

Massively parallel tabu search for the quadratic assignment problem

Jaishankar Chakrapani

*Department of Applied Mathematics and Statistics, State University of New York
at Stony Brook, Stony Brook, NY 11794, USA*

Jadranka Skorin-Kapov

*Harriman School for Management and Policy, State University of New York
at Stony Brook, Stony Brook, NY 11794, USA*

Abstract

A new heuristic algorithm to perform tabu search on the Quadratic Assignment Problem (QAP) is developed. A massively parallel implementation of the algorithm on the Connection Machine CM-2 is provided. The implementation uses n^2 processors, where n is the size of the problem. The elements of the algorithm, called *Par_tabu*, include dynamically changing tabu list sizes, aspiration criterion and long term memory. A new intensification strategy based on intermediate term memory is proposed and shown to be promising especially while solving large QAPs. The combination of all these elements gives a very efficient heuristic for the QAP: the best known or improved solutions are obtained in a significantly smaller number of iterations than in other comparative studies. Combined with the implementation on CM-2, this approach provides suboptimal solutions to QAPs of bigger dimensions in reasonable time.

1. Introduction

The Quadratic Assignment Problem (QAP) is a combinatorial optimization problem with applications to facility location [11], backboard wiring [15], scheduling [3], etc.

A QAP of size n is characterized by two $n \times n$ matrices $D = \{d_{ij}\}$ and $F = \{f_{pq}\}$. Denoting by N the set $\{1, 2, \dots, n\}$ and Π the set of all permutations of N , the problem can be defined as follows.

$$\min_{\pi \in \Pi} C(\pi) = \sum_{i=1}^n \sum_{j=1}^n d_{ij} f_{\pi(i)\pi(j)}.$$

The sequel assumes that both matrices D and F are symmetric with zeroes as main diagonal elements. The QAP is known to be NP-hard [12]. The problem can also be posed as a 0–1 optimization problem as follows.

$$\begin{aligned}
& \text{minimize} && \sum_{i=1}^n \sum_{k=1}^n \sum_{j=1}^n \sum_{l=1}^n d_{ij} f_{kl} x_{ik} x_{jl} \\
& \text{subject to} && \sum_{k=1}^n x_{ik} = 1 \quad \forall i, \\
& && \sum_{i=1}^n x_{ik} = 1 \quad \forall k, \\
& && x_{ik} = 0, 1 \quad \forall i \forall k.
\end{aligned}$$

Obviously, the feasible set is the set of $n \times n$ permutation matrices $X = (x_{ik})$, where x_{ik} equals one if $\pi(i) = k$, and zero otherwise. Let the "pairwise exchange" neighborhood of a given permutation π refer to the set of permutations that can be obtained by performing a pairwise exchange on π .

Local improvement heuristics proceed by examining some neighborhood of the current solution. Tabu search is a meta strategy superimposed on local improvement heuristics to cope with local optimality entrapment in solution processes. For a comprehensive description of tabu search the reader is referred to a two part paper by Glover [5,6]. Tabu search based algorithms have already been applied to the QAP [14,13,1,16]. The objectives of this paper are (1) to develop a new, efficient tabu search strategy for the QAP with special emphasis on applicability to larger problem sizes and (2) to provide a massively parallel implementation of this strategy.

With respect to the first objective an algorithm called *Par_tabu* is developed. *Par_tabu* makes use of various, well established as promising, elements of tabu search including aspiration, long term memory and varying tabu list sizes. Specific attention is given to intensive search in the proximity of "good" solutions. When dealing with large problems searching the whole neighborhood at every iteration could become computationally too demanding. A new intensification strategy, which searches in a neighborhood restricted using intermediate term memory, is proposed. This is then superimposed on *Par_tabu* resulting in a very effective strategy.

When a whole neighborhood is examined, a single iteration of tabu search involves evaluation of $O(n^2)$ pairwise exchanges and finding the best subject to tabu restrictions. The best pairwise exchange is then performed. A pairwise exchange is evaluated based on the change in the objective function value it effects if performed. Except for the first iteration each pairwise exchange, independent of the others, can be evaluated in constant time (see, e.g. [1]). Thus, with $O(n^2)$ processors a parallel algorithm can evaluate the whole neighborhood in constant time. The best pairwise exchange (subject to tabu restrictions) can be determined in $O(\log n)$ time (see, e.g. [10]). Based on the above the tabu search strategy developed in this paper is implemented on the Connection Machine CM-2 – a massively parallel SIMD machine. Due to the architecture of the machine, an efficient

implementation required a fine grain decomposition of the problem into small identical subproblems suitable for data-level parallel computing.

