RT
229

A Tabu Search Algorithm for Mapping Data to Multicomputers

By
Kamal M. Diab

June 1997

A Tabu search algorithm for mapping data to multicomputers

By
Kamal M. Diab
B.Sc., Lebanese American University

PROJECT

Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science
at the Lebanese American University
June 1997

Signatures Redacted

Dr. Nashat Mansour (Advisor)
Assistant Professor of Computer Science
Lebanese American University

Signatures Redacted

Dr. Issam Moughrabi
Assistant Professor of Computer Science
Lebanese American University

Signatures Redacted

Dr. Ramzi Haraty
Assistant Professor of Computer Science
Lebanese American University

Abstract

We present a Tabu Search algorithm for mapping data to multicomputer nodes, assuming certain computation and multicomputer models. The experimental results show that the Tabu algorithm offers both reasonable solution quality and mapping time, which ranks about fourth in comparison with five other algorithms.

To my father

Acknowledgements

First of all, I would like to thank Professor Nashat Mansour, my advisor, for all the help, encouragement, and support he provided me. Also, I would like to thank my second reader Professor Issam Moughrabi, and my third reader Professor Ramzi Haraty. I greatly thank the chairman of Natural Science Division at LAU, Dr. Ahmad Kabbani, and the computer manager in the Creditcard Services Company, Mr. Omar Bader, for the support they offered.

Special thanks go to my colleagues Jalal Kawash, Tarek Hijazi, and Abbas Tarhini for their valuable comments and discussions, my roommates, the A.C.C. supervisor Mr. Tarek Dana, and the assistant A.C.C. supervisor Mr. Ali Aleywan for their help.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Given an algorithm, *ALGO*, for solving a problem with an underlying data set, *DATA*, the data mapping problem refers to mapping disjoint subsets of *DATA* to the nodes of a multicomputer *MCOMP*, such that the execution time of *ALGO*, on *MCOMP* is minimized. The development of data mapping algorithms that minimize execution time is important for cost-effective utilization of the computational resources offered by the current and future multicomputer.

The assumptions made in this work are the following:

(a) *MCOMP* is a distributed-memory message-passing multicomputers whose nodes are connected by a static point-to-point interconnection [Hwang 1993].

(b) The routing used in *MCOMP* is wormhole routing [Hwang 1993].

(c) The computations model used is loosely synchronous computations model in which nodes perform compute-communicate cycles in a *SPMD* (single program, Multiple Data) scheme [Fox et al. 1988].

Under such assumptions, the execution time on *MCOMP* is determined by the execution time of the slowest node. Thus, the data mapping problem is obviously an optimization problem, and achieving data-parallelism involves nearly-equal distribution of computation workloads over the nodes of *MCOMP*, as well as the minimization of the amount of their inter-node communication. The data mapping problem is an NP-complete, and no optimal solutions can be found in reasonable time. Instead, several methods have been proposed to find acceptable and near-optimal solutions.

Data-parallelism is based on distributing data and associated computations among the nodes [Hwang 1993]. With the single program multiple data programming model, nodes execute the same program independently on the data subsets distributed to them and communicate when non-local information is needed. Thus, the minimization of the execution time of data-parallel programs requires equal distribution of the work load associated with the data objects and the minimization of the concurrency overheads, such as overheads due to the communication, synchronization and other hardware and software factors.

For applications with regular and uniform data sets and certain multicomputer architectures, optimal or near-optimal data mapping can be accomplished by inspection or by simple techniques. However, optimal mapping of general and irregular data sets to various multicomputer topologies is an intractable problem [Ibaraki and Katoh 1988].

Several methods have been proposed for data mapping, such as greedy algorithms, nearest neighbor mapping, clustering techniques, mincut-based heuristics, recursive bisection, geometry based mapping, block-based spatial decomposition, scattered decomposition, neural network, simulated annealing and genetic algorithms [Berger and Bokhari 1987; Fox 1988; Chrisochoides, Houstis, and Houstis 1991; Ercal, 1988; Farhat 1988; Kramer and Muhlenbein 1989; Fox and Furmanski 1988; Simon 1991; Williams 1991; Mansour and Fox 1990]. These algorithms have different properties pertaining to their solution quality, execution time and general applicability .

In this thesis, we present an algorithm that adapts the Tabu search (TS) paradigm [Glover 1989, 1990] for solving the data mapping problem. The TS algorithm is guided by an objective function and does not make apriori assumptions about the underlying data set or multicomputer model. The design decisions for the algorithm components are made in a way to improve the mapping quality while not increasing its execution time. The experimental results show that the TS algorithm yields good-quality solutions which are comparable with those produced by other methods for a number of test cases.

The main contributions of this thesis can be summarized as follows:

(a) Adaptation, for the first time, of Tabu Search algorithm to the data mapping problem. The TS algorithm also serves as general paradigms for solving other optimization problems.

(b) Comparative performance evaluation of tabu search with other algorithms like simulated annealing, genetic algorithm, neural networks, simulated tempering, and mean field annealing.

This thesis is organized as follows. In Chapter 2, the computation and multicomputer models are defined. Chapter 3 contains a general overview of the Tabu Search algorithm. In Chapter 4, we present the Tabu Search algorithm for data mapping. Chapter 5 gives and discusses the experimental results. Chapter 6 concludes the paper.

Chapter 2

Data Mapping Problem

Let *ALGO* be an optimization algorithm intended to solve a given problem with an associated data set, *DATA*, on multicomputer *MCOMP*. The data mapping problem is the problem of partitioning *DATA* into mutually exclusive subsets, and mapping each of these subsets to a node in *MCOMP*. The aim is minimizing the execution time of *ALGO* on *MCOMP*. The execution time of the parallel program depends on the algorithm itself, the underlying data set, the computation model, and the multicomputer machine. In this chapter, we define the data mapping problem in detail. We also presents our assumptions and the objective function used.

1. The Computation Model

To formulate objective functions for the mapping problem, a loosely synchronous data parallel computation model [Fox, Johnson, Lyzenga, Otto, Salmon, and Walker 1988; Fox 1991] is assumed, where nodes run the same instructions and repeat compute-communicate iterations. In every iteration, the nodes perform computations on their allocated subgraphs and then communicate with other nodes to exchange boundary vertex information. For this model, the total parallel execution time is determined by the slowest node [Mansour and Fox 1992, 1994a, 1994b].

