Connection Machine Implementation of a Tabu Search Algorithm for the Traveling Salesman Problem

Jaishankar Chakrapani[1] and
Jadranka Skorin-Kapov[2]

[1]Department of Applied Mathematics and Statistics, State University of New York at Stony Brook
[2]Harriman School for Management and Policy, State University of New York at Stony Brook

A tabu search algorithm for the traveling salesman problem (TSP) is developed. The algorithm is based on the well known 2-opt move which is implemented in parallel on the connection machine CM-2. This involves decomposing the evaluation of the whole 2-opt neighborhood into small independent steps that can be executed in parallel by different processors. The implementation is efficient and highly scalable. Implementation details and results of computation for some TSPs from the literature are presented.

Introduction

The traveling salesman problem (TSP) continues to receive an enormous attention from researchers in computer science, operations research, and other related disciplines, as a classical NP-hard combinatorial optimization problem. An instance of a TSP is given by a set of, say, n cities and a distance matrix among them. The objective is to find the shortest tour (i.e.,a trajectory that starts from a city, visits all other cities exactly once, and returns to the starting city). The TSP can be formulated as a special case of a more general NP-hard classical problem, the quadratic assignment problem (QAP) which is characterized by two $n \times n$ matrices $D=\{d_{ij}\}$ and $F=\{f_{kl}\}$ as follows

$$\min \sum_{i=1}^{n} \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{l=1}^{n} d_{ij} f_{kl} x_{ik} x_{jl}$$

subject to

$$\sum_{k=1}^{n} x_{ik} = 1 \ \forall i$$

$$\sum_{k=1}^{n} x_{ik} = 1 \ \forall k$$

$$x_{ik} = 0,1 \ \forall i,k$$

In the TSP context, $x_{ik}=1$ if city i is the kth visited city, matrix D is the distance matrix between the cities, and matrix F reduces to a cyclic permutation matrix.

TSP is, in general, difficult to solve optimally and therefore a “good” suboptimal solution is sought by applying a heuristic method. Some recent references to such methods include (Rei92, Joh90, Ben90). However, some instances of dimension up to 2392 were optimally solved by Padberg and Rinaldi (Pad91) using a combination of cutting plane and branch and bound methods.

Most of the heuristic approaches to the TSP are local optimization procedures, where a given tour is modified in an attempt to achieve a shorter tour. In the general context of local optimization procedures, a *move* is a predefined way to perturb a solution to get another. The *neighborhood* of a solution consists of all solutions that can be reached from that solution by performing one move. Local optimization proceeds greedily from the current solution performing the move that gives the best improvement. In the context of TSPs, one of the popular moves is the so-called 2-opt move. Performing a 2-opt move involves deleting two tour segments, which in turn identifies uniquely two new segments that should be added to obtain a feasible tour.

The main problem with such an approach is the "locality" of the search. If, at some point during the search, none of the neighboring solutions are better than the current solution, the heuristic stops. Such a solution is locally optimal with respect to the move. Apart from "random restart" approach, some new strategies have been developed to cope with the problem of encountering local optimality during a search procedure. These include simulated annealing, genetic algorithms and tabu search (see for example (Kir83, Glo89a, Glo89b)), and present ways to incorporate a certain form of intelligence in the search process. The success of a tabu search approach when attacking the QAP (Sko90a, Sko90b, Tai91, Cha91) motivated the development of a tabu search heuristic specialized for the TSP.

Another ingredient of our approach stems from the recent surge in parallel computing. Recall that the 2-opt move has a neighborhood of size $O(n^2)$. In other words, from any given solution, $O(n^2)$ different solutions can be obtained by performing a 2-opt move. The 2-opt based tabu search algorithm evaluates all these different solutions and performs a move corresponding to the "best" solution in a restricted sense, which will be explained in the following sections. We propose a massively parallel implementation of the 2-opt move on the connection machine CM-2. A massively parallel tabu search algorithm for the QAP has been proposed by the authors (Cha91). It is possible to treat the TSP as a special case of the QAP, and use our QAP solving algorithm directly. However, the moves (pairwise exchange) which are extremely effective in solving the QAP do not perform well when applied to the TSP (Joh90). Therefore, we develop a tabu search heuristic based on 2-opt moves and provide an implementation on CM-2.