Previous work on some of the elements of tabu searching include the following. Skorin-Kapov [13] used restricted neighborhoods to speed up the computations while solving large QAPs. In her work, a neighborhood was restricted by fixing some entries in a permutation, i.e. forbidding some pairwise exchanges. This was done using the knowledge about good permutations encountered during the search process. In a broader context, approaches to intelligently restricting the search to good solution space areas via candidate lists were proposed by Glover [4]. With respect to varying tabu list sizes, Glover and Hübscher [7] have proposed a different strategy of dynamically varying tabu list sizes by introducing "moving gaps" in the tabu list. Skorin-Kapov [13] uses a similar strategy in her paper. In his work on QAPs, Taillard [16] varies the tabu list size randomly in an interval. He also develops a parallel implementation of his algorithm on a ring of 10 transputers.

The rest of the paper is organized as follows. Section 2 describes elements of *Par_tabu*. It also provides a formal description of the algorithm and a detailed discussion on the new intensification strategy proposed. In section 3, an overview of the connection machine is provided along with some implementation details. In section 4, the results of computations over a set of QAPs are presented. Conclusions are drawn in section 5.

2. *Par_tabu* – a parallel tabu search algorithm

Par_tabu was designed to perform effectively while keeping the number of iterations relatively small. It usus a preliminary phase which terminates quickly identifying the best solution found. An intensification phase then focuses the search around the current best solution found until no improvement is obtained for a fixed number of iterations. Long term memory is then invoked to diversify the search to unexplored regions. In section 2.1, a detailed explanation of these elements of *Par_tabu* is provided along with a formal description of the algorithm. Section 2.2 describes the new intensification strategy proposed on intermediate term memory.

2.1. ELEMENTS OF *PAR_TABU*

2.1.1. Basic strategy

Par_tabu is divided into four phases viz., initialization phase, preliminary search phase, intensification phase and diversification phase. The phases use various parameter settings which govern tabu list management, transition between phases and long and intermediate term memories. All the parameters were set empirically after testing various values on problems of sizes between 42 and 64.

The initialization phase consists of generating a random solution, and starting the preliminary search phase. Alternately, a construction algorithm could be used

to provide a starting solution. A random start was used to study the effectiveness of the algorithm independent of any construction algorithm.

Each iteration of the preliminary search phase evaluates all possible moves and determines the best valued move which either is non tabu or satisfies an aspiration criterion. Aspiration is satisfied if a move that is tabu results in a better solution than the best found so far. If no better solution is found in $25n$ iterations, the intensification phase is started.

The intensification phase starts, with an empty tabu list, from the best solution found in the current region. The algorithm proceeds as in the preliminary search phase. If a better solution is found, the intensification phase is restarted after $50n$ iterations are performed without any improvement. If however, the current intensification phase does not find a better solution, the diversification phase is initialized after $50n$ iterations. The intensification phase provides a simple way to focus the search around the current best solution.

Diversification is the only phase performed for a fixed, $10n$, number of iterations. It employs a frequency based long term memory with a threshold of 5% as explained in the following. Let $longmem_{ik}$ store the sum of the values of the 0–1 variable x_{ik} in all the iterations performed so far. Thus the $n \times n$ array $longmem$ stores the number of times each entry of a permutation took a value 1 in all the solutions examined by the search. If the most frequently used entries, based on a threshold, are kept tabu for a sufficient number of iterations, the search will be forced to explore new regions. The number of iterations with this additional tabu restriction should be large enough to drive the search out of the current region. After the specified number of iterations are performed, the tabu status based on frequency measure could be dropped. Alternatively, a new starting solution could be constructed based on the matrix $longmem$. The method described above allows the search process to select the new region to explore. After $10n$ iterations, the algorithm loops back to the preliminary search phase. The algorithm terminates after the search has been diversified to the required number of regions.