2. Problem Definition

Let $G_C = (V_C, E_C)$ and $G_M = (V_M, E_M)$ represent the computation graph and the multicomputer graph, respectively. The vertex set, V_C , represents the set of data elements on which computations are to be performed. The edge set, E_C, represents the computation dependencies among the data elements as specified by *ALGO*. The vertices of the multicomputer graph, V_M, refer to the nodes, and the edges, E_M, refer to the physical interconnections.

The data mapping problem is an optimization problem that refers to determining an onto (many-to-one) function, $\mathcal{MAP} : V_C \rightarrow V_M$, such that an objective function, associated with the execution time of *ALGO* is minimized. A solution that satisfies the minimization criterion is an optimal mapping. Thus, mapping results in partitioning the computation graph into subgraphs allocated to the nodes of the multicomputer.

3. The Objective Function

The objective function, OF_{typ}, is a typical objective function corresponding to the time taken by the slowest processor in computing and communicating in a loosely synchronous computation model [Mansour and Fox 1992, 1994a, 1994b]. OF_{typ} simulates the execution time of the parallel *ALGO*, and is equal to the maximum combined workload of computation, *W(n)*, and communication, *C(n)*, for a node, *n*, in a loosely synchronous iteration. That is,

$$OF_{typ} = \max_{n \in V_M} \{W(n) + C(n)\} \qquad (1)$$

The computation workload, *W(n)*, for a node, n, is given by

$$W(n) = \sum_{v \in V_C} w(v).\delta(v,n) \qquad (2)$$

where

$$\delta(v,n) = \begin{cases} 1 & if \quad v \quad is \quad mapped \quad to \quad n \\ 0 & otherwise \end{cases}$$

w(v) is the computation per vertex *v*. It is given by

$$w(v) = t_{float}.\lambda.\theta(v) \qquad (3)$$

where t_{float} is the time taken by a node to perform a floating point operation. Hence, it is the smallest reasonable time of a machine operation. λ is the number of computation operations per an edge in E_C in one iteration. $\theta(v)$ is the degree of vertex v in G_C, it represents the number of computation operations required for updating a value for vertex v. Both λ and $\theta(v)$ are determined by the particular algorithm used. λ is a constant expressing the number of values updated for a data element in an iteration. Substituting equation 3 in equation 2 yields

$$W(n) = t_{float} . \lambda \sum_{v \in V_C} \theta(v) . \delta(v,n) \tag{4}$$

If we define $S_v(n) = \sum_{v \in V_C} \theta(v) . \delta(v,n)$ to denote the number of edges in E_C which are mapped to node *n*, then (4) becomes

$$W(n) = t_{float} . \lambda . S_V(n) \tag{5}$$

The amount of communication for a node, *n,* is difficult to express accurately. It depends on several hardware and software components of a multicomputer, which varies from one machine to another and some of them might be impossible to quantify. We use $C(n) = t_{\text{float}}\ \zeta(n)$ so that all parameters can be normalized with respect to t_{float} and simulate the multicomputer's communication cost using the model [Bokhari 1990]:

$$\zeta(n) = \sum_{m \in V_M} [\rho B(n,m) + \sigma + \tau H(n,m)] \operatorname{sgn}(B(n,m)) \tag{6}$$

where

$$sign(x) = \begin{cases} 1 & if \quad x > 0 \\ 0 & if \quad x = 0 \end{cases}$$

ρ is the machine time for communicating one word. It is given by the number of "t_{float}" s (i.e., it is divided by t_{float}) required to communicate a word. σ is the message

OF_{typ} is the basis for evaluating the solution qualities of mapping algorithms. The solution quality is represented by the concurrent efficiency of the parallel *ALGO*, which corresponds to a mapping configuration. Using the models defined in equations 1-5, efficiency is given by

$$\eta = \frac{\sum_{n \in V_M} W(n)}{|V_M| . OF_{typ}}$$

$$= \frac{\lambda \sum_{n \in V_M} S_v(n)}{|V_M| . \max_n \{\lambda S_v + \zeta(n)\}} \tag{8}$$

However, it is unfeasible to use OF_{typ} because both *W(n)* and *C(n)* correspond to conflicting requirements. In other words, minimizing OF_{typ} gives rise to a min-max criterion which is computationally expensive, because the calculation of a new OF_{typ} caused by an incremental change in the mapping of data objects to nodes may require the calculation of the loads of all nodes. We will circumvent this problem by using a quadratic objective function, OF_{appr}, which is considered to be a good approximation of OF_{typ} [Mansour and Fox 1992, 1994a, 1994b; Fox et al.1988].

$$OF_{appr} = \lambda^2 \sum_{n \in V_M} S_v^2(n) + \mu \sum_{n \in V_M} \zeta(n) \tag{9}$$

where μ is a scaling factor expressing the relative importance of the communication term with respect to the computation term. It is given by

$$\mu = \mu_{user} \frac{\lambda^2 \sum \theta(n)}{4\rho |V_M| \sqrt{V_C} (\sqrt{|V_M|} - 1). E_b\_V_b} \qquad \textbf{(10)}$$

The derivation of μ is given in [Mansour 1992]. μ_{user} and $E_b\_V_b$ are user-defined parameters selected based on experience.

The main advantages of the quadratic objective function, OF_{appr}, are its smoothness, its locality property, and that it is cheaper to parallelize. Smoothness makes it more suitable for optimization methods, that is cheaper. Locality means that a change in the cost due to change in the mapping of data objects to nodes is determined by the remapped objects and the relevant nodes only. Specifically, the change in OF_{appr} due to remapping of object v from node n to node m is given by

$$\Delta OF_{appr} = 2\lambda^2 \theta(v) \left[\theta(v) + S_v(n) - S_v(m)\right] + \mu[\Delta C] \qquad \textbf{(11)}$$

The use of OF_{appr} is essential to most optimization algorithms because they employ vertex remapping extensively.