In the sequel we present the heuristic for the TSP, followed by a brief overview of the connection machine CM-2 along with implementation details. Finally, we present the computational results and our conclusions, along with outlines for possible future work.

A tabu search heuristic for the TSP

For the sake of completeness we briefly state the main elements of tabu search. A comprehensive description can be found in a two part paper by Glover (Glo89b,Glo90).

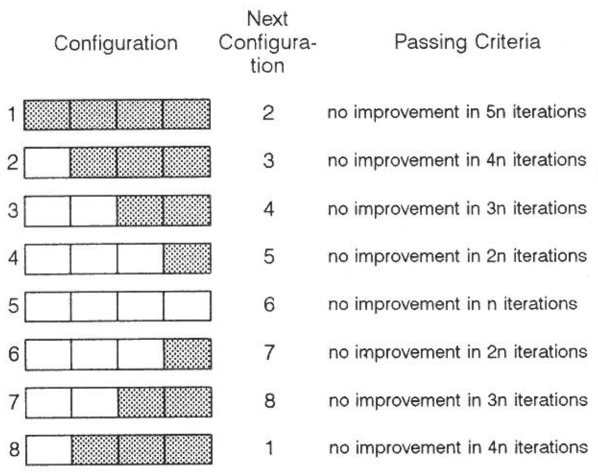
Tabu search is a meta-strategy that can be super imposed on any local optimization procedure. The basic idea of tabu search is to keep track of recently performed moves, and forbid their reversal for a specified number of iterations. When a local optimum is reached, the search will not stop. Instead, an inferior solution from the neighborhood will be taken as the current solution (say, the best in the neighborhood), and the search will proceed. Making 'the comeback' to a local minimum tabu, i.e. forbidden, is then crucial to avoid cycling or entrapment in the local minimum. Forbidding the reversal of recently performed moves serves this purpose. Continuing with this strategy one intuitively assumes that the probability of falling back to a local minimum diminishes as the search moves away from it. Therefore, after some iterations, older tabu restrictions could be released. It is clear that with this strategy the search will encounter a number of different local minima, and therefore will possibly improve the incumbent. The usual organization of the tabu restriction is via a *tabu list* of a certain length (*tabu list size*), fixed or variable, which is updated circularly. This is also called *short term memory* of the search. If, during the search, there is an indication (via evaluation of the objective function) that better solutions might be found in a partially explored "chunk" of the feasible region, some form of intensified search should be applied. This is the *intermediate memory* of the search. Finally, one would want to diversify the search by preventing (via penalties) the re-examination of a subset of the feasible region. This is the *long term memory* of the search. During the search, the more frequently one diversifies the broader would be the search and one can expect to have "visited" a large region of the search space. Alternately, in the same time span, the less frequently one diversifies the deeper would be the search and one can expect to have searched the explored region(s) of the search space more "thoroughly". Time limitations and solution quality requirements often dictate the levels of intensification and diversification incorporated in a search strategy.

Our tabu search heuristic for the TSP is similar to the one developed for the QAP (Cha91). The parameters of the heuristic that is described below, have been set after empirical evaluation of different settings based on the size of the problem. It is quite impossible to obtain an unique parameter setting that gives the best results for all problems. A different parameter setting may provide better results for specific problems. However, over a set of problems, slight variations in the parameter settings do not affect the quality of the results significantly.

The search starts from a random point and continues until there is no improvement in 25*n* iterations, where *n* is the size of the problem. It then enters a loop where intensification is achieved by repeatedly returning to the best solution found with an empty tabu list. The intensification loop is exited if 25*n* iterations find no better solution, and a frequency based long term memory is invoked providing the diversification. This represents a balance between intensification by pursuing it if better solutions are found, and diversifying otherwise.