2.1.2. Tabu list management

Let in the current solution $x_{ik} = x_{jl} = 1$. A pairwise exchange between i and j would result in a solution where $x_{ik} = x_{jl} = 0$ and $x_{il} = x_{jk} = 1$. In general, in every pairwise exchange there are exactly two variables that change their values from 1 to 0 and two variables from 0 to 1. Par_tabu determines the tabu status of a pairwise exchange only based on the two variables that would change their values to 1. Let t_iter_{pq} be the last iteration when x_{pq} changed its value from 1 to 0. If the difference between the current iteration and any of the (two) t_iter values of the relevant variables is less than the tabu size, then the pairwise exchange is classified as tabu.

The tabu list size is varied dynamically evolving through different configurations. It cycles through eight configurations passing from one to the other if no improvement









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

Fig. 1. Tabu list configurations.

is encountered in a fixed number of iterations. Figure 1 illustrates the configurations, with the shaded area indicating the active region of the tabu list in each configuration.

This set of tabu list configurations allows more detailed examination of the feasible region by systematically decreasing the number of actual tabu moves via configurations 2, 3, 4, and 5. It then breaks possible cycles and diversifies the search by systematically increasing the number of tabu moves in the remaining configurations. In order to provide an additional security against possible cycling a random component is introduced in the tabu list. Every time the tabu list starts at configuration 1, a new tabu size is selected as a multiple of 4 in the interval $[base_tabu_size, base_tabu_size + 12]$. The *base_tabu_size* together with the required number of diversifications to be performed are given as inputs to the algorithm.

2.1.3. The complete algorithm

ALGORITHM *Par_tabu*

Inputs: $D, F, base_tabu_size, no_of_restarts$

step 0 Initialization: Generate starting solution, clear tabu list, and go to step 1.

step 1 Preliminary search:

- Evaluate all moves and determine the best subject to tabu restrictions.
- Perform best move.
- Update tabu list.
- - If no improvement in $25n$ iterations go to step 2.
- Else go to step 1.

step 2 Intensification:

- (a) Go to the best solution found so far and clear the tabu list.
- (b)
 - Evaluate all moves and determine the best subject to tabu restrictions.
 - Perform the best move.
 - Update tabu list.
 - - If no improvement in $50n$ iterations:
 - If current step 2 has found a better solution, go to step 2a.
 - Else:
 - If required number of restarts have been performed, stop.
 - Else, go to step 3.
 - Else, go to step 2b.

step 3 Diversification:

- (a) Determine additional tabu restrictions from long term memory (elaborated in section 3.2).
- (b)
 - Evaluate all moves and determine the best subject to tabu restrictions.
 - Perform the best move.
 - Update tabu list.
 - - If $10n$ iterations performed:
 - Initialize best solution to the current solution.
 - Remove additional tabu restrictions due to long term memory, update number of restarts and go to step 1.
 - Else, go to step 3b.

2.2. INTENSIFICATION BASED ON INTERMEDIATE TERM MEMORY

In a general context Glover [5,6] proposes the use of intermediate term memory in a way complementary to long term memory. A straightforward adaptation of the idea was tried initially. The entries whose frequencies fell below a threshold were held permanently tabu for a fixed number of iterations. Let such variables be denoted as "blocked variables" since they are blocked from assuming the value 1. Various threshold values and tabu list sizes were tried. However, the results were not very encouraging. Somehow, the search seemed to be too restrictive. A possible reason for the performance of this straightforward intensification is suggested in the following.

Let $intmem$ be the $n \times n$ matrix such that $intmem_{pq}$ is the sum of the values of the 0–1 variable x_{pq} in all the iterations so far in the current region of search. Note that $longmem$ is a cumulative sum over all the different regions explored while $intmem$ is a frequency measure pertaining to the current region of search only. Suppose that a large number of solutions explored in the current region have $\pi(p) = q$. This would result in a high value of $intmem_{pq}$ and x_{pq} would be a blocked variable. The objective of intensification is to search more elaborately the search space defined by variables that are not blocked. Suppose that x_{ik} , x_{il} and x_{jl} are not blocked variables and that x_{jk} is a blocked variable. It is possible that at the current iteration x_{ik} and x_{jl} have value 1. However, x_{jk} is blocked indirectly causing x_{il} to be blocked. This seems to defeat the purpose of intensification.