Chapter 3

Tabu Search

The Tabu Search paradigm [Glover 1989] aims for finding a good suboptimal solution in a complex solution space based on two strategies: freeing the search by functions of short-term memory and aspiration levels, and constraining the search by classifying certain moves as forbidden (i.e. Tabu). Tabu Search is a higher level heuristic procedure for solving optimization problems, designed to escape the trap of local optimality. This algorithm can be used to guide any process that employs a set of moves for transforming one solution (or solution state) into another and that provides an evaluation function for measuring the attractiveness of these moves. The form of the guidance provided by tabu search is highly flexible and often motivates the creation of new types of moves and evaluation criteria to take advantage of its adaptability to different problem structures and strategic goals.

Figure 1 presents the general Tabu Search algorithm. *TS* starts with a random initial solution and then generates a candidate list of moves, S. A candidate move is that which can be applied to the current solution to generate a new solution. Different problems would impose different constraints on the size of S and how its elements are determined. In Chapter 4 we discuss the set S for the data mapping problem.

```
1- Begin with a starting current solution.
{Obtain the solution from initialization or from intermediate or long-term memory}
2-Create a candidate list of moves
{if applied, each move would generate a new solution from the current solution}
3-Choose the best admissible candidate
{Admissibility is based on the Tabu restrictions and aspiration criteria.) Designate the solution obtained as the new current solution. Record it as the new best solution if it improves on the previous best}.
4-If a chosen number of iterations has elapsed in total or since the last best solution was found then stop by either terminating globally or transfer.
{A transfer initiates an intensification or diversification phase embodied in an intermediate or long-term memory component}.
else
    Update admissibility conditions and go to step 2.
    {Update tabu restrictions and aspiration criteria}.
```

Figure 1. Tabu Search Algorithm

If certain number of iterations has elapsed, the algorithm would either terminate globally or initiates an intensification or diversification phase embodied in an intermediate or long term-memory component. Otherwise, admissibility conditions would be updated, and the algorithms restart for other number of iterations.

TS uses intermediate- and long-term memory functions to achieve local intensification and global diversification of the search. The long-term memory function employs principles that are roughly the reverse of those for intermediate-term memory. Instead of inducing the search to focus more intensively on regions that contain good solutions previously found, the long term-memory function guides the search to regions that contrast with those examined thus far.

Figure 2 outlines the features of selecting the best admissible move. The algorithm searches all elements (moves) of the candidate list *S*, and selects the best one. If the move is not tabu, the move would be considered as admissible. Otherwise, that is if the move is tabu, it will not be disregard but it takes another chance to be admissible by checking it against aspiration criteria.

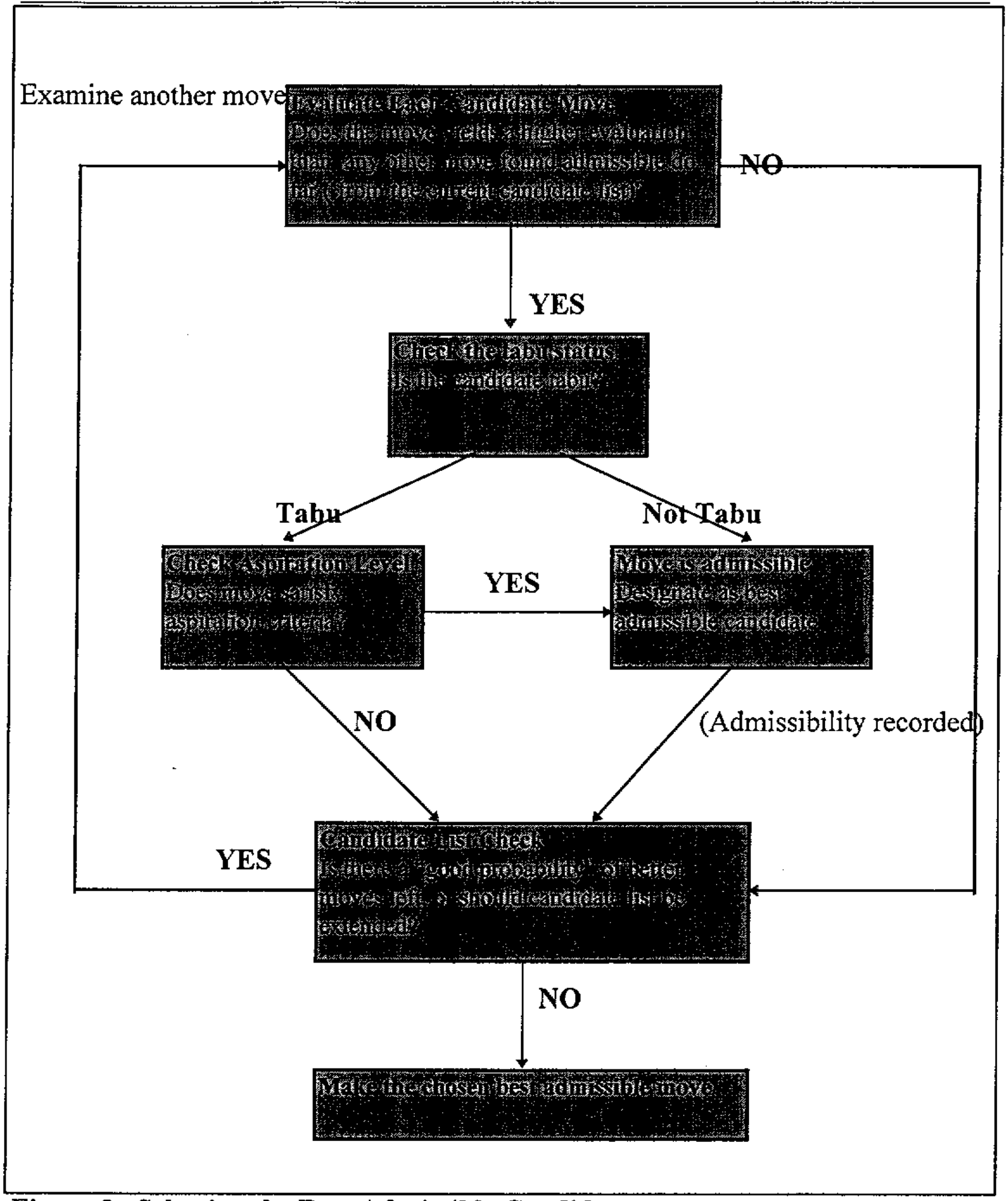
Examine another move
Evaluate Each Candidate Move
Does the move yield a higher evaluation than any other move found admissible so far (from the current candidate list)?
NO
YES
Check the Tabu Status
Is the candidate tabu?
Tabu
Not Tabu
Check Aspiration Level
Does move satisfy aspiration criteria?
YES
Move is admissible
Designate as best admissible candidate
NO
(Admissibility recorded)
YES
Candidate List Check
Is there a "good probability" of better moves left, or should candidate list be extended?
NO
Make the chosen best admissible move