Diversification is performed for a fixed, 10*n*, number of iterations. It employs a frequency based long term memory with a threshold of 5 % as explained in the following. Let ltm_{ij} store the number of times the edge (i,j) was in the traveling salesman tours visited by the tabu search procedure so far. The search is forced to explore new regions by imposing a tabu restriction, unconditionally for 10*n* iterations, on all the edges that were present in more than 5 % of the tours examined so far. After 10*n* iterations have been performed, the tabu restrictions based on the frequency of edge occurrence are dropped.

Every time a 2-opt move is performed, the two edges that are deleted from the tour are added to the tabu list. This forbids the addition of these edges when generating a new tour for a certain number of iterations (based on the tabu list size). Since the tabu list is maintained cyclically, the tabu status of these edges are dropped after exactly the tabu list size number of iterations. Our implementation of the tabu list is as follows. Let t_iter_{ij} be the last iteration when the edge (i,j) was deleted from a tour. In any single 2-opt move two edges are added and two edges are deleted. If the difference between the current iteration and any of the (two) *t_iter* values of the edges that would be added is less than the current tabu list size, then the corresponding 2-opt move is classified as tabu. Throughout all phases of the search, the tabu list size is varied dynamically by cycling through eight configurations of the tabu list size (see Figure 1). In the figure, the shaded



	Configuration	Next Configuration	Passing Criteria
1	▨▨▨▨	2	no improvement in 5n iterations
2	□▨▨▨	3	no improvement in 4n iterations
3	□□▨▨	4	no improvement in 3n iterations
4	□□□▨	5	no improvement in 2n iterations
5	□□□□	6	no improvement in n iterations
6	□□□▨	7	no improvement in 2n iterations
7	□□▨▨	8	no improvement in 3n iterations
8	□▨▨▨	1	no improvement in 4n iterations

Figure 1: Tabu list configuration

area indicates the active region of the tabu list in each configuration, and the transition from one configuration to the next is given as the passing criteria. For example, if the tabu list is in configuration 1 and there has been no improvement of the current best solution in 5*n* iterations, the tabu list changes to the next configuration (configuration 2). In configuration 2, the new tabu list size is 3/4th the original tabu list size. Every time the tabu list starts at configuration 1, a new tabu list size is selected as a multiple of 4 in the interval [*base_tabu_size*, *base_tabu_size*+12]. Dynamic tabu list management has been studied before by Glover and Hübscher (Glo91a), Taillard (Tai91), and Skorin- Kapov (Sko90b). The *base_tabu_size* along with the required number of diversifications to be performed are given as inputs to the algorithm. The complete algorithm follows.

Inputs: *D*-**distance matrix**, *base_tabu_size*, *no_of_restarts*

```
0)  generate starting solution
    clear tabu list, and go to step 1
1)  evaluate all moves
```

```
     determine the best move subject to tabu
     restrictions.
     perform the best move
     update tabu list
     if no improvement in 25n iterations
          go to step 2a
     else
          go to step 1
2a)  go to the best solution found so far
     clear the tabu list
2b)  evaluate all moves
     determine the best move subject to tabu
     restrictions
     perform the best move
     update tabu list
     if no improvement in 25n iterations
          if current step 2b has found a better
          solution
               go to step 2a
          else
               if required number of restarts have
               been performed
                    stop
               else
                    go to step 3a
     else
          go to step 2b
3a)  invoke long term memory
3b)  evaluate all moves
     determine the best move subject to tabu
     restrictions
     perform the best move
     update tabu list
     if 10n iterations performed
          initialize best solution to the current
          solution
          revoke long term memory
          update number of restarts and go to
          step 1
     else
          go to step 3b
```

Parallel Implementation

The 2-opt move has a neighborhood of size $O(n^2)$. We provide an implementation of 2-opt on the connection machine CM-2. With a massively parallel machine like the connection machine, all the 2-opt moves for a single iteration of tabu search can be evaluated in parallel. In the context of tabu search, parallelism is achieved in the exploration of the neighborhood.