The solution space searched during intensification was enlarged in order to avoid implicit blockings such as explained above. The results of applying a threshold to the $intmem$ matrix is an $n \times n$ matrix of zeroes and ones. An entry is zero if it falls below the threshold and one otherwise. This matrix could also be thought of as an adjacency matrix of a directed graph G . Let $G' = (U', E')$ be a directed graph where $U' = \{i1, i2, j1, j2\}$ is the set of vertices and $E' = \{(i1, j1), (i1, j2)\}, (i2, j2)\}$ is the set of edges. Enlarging the solution space as described in the previous paragraph is equivalent to identifying directed subgraphs isomorphic to G' on the original graph G , and adding an edge corresponding to $(i2, j1)$. A naive algorithm which checks all possible combinations was used. The algorithm takes $O(n^3)$ time and with n^2 processors parallelizes easily to an $O(n)$ algorithm.

A threshold of 5% was set after preliminary experimentation with various values. This intensification phase was superimposed on the intensification phase of *Par_tabu*. First, the simple intensification of *Par_tabu* is performed until the "no improvement criterion" is satisfied. Then, instead of diversifying, the search space is reduced as explained above and the new intensification is performed. The algorithm for this new intensification phase is identical to the original one except that it operates on a reduced search space and that the base tabu size is reduced in proportion to the reduction in search space. In the sequel, *Par_tabu* combined with the intermediate term memory based intensification phase is referred to as *Augmented Par_tabu*.

3. Parallel implementation

The *Par_tabu* algorithm in the previous section requires a massively parallel machine. In section 3.1 one such system (the Connection Machine system CM-2) is described. Section 3.2 provides the implementation details of *Par_tabu* on a CM-2 system.

3.1. THE CONNECTION MACHINE MODEL CM-2

The Connection Machine system [8] is a dynamically reconfigurable, fine grain SIMD, computing system. A CM-2 model may contain 16K, 32K, or 64K processors [17]. Each processor has, associated with it, its own memory which could either be 8K or 32K bytes. The hardware is comprised of chips interconnected in a hypercube configuration. Each chip contains 16 processor/memory units and a router to handle communication.

The hardware unit is controlled by a front computer (VAX or Sun). The user does all the coding on the front end which also executes the sequential parts of the code. The parallel instructions are broadcast to the CM-2 hardware with the help of a microcontroller. Parallel code can be executed (simultaneously) on a preselected set of processors. The programming languages currently supported are CM-FORTRAN, C* and *lisp. Program variables are either sequential or parallel. Sequential variables reside in the front end's memory and can be broadcast to all processors of the Connection Machine if necessary. Parallel variable reside in the local memory of the CM-2 processors and can be accessed freely by the front end. For e.g. each element of a 16K array can be stored (as a parallel variable) in a separate processor of a 16K machine. When executing a parallel instruction, each processor acts on its own copy of the parallel variables involved. Furthermore, the CM-2 system supports virtual processing. Each processor/memory unit can be sliced into many units providing the user, virtually, with more than the physical number of processors. For e.g. slicing each processor/memory unit into two provides, virtually, double the number of physical processors. 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*.

In addition to computation, the CM-2 system can perform interprocessor communication in parallel. 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.

3.2. IMPLEMENTATION OF *PAR_TABU* ON CM-2

At each iteration, a move from the current permutation to one of its neighbors is made by performing the best pairwise exchange subject to tabu restrictions. Through a series of such moves the search can reach any arbitrary permutation. Therefore, a straightforward implementation associating one processor with a pairwise exchange would require at least one of the matrices *D* or *F* to be stored in all the processors. This is a severe limitation for large *n* necessitating a decomposition of the storage. Further, it is clear that the computations would involve indexing a parallel array with parallel variables. This operation, called *parallel right indexing*, is highly inefficient [18].

Recall that $x_{pq}, p, q = 1, 2, \dots, n$ are the 0–1 variables in the 0–1 formulation of a QAP. Consider the following configuration of the CM-2. The processors are organized as a two-dimensional $n \times n$ grid. All processors in row *p* store a parallel array corresponding to row *p* of the matrix *D*. Similarly, all processors in column *q* store the corresponding column *q* of the matrix *F*. Further, each processor u_{pq} stores the value of its corresponding 0–1 variable x_{pq} . Let $S = \{u_{pq} \mid p, q = 1, 2, \dots, n\}$ be the set of processors and let $S_1 = \{u_{pq} \mid x_{pq} = 1\}$. Let π be a permutation such that $\pi(i) = k$ and $\pi(j) = l$. This implies that $x_{ik} = x_{jl} = 1$ and $x_{il} = x_{jk} = 0$. A change in the values of x_{ik}, x_{jl}, x_{il} and x_{jk} corresponds to a pairwise exchange of $\pi(i)$ and $\pi(j)$. Define $\Delta\mathcal{C}_{ik}$ by the following.