Figure 2. Selecting the Best Admissible Candidate

A fundamental feature of the *TS* is the construction of a list of Tabu moves that are not allowed at the current step. The objective is to exclude moves which bring the search back to a previously visited point, which might trap it at a local minimum. In fact, Tabu moves are recorded in this list, for a specified number of iterations and then are removed, freeing them from their Tabu status. This short-term memory function of *TS* is customarily handled by treating the Tabu list as a circular list.

TS incorporates aspiration level functions which provide added flexibility to choose good moves by allowing the Tabu status of a move to be overridden if the aspiration level is attained. In fact, the Tabu restrictions and aspiration level function play a dual role in constraining and guiding the search process. Tabu restrictions allow a move to be regarded as admissible if they do not apply, while aspiration criteria allow a move to be regarded as admissible if they do apply.

Chapter 4

Tabu Search Algorithm Applied to Data Mapping Problem

The Tabu Search paradigm has been applied to graph partitioning [Tao, Zao, Guo, Thulasiramon, and Swamy 1991; Tao and Zao 1993] and to graph coloring [Hertz and DeWerra 1987], but not to data mapping which involves different requirements and constraints. Adapting Tabu Search to data mapping (Figure 3) requires design decision concerning the candidate list of moves, move selection, Tabu lists, aspiration levels, and techniques for intensifying and diversifying the search, and convergence. *TS* starts with a random initial solution and then generates a candidate list of moves, *S*. A candidate move is that move which can be applied to the current solution to generate a new solution. The best move is usually selected from *S* and checking Tabu status is the first step in screening for admissibility. Admissibility is based on the Tabu restrictions and aspiration criteria. That is, if the move is not Tabu, it is immediately admitted; otherwise, the aspiration criteria are given an opportunity to override the Tabu status, providing the move a second chance to qualify as admissible. If this fails, the next best non-Tabu move is selected from S and will be admitted. Therefore, at each iteration the algorithm must select two moves: the best move and the second non-Tabu best one; otherwise, if the admissible selected move is Tabu, and does not improve the solution, the aspiration criteria would not be satisfied leaving the algorithm in an infinite loop.

```
1- Begin with random initial mapping, MAP[], and empty list of Tabu moves, T;
2- Generate a candidate list of moves, S;
3- Select the best move from S;
4- If the move is Tabu then  If it satisfies aspiration criteria
                                     then admit it
                             else reject it and admit the
                                  next best move in S which is not Tabu
       else admit it;
5- If converged, stop. Otherwise, update T and return to step 2;
```

Figure 3. Tabu Search algorithm Applied to Data Mapping.

1. Candidate List of Moves and Objective Function

We use the array *MAP* with *D* data elements ($D=|V_C|$) to represent a mapping configuration, where the value of *MAP[i]* represents the multicomputer node *n* to which the *i*th data element is mapped. A move consists of remapping *i* from n to node m. Thus, the candidate list, *S*, consists of all moves produced by remapping every data element to a new node. For N nodes ($N=|V_M|$) and D data elements, the size of S is equal to D*(N-1).

A move is evaluated by computing the objective function value (Equation 1). This is not an expensive operation, since it is based on the change produced in the objective function. In Step 4 of the *TS* algorithm, a move is usually selected from S such that it corresponds to the best change (greatest decrease) in the objective function.

2. The Tabu List

We use a Tabu list, T, of length 7 to save the 7 most recent moves, which are then removed, freeing them from their Tabu status. This short-memory function of *TS* can be accomplished by making T a circular list. Each item in the list has three attributes: the data element number, its old node, and its new one. T is used to prevent cycling or repetition of moves by rendering the attributes of these seven moves forbidden (Tabu). This would permit the search to go beyond points of local optimality while still making good quality moves at each step.

3. The Aspiration Criteria

Another important feature incorporated in *TS* is the aspiration level functions which provide added flexibility to choose good moves by allowing the Tabu status of a move to be overridden if the aspiration level is attained. In fact, the Tabu restrictions and aspiration level function play a dual role in constraining and guiding the search process. We use three types of aspiration levels:

i) First aspiration level is to accept a Tabu move if it offers an improvement in the current objective function.

ii) Second level is to accept a Tabu move from node *n* to node *m* if node *n* is allocated 20 or more data elements than node *m*.

iii) Third level is to accept a Tabu move for a data element depending on the mapping of its neighboring data elements. Specifically, we allow remapping an element if the majority of its neighbors are mapped to the target node.

If the best move selected from *S* is Tabu, one or more of these aspiration criteria is used to override the Tabu status, otherwise the next best non-Tabu move is admitted.

4. Intensifying and Diversifying the Search

For data mapping, we employ a simple function to achieve both local intensification and global diversification. This function is based on two types of moves. The first type consists of remapping the selected data element to the best target node. The second type consists of remapping the data element and all its neighborhood to the best target node. The first type is the normal move and would intensify the search. The second type is applied every 100 moves for 10 moves and is meant to diversify the search.

5. Convergence

The intensification and diversification features of *TS* make it possible to avoid trapping at a local minimum because they always guide the search to different regions. Moreover, the short-term memory of *TS* prevent cycling in any region. We did make use of these two facts to implement the concept of convergence by making the algorithm converge every 100 iteration.

6. Complexity

The complexity of TS is of the order of (Num_of_passes* $|V_C|*|V_M|$), where Num_of_passes is the maximum number of iterations, $|V_C|$ and $|V_M|$ represent the sizes of the computation and multicomputer graphs respectively. Therefore, the execution time of TS is directly proportional to the number of multicomputer nodes, and the number of the computation nodes.