The Connection Machine system (Hil85) is a fine grain, SIMD, computing system. A CM-2 model may contain 16*K*, 32*K*, or 64*K* processors. Each processor has, associated with it, its own memory which could either be 8*K* or 32*K* bytes (Thi89). Processors can access their own memory directly and the memory of other processors through hardware supported parallel communication. The processors can be configured in any way to suit the application. When executing a parallel instruction, each processor acts on the data stored in its own memory and for this reason the parallelism achieved on CM-2 is also known as *data level parallelism*. The CM-2 system also supports virtual processing. Each processor (along with its local memory) can be sliced into many units providing the user, virtually, with more than the physical number of processors. For example, slicing each processor unit into two provides, virtually, double the number of physical processors (each with half the memory of a real processor). Though in the above example a single processor would sequentially execute code on each half of the memory, programming can be done at a higher level of abstraction assuming the availability of twice the actual number of processors. The number of virtual processors available is limited only by the memory requirements specific to the problem. The ratio of the number of virtual processors to the number of physical processors is called *VP ratio*.

Inter-processor communication operations can be classified either as *send* or *get* operations. If a "sending processor" knows the address of the receiving processor, the *send* operation can be used. The "receiving processor" need not know where the data comes from. The converse is true for the *get* operation. The most general purpose communication, where any processor can communicate with any other, is supported by a high speed router. The CM-2 system also supports a faster communication mechanism called NEWS grid. NEWS grid communication is a structured form of communication which requires the processors to be organized as a multi (up to 31) dimensional grid. NEWS grid effects communication between neighboring processors relative to the specified grid. NEWS grid also supports functions like *scan*, *spread*, etc., which combine communication with computation. For example, the *spread* function can compute based on the data from all processors along a particular dimension of the grid and communicate the results to the same. In general, NEWS grid communication is more efficient than general communication and, send operations are faster than get operations (Thi 90).

In addition to the general form of inter-processor communication mentioned above, there are some special forms of communication operations. These include the *broadcast* operation where a single information is to be communicated to all the processors, and its inverse – the *reduction* operation where information from all the processors is used to produce a single information. Broadcast and reductions operations have separate, highly efficient, hardware implementations.

An efficient implementation of an algorithm on a machine such as the CM-2, basically involves decomposing the algorithm into small, independent steps that can be performed simultaneously. Ideally, if each step of the algorithm can be broken into several small and completely independent steps, parallelization is simple. In reality, however, such a decomposition is not possible except for trivial problems. There would exist some inter-dependency where results from the small steps have to be put together causing the necessity for inter-processor communication. Thus, the main step in the implementation is an efficient decomposition of the problem to maximize processor utilization and minimize communication time.

Consider a tour to be a permutation π where $\pi(i)$ is the ith visited city. A 2-opt move between $\pi(i)$ and $\pi(j)$, $\pi(i)<\pi(j)$ would result in a tour represented by a permutation obtained by reversing the entries of p between i and j inclusive. In other words, the result of the 2-opt move is the tour $\pi(1)\ \pi(2)...\pi(i-1)\ \pi(j)\ \pi(j-1)...\pi(i+1)\ \pi(i)\ \pi(j+ +1)...\pi(n-1)\pi(n)$. The two edges deleted are the edges between cities $\pi(i-1)$ and $\pi(i)$ and $\pi(j)$ and $\pi(j+1)$ and the two edges added are between cities $\pi(i)$ and $\pi(j+1)$ and $\pi(i-1)$ and $\pi(j)$.

Let the processors of the CM-2 be organized as a $n \times n$ matrix. Each processor p_{ij} stores the following information: the left and right neighbors of city *i*–*left*(*i*), *right*(*i*) and the distances; the position of cities *i* and *j* in the current tour–*pos*(*i*), *pos*(*j*) ($\pi^{-1}(i)$ and $\pi^{-1}(j)$); and d_{ij}. The value of a 2-opt move where the swap points are cities *i* and *j* are computed jointly by processors p_{ij} and p_{ji}. The decrease in tour length due to the deleted edges can be computed locally since the processors have that information stored in their memory. The increase in tour length due to the added edges is communicated to the processors. The algorithm to communicate and compute follows.

```
communication step
pos(i)<pos(j)
        edge i-j added during swapping of
        right(i) and j
else
        edge i-j added during swapping of
        left(i) and j
computation step
pos(i)<pos(j)
        edge left(i)-i deleted during swapping
        of i and j
else
        edge i-right(i) deleted during swapping
        of i and j
```