$$\Delta\mathcal{C}_{ik} = \delta\mathcal{C}_{ik}(1 - 2x_{ik}),$$

where $\delta\mathcal{C}_{ik} = \sum_{u_{pq} \in S_1} 2d_{ip}f_{kq}$. Note that the local memory of u_{ik} contains all the data needed for this computation and the storage requirement is only $O(n)$. The effect due to a pairwise exchange between $\pi(i)$ and $\pi(j)$ is given by

$$\begin{aligned} \Delta\mathcal{C}_{ikjl} &= - \sum_{u_{pq} \in S_1 - \{u_{jl}\}} 2d_{ip}f_{kq} - \sum_{u_{pq} \in S_1 - \{u_{ik}\}} 2d_{jp}f_{lq} - 2d_{ij}f_{kl} \\ &+ \sum_{u_{pq} \in S_1} 2d_{ip}f_{lq} + \sum_{u_{pq} \in S_1} 2d_{jp}f_{kq} + 2d_{ij}f_{kl} \\ &= \Delta\mathcal{C}_{ik} + \Delta\mathcal{C}_{jl} + 2d_{ij}f_{kl} \\ &+ \Delta\mathcal{C}_{il} + \Delta\mathcal{C}_{jk} + 2d_{ij}f_{kl} \\ &= \Delta\mathcal{C}_{ik} + \Delta\mathcal{C}_{jl} + \Delta'\mathcal{C}_{il} + \Delta'\mathcal{C}_{jk}, \end{aligned} \tag{1}$$

where, $\forall u_{p\pi(q)} \in S - S_1$, $\Delta' \mathcal{C}_{p\pi(q)} = \Delta \mathcal{C}_{p\pi(q)} + 2d_{pq} f_{\pi(p)\pi(q)}$. The required information to evaluate a pairwise exchange is computed in parts by four different processors, two of which are in S_1 and the other two are in $S - S_1$. These computations can be done in constant time, except for the first iteration, by storing of the $\Delta \mathcal{C}$ s from the previous iteration [1, 16]. Now, the results have to be communicated to the processor which is to evaluate the pairwise exchange.

Communication is done in three steps. Processors along the diagonal do not take part in communication. In the first step, all processors in $S - S_1$ communicate the sum of their $\Delta' \mathcal{C}$ to a designated processor in their row. This is achieved by a *send* operation. Each processor $u_{p\pi(q)}$ sends its $\Delta' \mathcal{C}_{p\pi(q)}$ to processor u_{pq} . In the second step, all processors in S_1 communicate their $\Delta \mathcal{C}$ values to all the processors in their row. This is done with the help of the communication function *spread*. As a result of these two steps processor u_{ij} would have received $\Delta \mathcal{C}_{ik}$ and $\Delta' \mathcal{C}_{il}$ and processor u_{ji} would have received $\Delta \mathcal{C}_{jl}$ and $\Delta' \mathcal{C}_{jk}$. Processors u_{ij} and u_{ji} between themselves have all the information needed to compute $\Delta \mathcal{C}_{ikj}$. In general, processors u_{pq} and u_{qp} have the information needed to compute the effect of a pairwise exchange of $\pi(p)$ and $\pi(q)$. In the third step, all the information required to evaluate a pairwise exchange is communicated to a single processor. Each processor u_{pq} communicates with the processor u_{qp} . This involves, again, a *send* operation.

Tabu status is stored in each processor as the last iteration t_iter in which the value of the corresponding variable x_{pq} changed to 0. If the difference between the current iteration and t_iter is less than the tabu size, then the corresponding processor has a tabu status of 1. Thus, evaluating tabu status of a pairwise exchange requires information regarding the tabu status of the processors involved. If the data involved is integral this does not require extra communication steps. Observing that the information communicated by the processors in the above communication steps is a product of 2, the tabu status (which is either 0 or 1) can be simply added and decoded correspondingly at the receiving end.

