Parallelizing Tabu Search on a Cluster of Heterogeneous Workstations

Ahmad Al-Yamani Sadiq M. Sait Habib Youssef
Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
sadiq@ccse.kfupm.edu.sa

Hassan Barada
Etisalat College of Engineering, Sharjah, UAE

Abstract

In this paper, we present the parallelization of tabu search on a network of workstations using PVM. Two parallelization strategies are integrated: functional decomposition strategy and multi-search threads strategy. In addition, domain decomposition strategy is implemented probabilistically. The performance of each strategy is observed and analyzed. The goal of parallelization is to speedup the search in finding better quality solutions. Observations support that both parallelization strategies are beneficial, with functional decomposition producing slightly better results. Experiments were conducted for the VLSI cell placement, an NP-hard problem, and the objective was to achieve the best possible solution in terms of interconnection length, timing performance (circuit speed), and area. The multiobjective nature of this problem is addressed using a fuzzy goal-based cost computation.

Index Terms: Tabu Search, Parallel Tabu Search, Metaheuristic, Functional Decomposition, Multi-Search Threads, Combinatorial Optimization, VLSI, Standard Cell Design, Placement, Fuzzy Logic.

1 Introduction

Tabu Search (TS) belongs to the class of general iterative heuristics that are used for solving hard combinatorial optimization problems. It is a generalization of local search that searches for the best move in the neighborhood of the current solution. However, unlike local search, TS does not get trapped in local optima because it also accepts bad moves if they are expected to lead to unvisited solutions [1, 2].

Among the iterative stochastic heuristics applied to combinatorial optimization problems are *Simulated Annealing* (SA) [3, 4, 5, 6], *Genetic Algorithm* (GA) [7, 8], and *Simulated Evolution* (SE) [9, 10]. A common feature of these stochastic iterative heuristics is that they are memoryless. They do not have memory or use any memory structure to keep track of previously visited solutions. On the other hand, TS, utilizes some memory to make decisions at various stages of the search process [2, 11, 12]. Memory structures are used to prevent reverses of recent moves by keeping their attributes in a tabu list (also known as short-term memory) in order to prevent cycling back to already visited solutions. Memory structures are also used (1) to force new solutions to have different features from previously visited ones (*diversification*) [13, 14]; (2) to force the new solution to have some features that have been seen in recent good solutions (*intensification*) [11, 15]. It is up to the user to specify what is required from TS at various stages of the search process.

Because of its search strategy the parallelization of TS can result in improved solution quality and reduced execution time. Encouraging results are obtained for computationally intensive tasks even with a small number of workstations in a local area network.

In 'Distributed Computing', several interconnected computers work together to solve a large problem [16]. The benefits of creating a distributed computing environment employing a network of workstations has been addressed and investigated. These benefits include: (i) cost effectiveness compared to an expensive large multiprocessor supercomputer alternative [17], (ii) the utilization of the abundant computation power of PCs and workstations that remain idle for a large fraction of time, (iii) the availability of high speed transmission links that have capacities in the order of gigabits/sec [18], etc., to name a few. Researchers from Oak Ridge National Laboratory, the University of Tennessee, and Emory University, developed a Parallel Virtual Machine **(PVM)** system that assists in the implementation of developed parallel algorithms to be executed on a network of heterogeneous workstations. The algorithm is developed as a collection of communicating tasks, where, message passing, format conversion, task scheduling, etc., are handled by PVM [16, 19].

The paper is organized as follows. In the following section (Section 2) we state the placement problem. Our method of estimating the cost with respect to three objectives using fuzzy goal-based approach is explained in Section 2.1. A brief description of tabu search is then presented in Section 3.

Various parallelization strategies proposed in the literature are reviewed in Section 4. The proposed parallel tabu search algorithm is described in Section 5. In this section we also discuss the application of the algorithm to a network of heterogeneous machines. Experimental results, their analysis and discussion are given in the final section.

