems tested. Conditions 1, 2, 3 and 6 required tabu list sizes in the vicinity of the problem size. On an average, tabu conditions 4 and 7 produced better results in a shorter time than the other conditions.

The next step was to include the long term memory function. This function requires the tuning of an additional input parameter - the number of iterations in tabu search before generating a new starting tour. The algorithm must be given sufficient search time in its current path before generating a totally different starting tour. The performance of the algorithm with long term memory function was compared with the original tabu search algorithm by making the total number of iterations and the starting points identical for both versions. For every run, the algorithm with the memory function outperformed the simpler version in both computation time and the quality of the solution.

On the 33 and 42 city problems, the tabu search algorithm produced the published optimal results of 10861 and 699 miles from each of the randomly selected starting tours. For the 33 city problem, the number of iterations required to produce optimum result ranged from 29 to 271 iterations taking a total run time of about 30 seconds. For the 42 city problem, the optimum

Table 2: Results of tabu search algorithm.

Problem size	Average cost	Average % above best known solution	Average time (for best cost)	Best cost	Total time (seconds)
33	10861	0	15	10861	31
42	699	0	31	699	31
50	430	1.2	47	426	100
57	12999	0.34	109	12955	260
75	545	1.9	117	537	225
100	21611	1.7	792	21317	1000

was attained in less than 30 seconds in each run with the number of iterations ranging from 33 to 117.

The 57 city problem proved to be more difficult with the optimum of 12955 miles attained twice with six randomly selected starting tours. But, the worst tour cost in these runs was only 13067 miles. The number of iterations for these random starting tours to reach the best tour varied from 73 to 897 iterations requiring a maximum of 260 seconds.

The best solution found by tabu search for the 100 city problem was 21317 miles taking 1193 iterations (about 16 minutes). For the 50 and 75 city problems whose optimal solutions are unknown, tabu search easily beat the best previously published results. The best solution for the 50 city problem had a tour length of 426 miles and tabu search equalled or bettered the best previously known solution of 430 miles in 7 out of 12 runs conducted. The average time at which the best solution was found in each run was 47 seconds. Tabu search superceded the previous best result of 553 miles for the 75 city problem in 11 out of 12 runs. The best solution had a tour length of 537 miles and the average time taken to reach the best solution in each run was 117 seconds. The results of the tabu search algorithm for the six problems tested are tabulated in table 2.

4.3 SATH

Our experiments with SATH included three configurations, namely with two, four, and eight low level processes. In each case there were an equal number of simulated annealing and tabu search processes. Each low level

process executed in parallel. The experiments were conducted much like that described in the previous sections for simulated annealing and tabu search algorithms. First we tuned the SATH algorithm for a given starting tour and then applied the tuned algorithm to a set of six randomly generated starting tours.

For the 33 and 42 city problems the two, four, and eight process SATH algorithms easily found the optimum solution for each starting tour. For both problems the average time required for executing the algorithm was comparable with the tabu search algorithm executing alone. The performance was much better than the simulated annealing algorithm executing alone in terms of both time and quality of solution.

Executing our SATH algorithms on the 57 city problem resulted in solutions with better average quality than either simulated annealing or tabu search algorithms executing alone. In addition, performance for both time and tour costs of solutions improved as the number of low level processes went from two to four and then to eight. The time performance, as compared to simulated annealing and tabu search algorithms alone, was in the intermediate range with tabu search performing the best.

The 100 city problem demonstrated the strength of our SATH algorithm on larger problems. The time performance for the two, four, and eight process SATH algorithms was 458, 444, and 287 seconds respectively. This compares with 792 seconds for tabu search and 1242 seconds for simulated annealing. At the same time the quality of solution was also better for each SATH algorithm. This leds us to believe our SATH algorithm can be applied to larger problems with acceptable performance.

For the 50 and 75 city problems, for which the optimal solutions is unknown, our SATH algorithms provided better average quality of solution than either simulated annealing or tabu search algorithms. For both problems the time performance was in the intermediate range, with tabu search performing the best. The eight process SATH algorithm produced results which was equal to our previously reported best solutions on both of these problems for many of the starting tours. In fact, the average tour cost found for the 50 city problem was just one mile longer than the best tour cost found to date for this problem. Likewise, the average tour cost for the 75 city problem was just 3 miles longer than the best tour cost found to date.