Chapter 5

Experimental Results

In this chapter, we present the experimental results of the TS algorithm, using the simulated models presented in Chapter 4. We also compare the TS algorithm with simulated annealing (SA), genetic algorithm (GA), neural network (NN), mean field annealing (MFA), simulated tempering (ST) using the results reported in [Kawash 1994; Aghazarian 1996]. We conduct our experiments on an AViiON 5000 UNIX machine. The values used for the parameters of the simulated computational and multicomputer models are given in Table 1, assuming an NCUBE-2 hypercube multicomputer [Berrenderf, and Helin 1992].

λ	ρ	σ	τ
7	**15**	**325**	**100**

Table 1. Simulation parameters.

1. Results for Published Test Cases.

In this section, we use three test cases used in the published literature [Kawash, Manour, and Diab 1995; Chrisochoides, Mansour, and Fox 1994], which are referred to as TCASE1,TCASE2, and TCASE3. These test cases are depicted in Figure 4 and their properties are given in Table 2, where D = $| V_C |$ and N = $| V_M |$ represent the sizes of the computation graph and the multicomputer graph respectively.

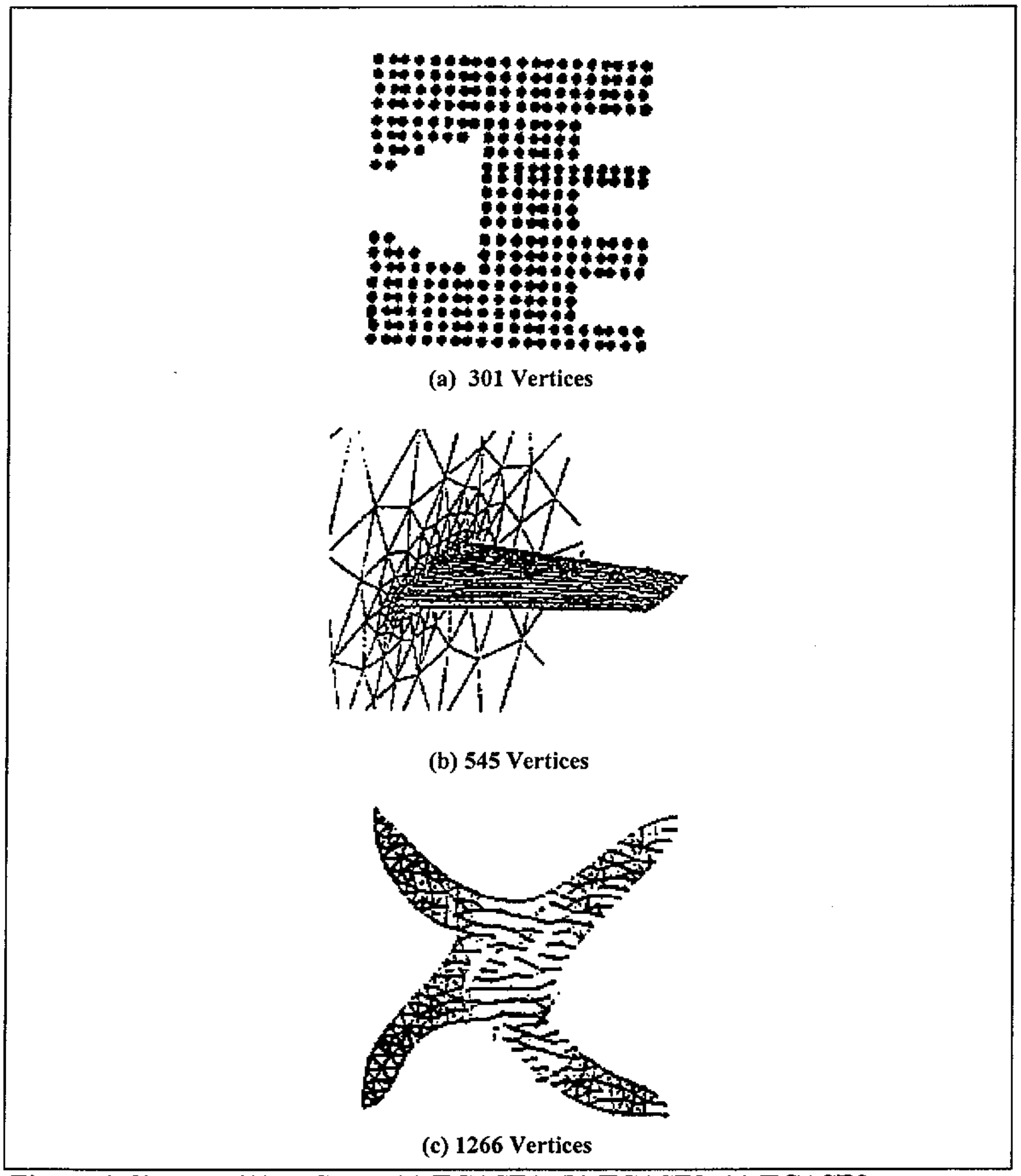

Figure 4. Shapes of Test Cases. (a) TCASE1. (b) TCASE2. (c) TCASE3.

	D	N	*Avg. Vertex Degree*	*Figure*
TCASE1	301	4	3	Figure 4(a)
TCASE2	545	8	11.5	Figure 4(b)
TCASE3	1266	16	5.5	Figure 4(c)

Table 2. Test Cases

In Table 3, we present the execution time, t_{exec} in seconds, and the solution quality, η (Equation 4) for the six mapping algorithms. These results show that for TCASE1, the six algorithms give close solution quality (59.7% TS - 65.2% SA), whereas t_{exec} differs significantly: 2 seconds for NN to 211 seconds for SA. TS has the fourth execution time. For TCASE2, η varies between 52.3% (NN) and 59.3% (GA) but the difference in t_{exec} is again considerable (5 seconds for NN, 1652 seconds for GA). TS has again the fourth t_{exec}, but the second best η. For TCASE3, the margin is larger for η (46.5% TS - 58.8% GA) but still NN has the fastest t_{exec} (8 seconds) compared to GA (940 seconds); TS gives the least η and the fourth t_{exec}.