Now, processor p_{ij} has enough information to compute the partial value of the 2-opt move involving the edge deleted from and the edge added to city *i*. Similarly, p_{ji} has the value of the edge deleted from and the edge added to city *j*. In another communication step processors p_{ij} and p_{ji} exchange their values to compute the complete effect of their 2- opt move. Figure 2 gives an example of a 2-opt move evaluation jointly by two processors, with the final communication step completing the evaluation shown by dotted lines. Tabu status of a move is determined from the last iteration in which the edge *i-j* was deleted and is stored in processor p_{ij}. The edges that were deleted in the last *tabu_size* (parameter describing the tabu list size) iterations have a tabu status of 1 (they are in the tabu list), and the other edges have a tabu status 0 (they are not in the tabu list). The edges that have a tabu status of 1 must be forbidden from entering the tour. Since tabu status is a 0-1 variable, it can be padded and communicated along with the edge distance. Thus, the processors evaluating a move would know the tabu status of the edges that would be added if the move were to be performed.

Evaluation of 2-opt move with swap cities 2 and 4
Starting tour - 1 2 3 4 5
Processor 2,4 - added edge 1-4 - communicated
- deleted edge 1-2 - local memory
Processor 4,2 - added edge 5-2 - communicated
- deleted edge 4-5 - local memory

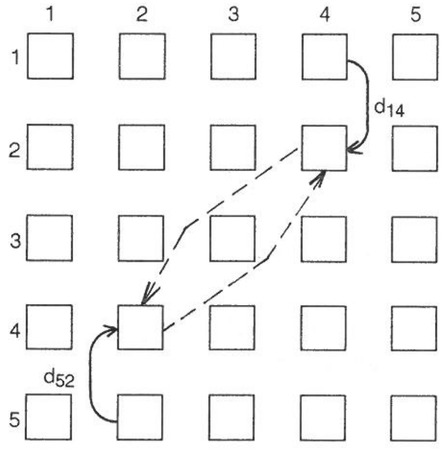


Figure 2: Example of a 2-opt evaluation on CM-2

Note that now the processors collectively have the values of all the moves, along with their corresponding tabu statuses. Determining the best move to be performed involves finding the processor that has the maximum move value subject to tabu restrictions. This is done very efficiently using a reduction operation.

After the best 2-opt move subject to tabu restrictions is made, the information in some of the processors has to be updated. Suppose that the 2-opt move performed be between cities i and $j, i<j$. Each processor p_{kl} updates the information stored depending on the positions of the cities k and l. The algorithm for the updates follows.

```
$pos(k)>pos(i)$ and $pos(k)<pos(j)$
        swap left and right neighbors of
        city k and their distances
        $pos(k) \leftarrow pos(i)+pos(j)-pos(k)$
$pos(l)>pos(i)$ and $pos(l)<pos(j)$
        $pos(l) \leftarrow pos(i)+pos(j)-pos(l)$
$k,l=i,j,i-1,j+1$
        explicitly set positions and/or distances
```

Note that the neighborhood of a 2-opt move has size of only $n(n-3)/2$. The configuration described so far uses more than twice the number of processors to evaluate all moves. It is possible to organize the processors in a one-dimensional configuration using about half the number of processors. Some additional storage and computation time are required as now one processor does the work of two from the previous configuration. Though the 1D configuration is slightly slower, it increases the size of the problem than can be solved without virtual processing. Both configurations were implemented and the 1D implementation was used to solve larger problems. Further details can be found in the next section. Recall, the amount of virtual processing possible depends only on memory restrictions. In both of our implementations we use a constant amount of memory per processor and achieve communication using only *send* operations. Thus, our implementations are efficient and highly scalable.