Tables 3, 4 and 5 show the results of SATH with two, four and eight low level processes. Columns 2 and 4 show the average time to obtain the best result for six randomly generated starting tours. Column 3 represents the

Table 3: Results of two processor simulated annealing/tabu search hybrid.

Problem size	Average cost	Average % above best known solution	Average time (for best cost)	Best cost	Total time (seconds)
33	10861	0	19	10861	49
42	699	0	46	699	82
50	427	0.47	106	425	120
57	12986	0.24	170	12955	280
75	542	1.3	158	535	254
100	21450	0.95	458	21267	525

Table 4: Results of four processor simulated annealing/tabu search hybrid.

Problem size	Avg. cost	% Optimum or Best known	Avg. time (for best cost)	Best cost	Total time (seconds)
33	10861	0	15	10861	52
42	699	0	37	699	82
50	427	0.47	79	425	134
57	12982	0.21	144	12955	305
75	540	0.93	158	535	253
100	21402	0.73	444	21247	530

percentage by which the average tour cost is inferior to the optimum cost (or in the case of 50 and 75 city problems, the best known solution). Column 5 shows the best tour cost obtained in the six runs. The last column shows the average of the total execution time for the six runs.

For graphical comparison of the experimental results for the quality of solution refer to Figure 3. Likewise, Figure 4 gives the graphical comparison of the relative time performance. The Figures include data for simulated annealing, tabu search, and two process SATH algorithms. Figure 3 shows how SATH generally outperforms both simulated annealing and tabu search in terms of solution quality. Figure 4 shows how SATH has a intermediate time performance. However, the trend, as the problem size increases,

Table 5: Results of eight processor simulated annealing/tabu search hybrid.

Problem size	Avg. cost	% Optimum or Best known	Avg. time (for best cost)	Best cost	Total time (seconds)
33	10861	0	14	10861	54
42	699	0	33	699	94
50	426	0.24	79	425	128
57	12965	0.07	74	12955	125
75	538	0.56	168	535	292
100	21319	0.33	287	21267	423

is toward better time and cost performance for our SATH algorithm. A in depth analysis of this data does bring out a problem with making comparisons. Each algorithm was executed for different amounts of time on each problem. This makes it difficult to analyze which algorithm would be best if a specified amount of time is the desired limiting factor. It is also difficult to compare time performance for a given quality of solution required.

To give a better idea of the relative performance of the algorithms for equal time or cost requirements, consider Figure 5. This figure was derived from executing each of our algorithms for the 57 city problem again and taking time and tour cost measurements at incremental times during execution. Each point on the graph represents the average tour cost at a given time value for a algorithm given the six randomly generated starting tours. As can be seen in the figure, for any given time or tour cost value, SATH performs better than or equal to tabu search, and always better than simulated annealing. Figure 6 shows a magnified portion of Figure 5 for better clarity.

In addition, SATH performance improves with the number of low level processes. The Figure 7 shows the improvement in the quality of the results as the number of processes in SATH is increased for the 50, 57 and 100 city problems. Figure 8 shows the reduction in time with increasing number of processes.