	TCASE 1		TCASE 2		TCASE 3	
	η	t_{exec}	η	t_{exec}	η	t_{exec}
SA	65.2%	211	57.2%	1095	54.2%	560
GA	61.9%	83	59.3%	1652	58.8%	940
NN	60.3%	2	52.3%	5	49.3%	8
MFA	63.7%	47	55.9%	122	53.5%	585
ST	60.3%	17	55.2%	132	50.0%	51
TS	59.7%	61	57.3%	413	46.5%	556

Table 3. Result of Mapping Algorithms for TCASE1, TCASE2, and TCASE3

Figures 5 through 10 show the solution qualities and execution time for TCASE1,TCASE2, and TCASE3.

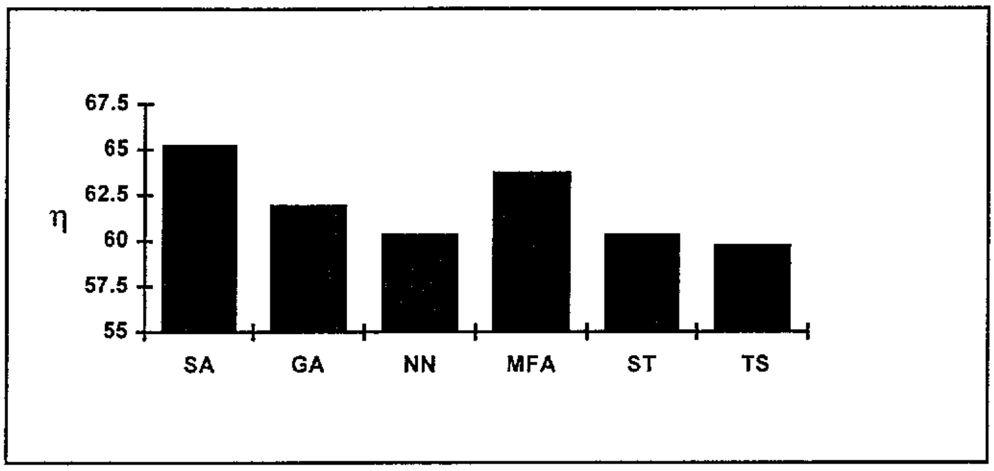

Figure 5. Solution Quality for TCASE1.

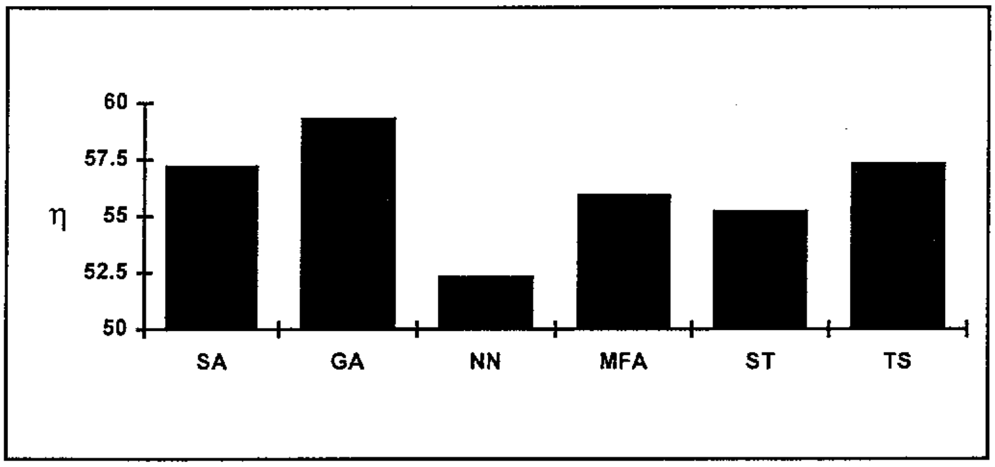

Figure 6. Solution Quality for TCASE2.

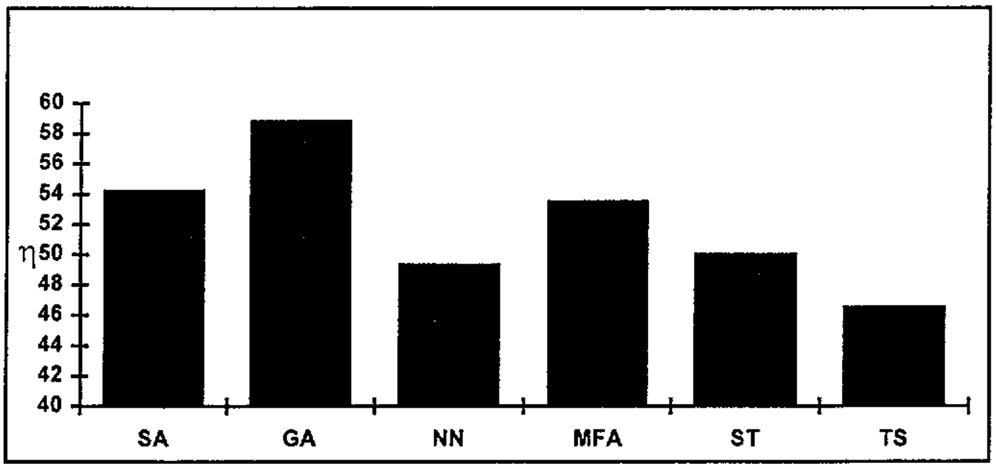

Figure 7. Solution Quality for TCASE3.

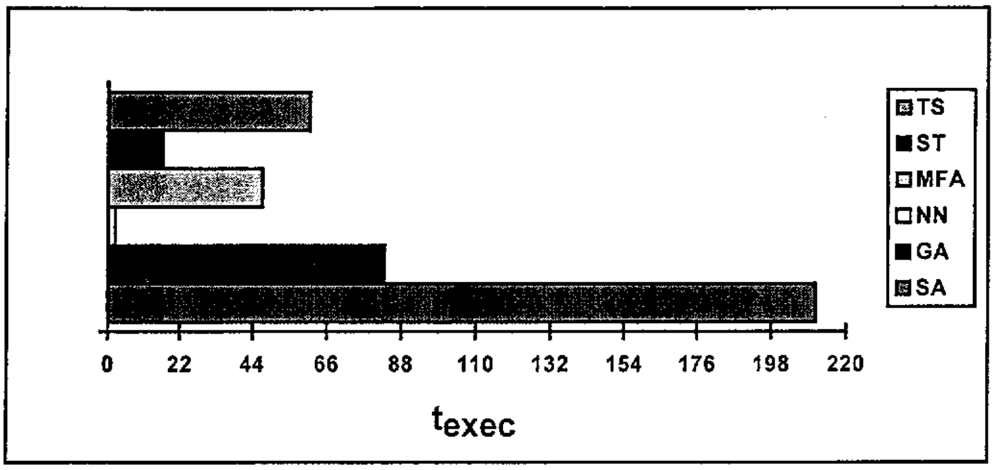

Figure 8. Execution Time for TCASE1.