Computational Results

The algorithm was tested on a set of symmetric traveling salesman problems from the literature, ranging in size from 100 to 1002 nodes. The algorithm is general in the sense it does not require the euclidean TSP instance, i.e., it does not explore the geometric structure of the problem. All the problems appear in the traveling salesman problem library (TSPLIB), which is available to researchers. For information about the access to the library, the reader is referred to (Rei92). For all problems we use, except D493, the optimal solution is known.

The computations were performed on CM-2s with 16K processors located at NPAC Syracuse, UMIACS College Park and Thinking Machines Corporation. Coding was done in C*. Problems of size larger than 150 were solved using the 1D configuration of the processors. Without virtual processing, the 2D implementation takes 6 seconds and a 1D implementation takes 8 seconds to perform 1000 iterations of tabu search. In both implementations, every time the *VP ratio* doubles, the time/iteration gets multiplied by a factor of 1.6. When organizing the processors as a multi dimensional grid, the CM-2 system requires that the number of processors in each dimension be a power of 2. With 16K processors, 128 is size of the largest TSP that can be solved without virtual processing. TSPs of size greater than 128 but less than 256 would require a 256 × 256 grid organization of the processors–a *VP ratio* of 4. Alternately, our 1D configuration uses only about half the number of processors and problems of size up to 180 can be solved without virtual processing. From then on the *VP ratio* doubles as the size of the problem gets multiplied by $\sqrt{2}$. Therefore, the 1D configuration is preferrable when solving problems larger than 128.

A maximum of five restarts were tried for all but the three largest problems. For all problems the base tabu size was fixed at $2n/3$ rounded down to the nearest multiple of four. The maximum time spent on each problem was less than 10 minutes. The results are given in Table 1. The best tours produced by the tabu search algorithm are compared against the previously best known tours (which are optimal for all problems except D493). The table also provides results from performing 2-opt based local search with 10 random starting tours for all but the three largest problems tested. From the table it is clear that the use of tabu search to escape from local optima is superior to local optimization with random restart.

Table 1: Results for TSP

Problem	Pct. above best known		
	after 5 restarts of tabu search	after 1 restart of tabu search	after 10 restarts of 2-opt
KROA100	0.000	0.000	4.474
KROB100	0.000	0.000	2.245
KROC100	0.000	0.011	0.800
KROD100	0.000	0.000	3.593
KROE100	0.000	0.000	4.500
LIN105	0.000	0.007	0.765
KROA150	0.003	0.760	3.815
KROB150	0.120	0.333	4.474
KROA200	0.370	0.601	5.319
KROB200	0.230	1.036	4.328
D198	0.250	0.520	1.787
GIL262	0.500	0.672	4.584
LIN318	0.800	1.040	4.580
D493		4.700	
ATT532		2.100	
PR1002		2.650	

The solutions obtained for the set of TSPs from the literature are encouraging: for relatively smaller problems (size 100 and 105), the optimal solutions have been obtained. For problems up to size 318, the heuristic solution is less than 1 % inferior to the optimal. For other problems, the heuristic solution is also sufficiently close to the best known.

Our intention in performing this computational study was to develop an efficient implementation of 2-opt on the connection machine. We were aware that by setting a relatively small time limit for problems of such complexities, it is possible that the optimal solution may not be reached. Performing more restarts will probably improve the results even further (restarting improved the results in all cases to which it was applied and the initial start did not reach optimality). Tabu search techniques have proved very effective for attacking a number of hard combinatorial optimization problems. When adapting tabu search to a new class of problems there usually is a "preprocessing" stage where the strategy is fine tuned and the search parameters are set. Our implementation provides a platform for testing various strategies and adding new insights to the tabu search approach.

Conclusions

A heuristic algorithm utilizing 2-opt moves within a framework of the tabu search approach is developed for the traveling salesman problem. Its implementation on the massively parallel architecture of the connection machine model CM-2 is proposed. The implementation is geared towards efficiently utilizing the processors, and mimimizing the communication among them. This is achieved by a careful decomposition of the problem.