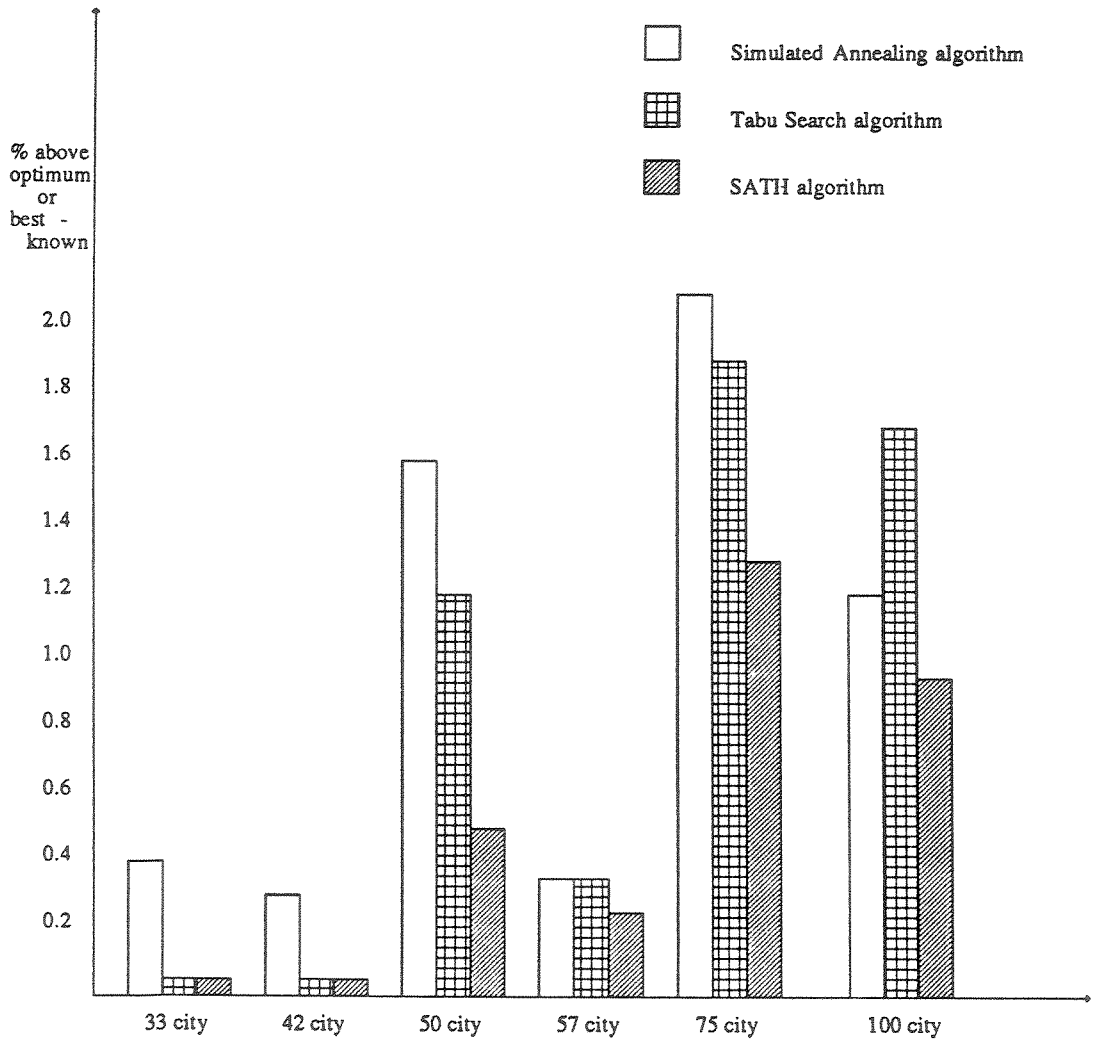

Figure 3: Comparison of solution quality for different problems.