4. Computational results

The computations were performed on CM-2s located at NPAC Syracuse, Sandia National Laboratories, UMIACS College Park and Thinking Machines Corporation. The front end was a Sun. Coding was done in C*. The CM-2 at Sandia National Laboratories has 64 bit floating point accelerators and all the timings reported were achieved on this machine.

Since the benefits of a massively parallel implementation increase with the dimension of the problem, only larger QAP problems of size between 42 and 100 were tested. The problems are available in literature [14, 13] and could also be obtained, by request, from the authors.

Two base tabu sizes, both multiples of four, were tried for all the problems. The first base tabu size was chosen to be the nearest multiple of 4 less than $2n/3$.

Such a tabu size was chosen after preliminary experimentation on tabu sizes in the interval $[\frac{n}{2}, n]$ – the same as the one tried in a previous study by the authors [1] using the same tabu conditions. A multiple of 4 was chosen since the tabu list was divided into four parts to be employed in different configurations. The second base tabu size was obtained by adding 4 (for problems of size 42–90) or 8 (for problems of size 100) to the first base tabu size. For all problems, nine restarts based on long term memory were performed. In order to provide a fair basis for comparison both *Par_tabu* and *Augmented Par_tabu* were tested on the same set of parameters. Same tabu sizes and starting solutions of *Par_tabu* were provided to *Augmented Par_tabu*. During the intermediate term memory intensification phase of *Augmented Par_tabu* the base tabu size used was half the original base tabu size. This was in accordance with the reduction in the search space since, on the average, about half the values of every entry of a permutation were tabu in this phase.

Figure 2 shows the growth in time/iteration with problem size of *Par_tabu* when the required number of processors are available. 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. For problems of size up to 64, 8K processors suffice. 16K processors are required for problems of size greater than 64 and up to 128. If, however, only 8K processors are available, virtual processing with a *VP ratio* of 2 is required and the time taken gets multiplied by a factor of about 1.6. *Par_tabu* spends about 55% of its time/iteration in communication.

Even when the whole neighborhood is to be searched there are only $n(n - 1)/2$ pairwise exchanges to be evaluated. The implementation developed in

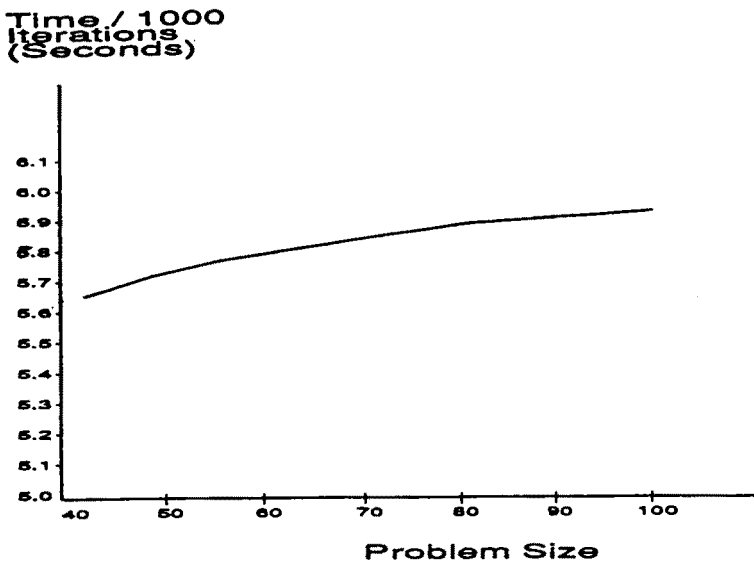


Fig. 2. Time versus problem size for *Par_tabu*.

this study uses n^2 processors where two processors, in parts, compute the change in the objective function value due to a single pairwise exchange. Once the results are communicated, about half the processors are inactive while the best pairwise exchange is being evaluated and performed. An implementation using only $n(n-1)/2$ processors would increase the size of the problem that can be solved without virtual processing. Also, during the intermediate term memory intensification phase of *Augmented Par_tabu* more than half the processors permanently have a tabu status of 1. Thus, there are at most $n^2/8$ pairwise exchanges to be evaluated in this phase. A more careful implementation of *Augmented Par_tabu* using only as many processors as needed could reduce the computational time spent in this phase by reducing the amount of virtual processing required.