The algorithm is very effective for smaller TSPs in the sense that it reached optimal solutions. For larger problems more effective tabu search based strategies might possibly be developed. Our implementation could serve as a valuable tool in the development. The near doubling of time with *VP ratio* nearly offsets the speedup for large problems. We have provided a general purpose implementation that examines the whole neighborhood at each iteration. A tabu search strategy that always examines the whole neighborhood would not be effecient when solving very large TSPs. Algorithms that examine restricted neighborhoods may be implemented using lesser processors improving the time/iteration.

Acknowledgements

This research was partially supported by NSF grant DDM-8909206. The computations were conducted using the computing resources of the

North-East Parallel Architectures Center (NPAC) at Syracuse University, Syracuse, NY; UMIACS College Park, Maryland; and Thinking Machines Corporation, Cambridge, Massachusetts.

References

(Ben90) J.L. Bentley. Experiments on geometric traveling salesman heuristics. Technical Report Computing Science Technical Report No. 151, AT&T Bell Laboratories, 1990.

(Cha91) J. Chakrapani and J. Skorin-Kapov. Massively parallel tabu search for the quadratic assignment problem. To appear in *Annals of Operations Research.*

(Glo89a) F. Glover and H.J. Greenberg. New approaches for heuristic search: A bilateral linkage with artificial intelligence. *European Journal of Operations Research*, 39(2):119-130, 1989.

(Glo89b) F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190-206, 1989.

(Glo90) F. Glover. Tabu search – part II. *ORSA Journal on Computing*, 2(1):4-32, 1990.

(Glo91) F. Glover and R. Hübscher. Bin packing with tabu search. Technical report, Center for Applied Artificial Intelligence, University of Colorado, Boulder, 1991.

(Hil85) W.D. Hillis. *The Connection Machine.* The MIT Press, 1985.

(Joh90) D.S. Johnson. Local optimization and the traveling salesman problem. In *Proceedings of the Seventeenth Colloquium on Automata, Languages and Programming*, pages 446-461. Springer-Verlag, 1990.

(Kir83) S. Kirkpatrick, C.D. Gellati, and M.P. Vecchi. Optimizing by simulated annealing, *Science*, 220:671-680, 1983.

(Pad91) M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60-200, 1991.

(Rei92) G. Reinelt. Fast heuristics for large geometric traveling salesman problems. *ORSA Journal on Computing*, 4(2):206-217, 1992.

(Sko90a) J. Skorin-Kapov. Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing*, 2(1):33-45, 1990.

(Sko90b) J. Skorin-Kapov. Extensions of a tabu search adaptation to the quadratic assignment problem. To appear in *Computer and Operations Research.*

(Tai91) E. Taillard. Robust tabu search for the quadratic assignment problem. Parallel Computing, 17:443-455, 1991.

(Thi89) Thinking Machines Corporation, Cambridge Massachussetts. *Connection Machine Model CM-2, Technical Summary Version 5.1, May 1989.*

(Thi90) Thinking Machines Corporation, Cambridge Massachussetts. C* Programming Guide, Version 6.0, November 1990.

Received: September 22, 1992
Accepted: April 7, 1992

Address for correspondence:
Jadranka Skorin-Kapov
Harriman School for Management and Policy
State University of New York at Stony Brook
Stony Brook, NY 11794
e-mail: jskorin@ccvm.sunysb.edu

Jaishankar Chakrapani received his Ph.D. in the Department of Applied Mathematics, State University of New York at Stony Brook. He received his B.S. in Applied Sciences from Bharathiar University, India and his B.S. in Computer Science and Engineering from the Indian Institute of Science. His research interests include heuristics for combinatorial optimization, tabu search, parallel algorithms, computational geometry and neural networks. He has published in Computers and Operations Research and Annals of Operations Research.

Jadranka Skorin-Kapov is an Associate Professor in the Harriman School for Management and Policy, State University of New York at Stony Brook. She received her B.Sc. and M.Sc. in Applied Mathematics from the University of Zagreb, Croatia and her Ph.D. in Operations Research from the University of British Columbia, Canada. Her research interests are in the area of combinatorial optimization. She has published in Mathematical Programming, Operations Research Letters, ORSA Journal on Computing, Computers and Operations Research, Discrete Applied Mathematics, European Journal of Operational Research, and Annals of Operations Research.