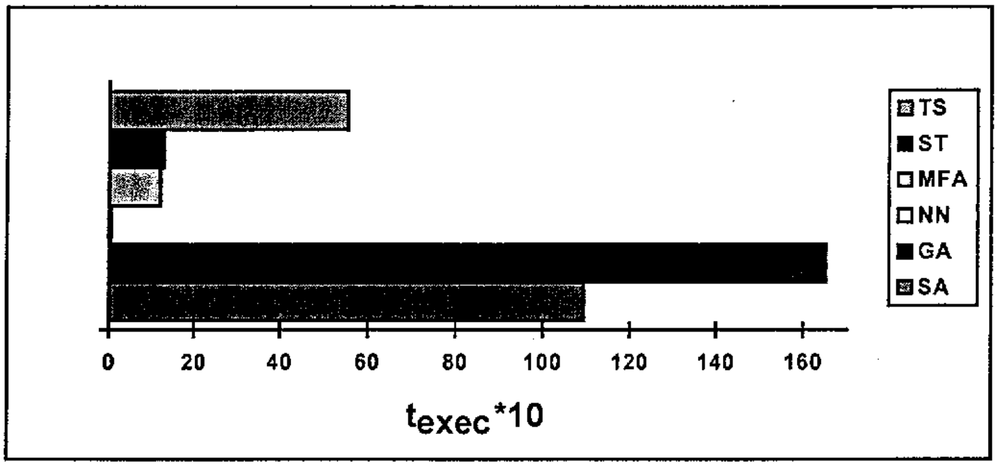

Figure 9. Execution Time for TCASE2.

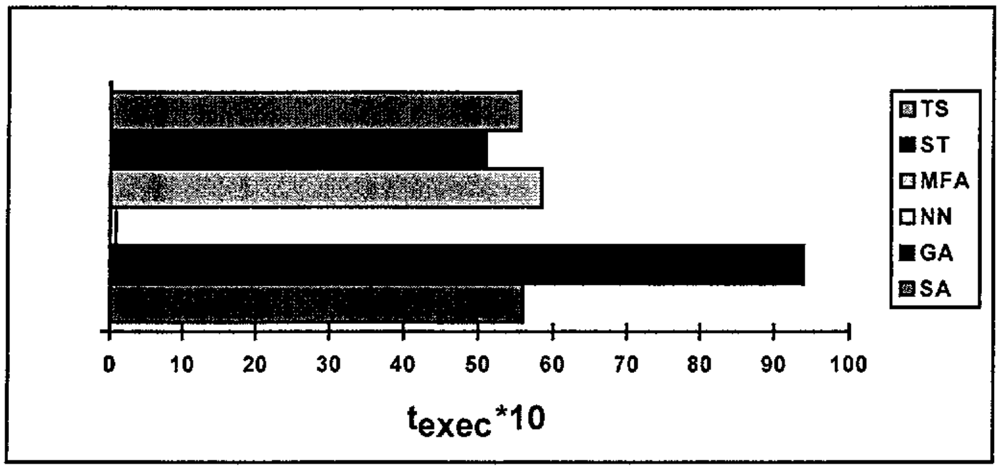

Figure 10. Execution Time for TCASE3.

2. Results for Random Graphs

In this section we present the results for eleven selected test cases (SCASEs) mapped to different numbers of nodes. The test cases, with different sizes and vertex degrees, were generated using a random graph generator and their characteristics are given in Table 4. The experimental results for these test cases, presented in Tables 5-15, show that NN is the fastest algorithm and that GA gives the best solution quality. The execution time of TS is always the fourth, whereas its solution quality ranges from third to sixth.

	D	N	Avg. Vertex Degree
SCASE1	300	4	3
SCASE2	450	4	3
SCASE3	500	4	4
SCASE4	500	8	4
SCASE5	750	8	6
SCASE6	1000	8	5
SCASE7	1000	16	5
SCASE8	1500	8	12
SCASE9	1500	16	12
SCASE10	2000	8	8
SCASE 11	2000	16	8

Table 4. Randomly Generated Test Cases

	SA	GA	NN	MFA	ST	TS
η	45.1%	46.6%	43.5%	43.5%	43.7%	38.3%
t_{exec}	325	153	2	45	19	67

Table 5. SCASE1

	SA	GA	NN	MFA	ST	TS
η	49.5%	52.6%	47.6%	50.3%	49.3%	49.3%
t_{exec}	362	722	4	32	23	155

Table 6. SCASE2

	SA	GA	NN	MFA	ST	TS
η	46.2%	53.9%	46.7%	45.7%	43.9%	46.7%
t_{exec}	589	886	2	18	47	101

Table 7. SCASE3

	SA	GA	NN	MFA	ST	TS
η	27.9%	29.1%	26.2%	21.7%	27.5%	27.3%
t_{exec}	731	765	4	59	52	257

Table 8. SCASE4

	SA	GA	NN	MFA	ST	TS
η	36.2%	37.5%	32.3%	30.0%	33.3%	29.5%
t_{exec}	1714	6345	8	100	140	515

Table 9. SCASE5

	SA	GA	NN	MFA	ST	TS
η	33.7%	38.9%	32.1%	38.0%	32.5%	28.1%
t_{exec}	1713	2717	12	396	131	483

Table 10. SCASE6

	SA	GA	NN	MFA	ST	TS
η	18.0%	18.3%	16.0%	13.6%	17.6%	17.7%
t_{exec}	2037	2372	17	637	161	1146

Table 11. SCASE7

	SA	GA	NN	ST	TS
η	40.8%	46.1%	34.9%	34.6%	40.3%
t_{exec}	4798	10845	26	448	1300

Table 12. SCASE8

	SA	GA	NN	ST	TS
η	26.0%	26.9%	21.1%	25.5%	25.8%
t_{exec}	5240	4460	37	482	3056

Table 13. SCASE9

	SA	GA	NN	ST	TS
η	31.7%	47.7%	35.8%	23.7%	22.4%
t_{exec}	5885	13171	40	644	4076

Table 14. SCASE10

	SA	GA	NN	ST	TS
η	40.8%	46.1%		20.4%	20.6%
t_{exec}	4798	10845		976	3610

Table 15. SCASE11

Chapter 6

Conclusions

We have presented a Tabu Search algorithm for data mapping, assuming loosely synchronous computational algorithms and Equations 1-4 for the computational and multicomputer models. We have also given experimental results for various test cases. These results show that in comparison with other five mapping algorithms, the *TS* algorithm ranks fourth in terms of execution time and, on average, close to fourth in terms of solution quality.