Table 1 compares the average time/iteration of *Par_tabu* with three other tabu search algorithms from literature, applied to the same data set. It should be noted that although these are different algorithms, in each case an iteration involves examination of the "pairwise exchange" neighborhood (which is the most expensive part with respect to time). The purpose of this table is to indicate the difference in the growth in time/iteration as a function of problem dimension on different computer architectures. Such a comparison is necessary since, to the knowledge of the authors, this is the first study on the use of the Connection Machines in this context. Clearly, the effective use of CM-2 compares favorably thereby supporting its exploration as a computer medium for combinatorial optimization. The times reported for *Par_tabu* are the Connection Machine times which account for more than 90% of the computational time. There was little difference in the average time/iteration between *Par_tabu* and *Augmented Par_tabu*. This since the only extra step that *Augmented*

Table 1

Growth of time/iteration with problem size for different algorithms.

Prob. size	Time for 1000 iterations			
	<i>Par_tabu</i>	TAI ^a	SKO ^b	SEQ ^c
42	5.62	10.94	19.00	57.80
49	5.71	14.89	23.00	189.00
56	5.77	19.44	30.00	244.00
64	5.80	25.40	40.00	321.00
72	5.83	32.14	60.00	400.00
81	5.85	40.68	80.00	529.00
90	5.86	50.22	110.00	662.00
100	5.87	—	—	—

^aTAI : Taillard's [16] parallel algorithm implemented on a ring of 10 transputers.

^bSKO : Skorin-Kapov's [13] vectorized algorithm on IBM 3090.

^cSEQ : A sequential algorithm by the authors [1] on SUN SPARC station 1.

Par_tabu performed, evaluating the restricted neighborhood, takes time negligible compared to the time taken by the rest of the strategy.

Table 2 gives the results achieved by *Par_tabu* and *Augmented Par_tabu*. The best known objective function value is taken from (a) [13, 16], (b) [13], (c) [9] and (d) [2]. The best known or improved results obtained in this study are shown in boldface. *Par_tabu* matches the best known results for problems of size up to 64 and produces results better than those given in (a), (b), (c) and (d) for all the 100 size problems. It also achieves very good results for problems of size 72, 81 and 90. *Augmented Par_tabu* for all the problems provides results as good as or better than *Par_tabu*. It obtains the best known results for 72 and 81 size problems.

Table 2
Results for QAP.

Prob. size	Base tabu size	Best known solution	<i>Par_tabu</i>		<i>Aug Par_tabu</i>	
			Best solution achieved	Total no. of iterations	Best solution achieved	Total no. of iterations
42	24	14,812 ^a	15,812	57,691	15,812	91,136
	28		15,812	61,318	15,812	89,432
49	28	23,386 ^a	23,386	71,801	23,386	112,810
	32		23,386	81,191	23,386	131,187
56	36	34,458 ^a	34,458	105,345	34462	136,901
	40		34,476	106,269	34,458	175,898
64	40	48,498 ^a	48,502	116,878	48,498	145,056
	44		48,498	121,489	48,502	214,262
72	48	66,256 ^a	66,260	150,188	66,256	198,129
	52		66,330	112,220	66,302	174,093
81	52	91,008 ^b	91,010	189,097	91,008	191,571
	56		91,116	142,618	91,028	246,182
90	60	115,534	115,692	142,095	115,586	268,416
	64		115,594	230,449	115,608	259,372
100a	64	152,096 ^c	152,014	224,296	152,076	330,702
	72		152,112	246,320	152,014	199,882
100b	64	154,102 ^d	153,910	182,675	153,900	274,480
	72		153,912	176,145	153,908	303,303
100c	64	147,894 ^c	147,888	165,384	147,878	290,104
	72		147,878	242,397	147,868	306,954
100d	64	149,886 ^c	149,894	233,370	149,596	257,855
	72		149,682	187,520	149,698	304,012
100e	64	149,772 ^c	149,160	209,892	149,164	270,573
	72		149,164	245,138	149,156	311,458
100f	64	149,176 ^c	149,206	189,839	149,348	224,461
	72		149,048	192,176	149,036	308,587