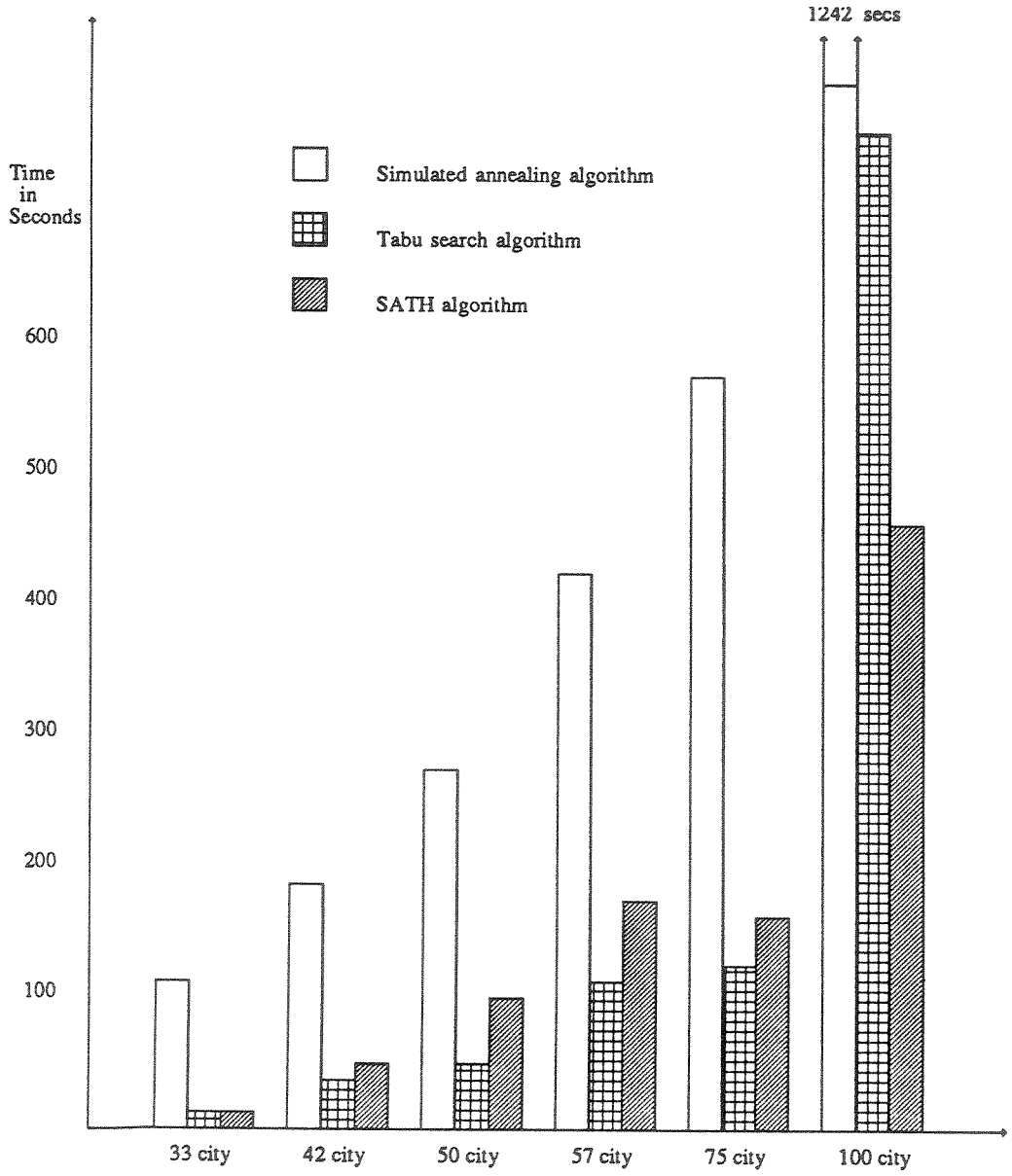

Figure 4: Comparison of execution time for the different problems.