References

Aghazarian, G. 1996. Simulated Tempering and Mean Field Annealing for mapping to multicomputers. MS Thesis, Lebanese American University.

Aghazarian, G., and Mansour, N. 1996. Simulated Tempering Algorithm for Mapping Data to Multicomputers. Science Week, a conference organized by the Syrian Higher Council for Sciences, Aleppo, November 2-7.

Berger, M, and Bokhari. 1987. A partitioning strategy for nonuniform problems on multicomputer. IEEE Trans. Computers, C-36, 5 (May), pp. 570-580.

Berrenderf, R., and Helin J. 1992. Evaluating the basic performance of the Intel iPSC/860 parallel computer. Concurrency: Practice and Experience, May.

Bokhari S. 1990. Communication overhead on the INTEL iPSC-860 hypercube. ICASE Report no. 90-10.

Chrisochoides, N., Mansour, N., and Fox, G. 1994. Performance evaluation of load balancing algorithms for parallel single-phase iterative PDE solvers. Scalable High-Performance Computing Conference, May.

Chrisochoides, N.P., Houstis, C.E., and Houstis E.N. 1991. Geometry based mapping strategies for PDE computations. Int. Conf. On Supercomputing, pp. 115-157, ACM Press.

Ercal, F. 1988. Heuristics Approaches To Task Allocation For Parallel Computing. Ph.D. Thesis, Ohio State University.

Farhat, C. 1988. A simple and efficient automatic FEM domain decomposer. Computers and Structures. Vol. 28, no. 5, pp. 579-602.

Fox, G.C. 1988. A review of automatic load balancing and decomposition methods for the hypercube. In Numerical Algorithms for Modern Parallel Computers, ed. M. Shultz, pp. 63-76, Springer-Verlag.

Fox, G.C., Johnson M., Lyzenga G., Otto S., Salmon J., and Walker D. 1988. Solving Problems on Concurrent Processors. Englewood Cliffs, N.J., Printice Hall.

Fox, G.C., 1991. The architecture of problems and portable parallel software systems. Supercomputing'91, also SCCS-78b, NPAC, Syracuse University.

Fox, G.C., and Furmanski, W. 1988. Load balancing loosely synchronous problems with a neural network. 3rd Conf. Hypercube Concurrent Computers and Applications, pp. 241-278.

Fox, G.C., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. 1988. Solving Problems on Concurrent Computers. Prentice hall.

Glover, F. 1989. Tabu search - part 1, ORSA Journal on Computing, Vol. 1, No.1, pp. 190-206. **Glover, F. 1990.** Tabu search - part 2, ORSA Journal on Computing, Vol. 2, No.1, pp. 4-32.

Hertz, A. and DeWerra, D. 1987. Using Tabu search techniques for graph coloring. Computing 29, pp. 345-351.

Hwang, K. 1993. Advanced Computer Architecture. McGraw-Hill, Inc.

Ibaraki, T., and Katoh, N. 1988. Resource Allocation Problems. MIT Press.

Kawash, J. 1994. Sensitivity to Parameters and General Applicability of Genetic Algorithms and Simulated Annealing Algorithms for Mapping Data to Multicomputers. MS Thesis, Lebanese American University.

Kawash, J., Manour N., and Diab, H. 1995. General applicability of genetic and simulated annealing algorithms for data mapping, IASTED Int. Conf. Parallel and Distributed Computing and Systems, October 18-21.

Kramer, O., and Muhlenbein H. 1989. Mapping strategies in message-based multicomputer systems. Parallel Computing 9, pp. 213-225.

Mansour, N., and Fox, G.C. 1990. Allocating irregular data to distributed-memory multiprocessors by genetic algorithms. Concurrency: Practice and Experience, 6(6), Sept., pp. 485-504.

Mansour N. 1992. Physical optimzation algorithms for mapping data to distributed-memory multiprocessors. Ph.D. diss., Comp. Sci., Syracuse, N.Y.

Mansour, N., and Fox G.C. 1992. Allocating data to multicomputer nodes by physical optimization algorithms for loosely synchronous computations. Concurrency: Practice and Experience, 4(7)557-574.

Mansour, N., and Fox G.C. 1994a. Parallel physical optimization for allocating data to multicomputer nodes. The Journal of Supercomputing, 8:53-80.

Mansour, N., and Fox G.C. 1994b. Allocating data to distributed-memory multiprocesor by genetic algorithms. Concurrency: Practice and Experience, 6(6)485-504.

Pothen, A., Simon, H., and Liou, K-P. 1990. Partitioning sparse matrices with eigen vectors of graphs. SIAM J. Matrix Anal. Appl., 11, 3 (July), pp. 430-452.

Simon, H. 1991. Partitioning of nonstructured mesh problems for parallel processing. Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications. Permagon Press.

Tao L., and Zao Y. 1993. Multiway-way graph partition by STOCHASTIC probe. Computers Ops. Res. Vol. 20, No.3, pp. 321-347.

Tao, L., Zao, Y.C., Guo, J., Thulasiramon, K., and Swamy, N.M.S. 1991. An efficient Tabu search algorithm for m-way graph partition. Supercomputing Symposium'91, Fredericton, NB Canada, June 3-5, pp. 263-270.

Williams, R.D. 1991. Performance of dynamic load balancing algorithms for unstructured mesh calculations. Concurrency: Practice and Experience, 3(5), pp. 457-481.