Also, it improves the best known results found by *Par_tabu* for five of the six 100 size problems. The best results reported by Skorin-Kapov [13] were achieved performing an extensive search over a large number (over 1 million) of iterations. Kelly et al. [9] also perform over 1 million iterations to achieve their results. Taillard, in a private communication, reports the best iteration for the size 90 problem to be between 1.5 million and 2 million iterations. Comparatively, the *maximum* number of iterations per base tabu size performed by *Par_tabu* and *Augmented Par_tabu* for any problem are less than 250,000 and 350,000 respectively. Also, the algorithms require no extra guidance from the user with the only inputs being the *base_tabu-size* and the number of restarts to be performed. *Augmented Par_tabu* performs more iterations than *Par_tabu*. However, it also achieves better results demonstrating the effectiveness of reduced neighborhood search. Finally, although both base tabu sizes obtain good results, it is not possible to determine which one is better. However, the total number of iterations is still relatively small even when both base tabu sizes are taken into account.

5. Conclusions

A massively parallel tabu search based heuristic has been developed for the Quadratic Assignment Problem. The tabu search strategy proved to be very efficient in terms of solution quality. It obtained best known or close to best known solutions for problems of size up to 90, and improved upon known solutions for problems of size 100. A new intensification strategy based on the intermediate term memory is proposed. The addition of this intensification feature of the tabu strategy resulted in further improved solutions to larger problems, with the price of an increase in the total number of iterations performed. Since the intensification is based on restricting neighborhoods, a single iteration involves less effort than an iteration examining the whole neighborhood. This leads to possibilities of efficiently attacking even larger sized QAPs.

A careful implementation on the Connection Machine, a massively parallel system, proved to be extremely suitable. The increase in time per iteration is apparently a logarithmic function of the size of the problem. Possible directions are provided towards alternate implementations that could be more efficient when solving very large sized QAPs.

Acknowledgements

This research was partially supported by NSF grant DDM-8909206. This work was conducted using the computing resources of the North-East Parallel Architectures Center (NPAC) at Syracuse University, Syracuse, NY; Sandia National Laboratories, Albuquerque, NM; UMIACS College Park, MD; and Thinking Machines Corporation, Cambridge, MA.

References

- [1] J. Chakrapani and J. Skorin-Kapov, A connectionist approach to the quadratic assignment problem, *J. Comp. Oper. Res.* 19(1992)287–295.
- [2] B. Gavish, Manifold search techniques applied to the quadratic assignment problem, Technical report, Owen Graduate School of Management, Vanderbilt University (1991).
- [3] A. Geoffrion and G. Graves, Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment/linear programming approach, *Oper. Res.* 24(1976)595–610.
- [4] F. Glover, Candidate list strategies and tabu search, Technical report, Center for Applied Artificial Intelligence, University of Colorado, Boulder (1989).
- [5] F. Glover, Tabu search. Part I, *ORSA J. Comput.* 1(1989)190–206.
- [6] F. Glover, Tabu search. Part II, *ORSA J. Comput.* 2(1990)4–32.
- [7] F. Glover and R. Hübscher, Bin packing with tabu search, Technical report, Center for Applied Artificial Intelligence, University of Colorado, Boulder (1991).
- [8] W.D. Hillis, *The Connection Machine* (The MIT Press, 1985).
- [9] J. Kelly, M. Laguna and F. Glover, A study of diversification strategies for the quadratic assignment problem, Technical report, Graduate School of Business and Administration, University of Colorado, Boulder (1991).
- [10] U. Manber, *Introduction to Algorithms* (Addison–Wesley, 1989).
- [11] C. Nugent, T. Volmann and J. Ruml, An experimental comparison of techniques for the assignment of facilities to locations, *Oper. Res.* 16(1968)150–173.
- [12] S. Sahni and T. Gonzalez, P-complete approximation problems, *J. ACM* 23(1976)555–565.
- [13] J. Skorin-Kapov, Extensions of a tabu search adaptation to the quadratic assignment problem, *J. Comp. Oper. Res.*, to appear.
- [14] J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem, *ORSA J. Comput.* 2(1990)33–45.
- [15] L. Steinberg, The backboard wiring problem: A placement algorithm, *SIAM Rev.* 3(1961)37–50.
- [16] E. Taillard, Robust tabu search for the quadratic assignment problem, *Parallel Comput.* 17(1991)443–455.
- [17] Thinking Machines Corporation, Cambridge, MA, Connection Machine Model CM-2, Technical Summary Version 5.1. (May 1989).
- [18] Thinking Machines Corporation, Cambridge, MA, C* Programming Guide, Version 6.0. (November 1990).