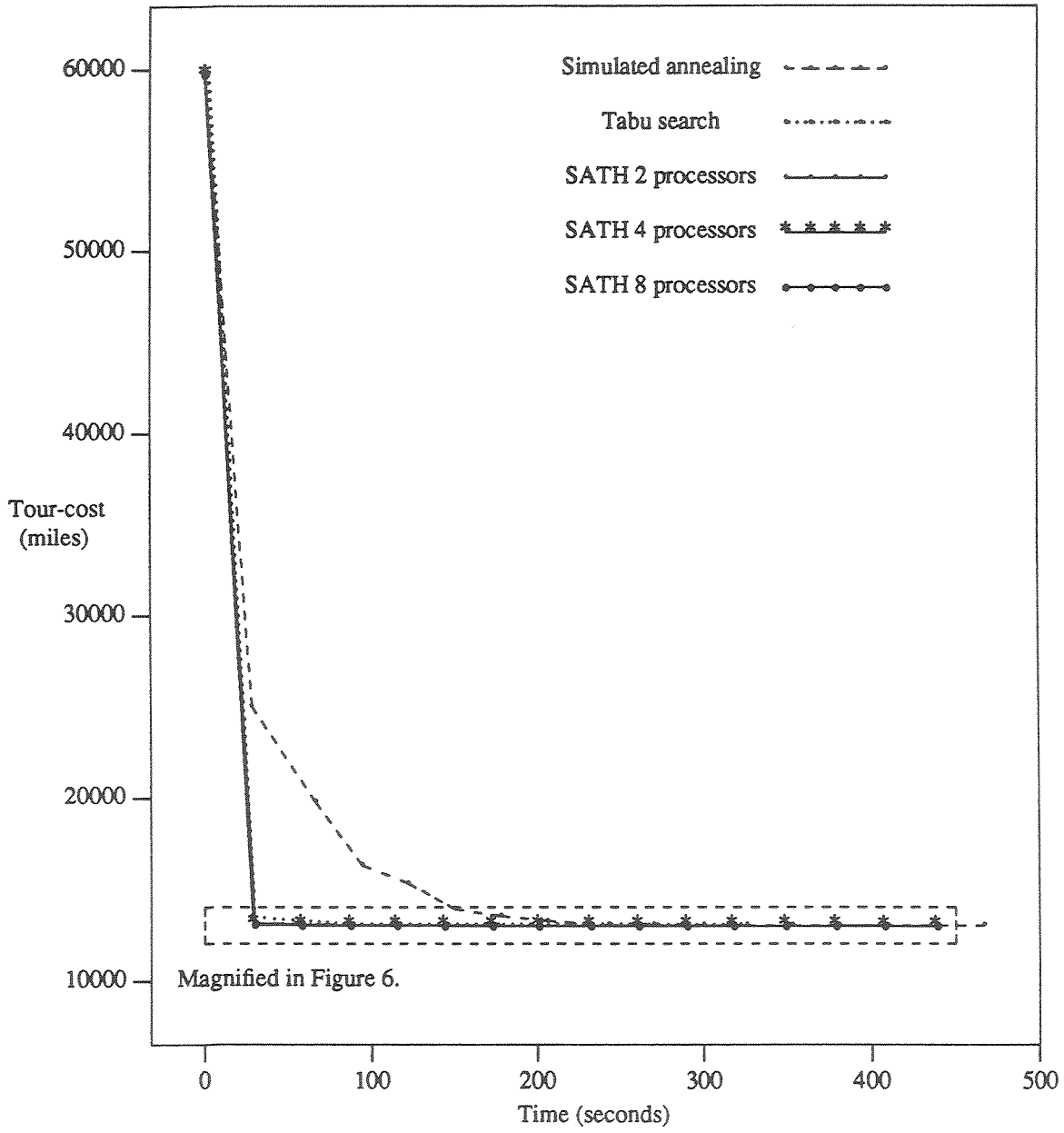

Figure 5: Quality versus time comparison of solutions on the 57 city problem.

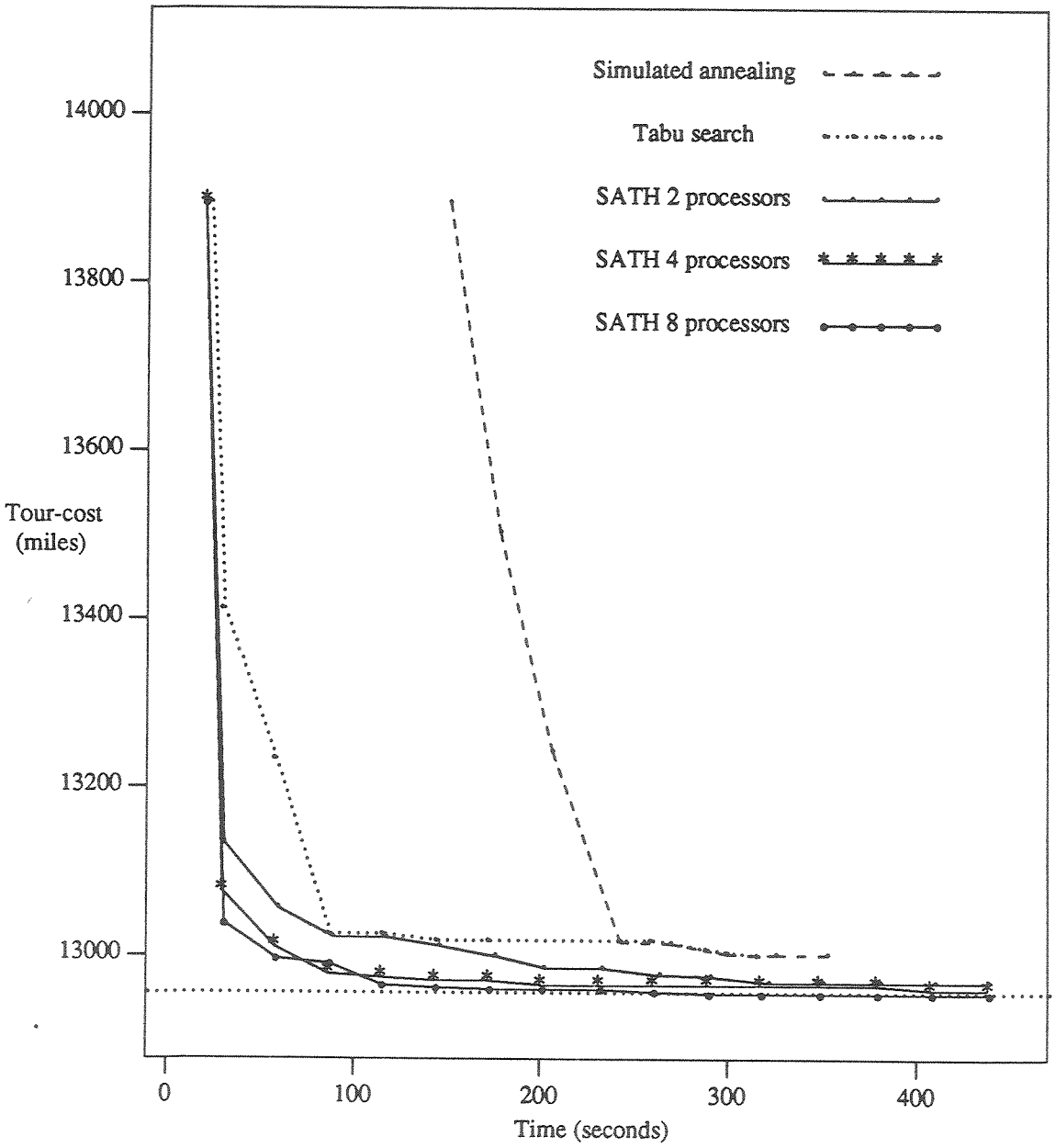

Figure 6: Magnified portion of Figure 5.

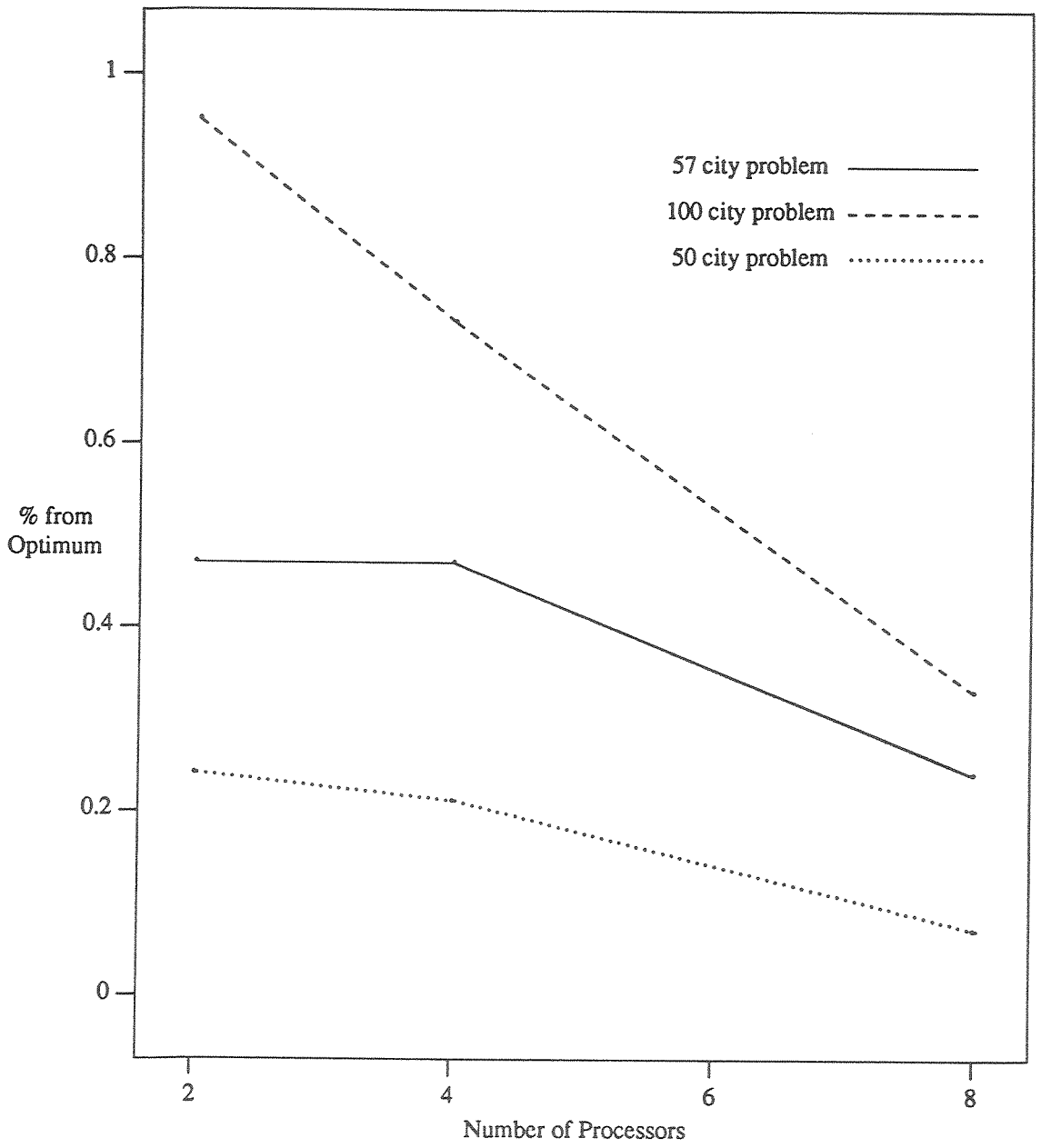

Figure 7: Cost performance vs. number of processors.

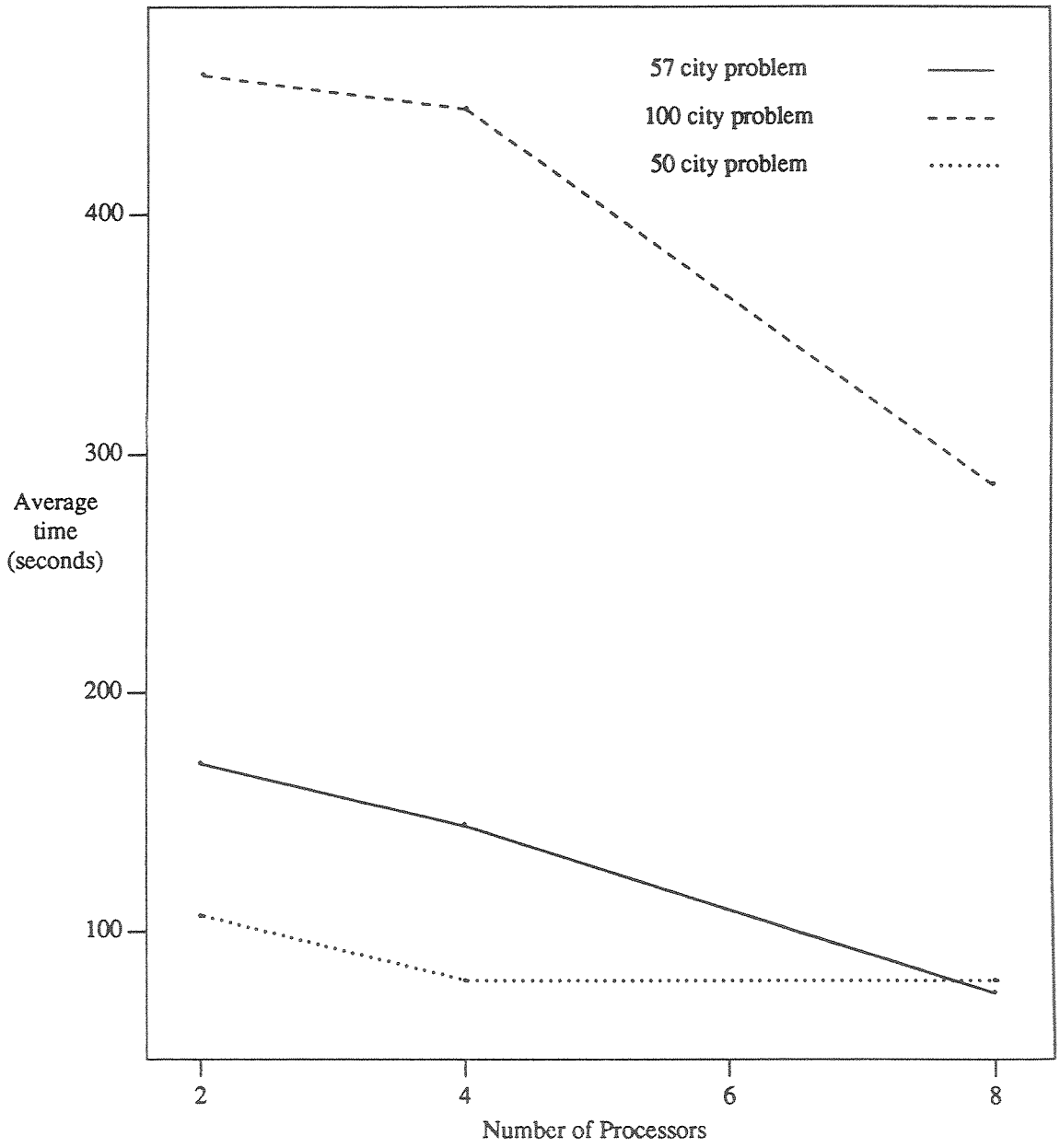

Figure 8: Time for best cost vs. number of processors.

5 CONCLUSIONS

A new hybrid algorithm technique based on the idea of mixing two or more algorithms for improved performance was proposed. The goal of achieving better solutions in less time was demonstrated by implementing HAT for a combinatorial optimization problem. Our experiments with the traveling salesman problem have illustrated the advantages of using a hybrid search technique based on mixing simulated annealing and tabu search algorithms. The hybrid algorithm performs very well for all of the investigated problems, namely 33, 42, 50, 57, 75 and 100 city problems. It holds considerable potential for reducing execution time for solving NP-complete problems and at the same time improving the quality of the solution. In our opinion, the hybrid algorithm is very well suited for other problems too, not necessarily search techniques. With the advent of parallel processing in the computing environment, it becomes especially attractive to exploit the inherent parallelism in our algorithm.

References

[1] Kirkpatrick, S., C. D. Gelatt, M. P. Vechi, "Optimization by simulated annealing", *Science,* May 13, 1983, vol. 220, Number 4598.

[2] Glover, F., "Tabu search methods in Artificial Intelligence and Operations Research", *ORSA Artificial Intelligence Newsletter,* vol. 1, 1987.

[3] Malek, M., M. Guruswamy, H. Owens, and M. Pandya, "Serial and parallel implementations of search techniques for the traveling salesman problem", *Technical Report, Dept. of Electrical and Computer Engineering, The University of Texas at Austin*, 1988.

[4] Hinton, G. E. and T. J. Sejnowski, "Learning and relearning in Boltzmann machines", *Parallel Distributed Processing,* vol. 1, MIT Press, 1986.

[5] Schwetman, H., "PPL Reference Manual", version 1.1, Microelectronics and Computer Technology Corporation, 1985.

[6] Kernighan, B. W. and D. M. Ritchie, "The C Programming Language", Prentice-Hall, 1978.

[7] Christofides N. and S. Eilon, "An algorithm for vehicle dispatching problem", *Operational Res. Q.* 20, (1969) 309-318.

[8] Dantzig, G.B., D.R. Fulkerson and S.M. Johnson, "Solution of a large-scale travelling-salesman problem", *Operations Research* vol. 2, 393-410, 1954.

[9] Karg, R.L. and G.L. Thompson, "A heuristic approach to solving travelling-salesman problems", *Management Science* vol. 10, 225-247, 1964.

[10] Lawler, E. L., J. K. Lenstra, and A. H. G. Rimnooy Kan, eds., *The Traveling Salesman Problem*, North-Holland, 1985.

[11] Lin, S., B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem", *Oper. Res.* 21, 495-516, 1973.

[12] Krolak, P., W. Felts and G. Marble, "A man-machine approach towards solving the travelling salesman problem", *Communications of the Association for Computing Machinery* vol. 14, 327-334, 1971.

[13] Held, M. and R. M. Karp, "The travelling salesman problem and minimum spanning trees, Part I", *Operations Research* vol. 18, pp 1138-1162, 1970; "Part II", *Mathematical Programming 1* pp 6-26, 1971.