2 VLSI Multi-Objective Cell Placement

The problem of cell placement can be defined as finding suitable locations for all cells in a rectangular layout surface. A suitable location is one that optimizes some objectives like wire length, delay, and/or area. A semi-formal definition of the VLSI Cell placement problem can be found in the literature [3]. For the standard cell layout style assumed in this work, all the cells are of equal height but vary in width. They must be arranged in rows without overlapping (see Figure 1). Channels between the rows are of variable height to allow routing of connections between cells [3, 20, 21].

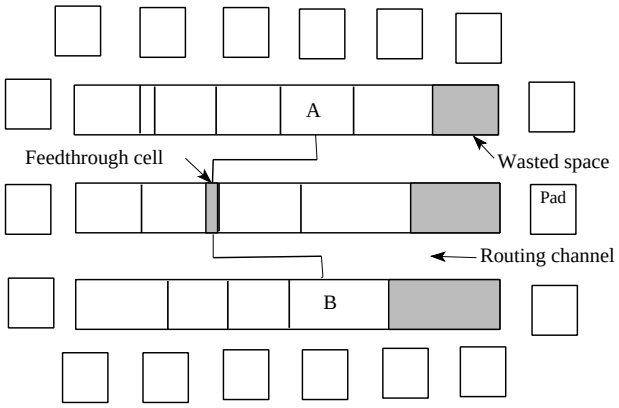


Figure 1: Configuration of a VLSI standard cell layout.

2.1 Fuzzy Goal-Based Cost Computation

Cell placement is a hard combinatorial optimization problem with a number of noisy objective functions. A solution is evaluated with respect to three main objectives: wire length, critical path delay (timing performance), and area, which is a function of cell delays and interconnection delays. Prior to final layout, these criteria cannot be accurately measured. Further, it is unlikely that a placement that optimizes all three objectives exists. Designers usually have to make tradeoffs. To deal with such complex and imprecise objectives, a fuzzy goal-directed search approach is applied [10]. Below we briefly summarize this approach.

Let there be Π solutions generated by the algorithm. Assume that we are optimizing a p-valued cost vector given by $C(x) = (C_1(x), C_2(x), \cdots, C_p(x))$

where $x \in \Pi$. Assume that a vector $O = (O_1, O_2, \cdots, O_p)$ gives lower bound estimates on individual objectives such that $O_i \leq C_i(x)\ \ \forall i, \forall x \in \Pi$, $1 \leq i \leq p$. O_i's, the lower bounds on each objective do not have to be achievable in practice. Further, assume that there is a user specified goal vector $G = (g_1, g_2, \cdots, g_p)$ which indicates the relative *acceptable limits* for each objective. This means that x is an acceptable solution if $C_i(x) \leq g_i \times O_i$ where $\forall i, g_i > 1.0$.

In the scheme used, the *acceptable solution* set is considered as a fuzzy set [22, 23, 24]. For VLSI cell placement problem of minimizing three parameters, the following rule is used to determine the membership in the fuzzy set *acceptable solution*.

Rule1: **If** a solution is *within acceptable wire length* AND *within acceptable delay* AND *within acceptable area*, **THEN** it is an acceptable solution.

Using fuzzy algebraic notation, while adopting the AND-like ordered weighted averaging operator of Yager [25], the above rule is expressed as follows,

$$\mu(x) = \beta \times min(\mu_1(x), \mu_2(x), \mu_3(x)) + (1 - \beta) \times \frac{1}{3} \sum_{i=1}^{3} \mu_i(x) \qquad (1)$$

where, $\mu(x)$ is the membership value for solution x in fuzzy set *acceptable solutions*, and β is an averaging constant. μ_i for $i = \{1, 2, 3\}$ represents the membership values of solution x in the fuzzy sets *within acceptable wire length*, *within acceptable circuit delay*, and *within acceptable area* respectively. The solution x^* which results in the maximum value for Equation 1 is reported as the best solution found, i.e., $x^* = \max \mu(x), \forall x \in \Omega$, where Ω is the solution subspace searched by the algorithm. The membership function for a general objective 'i' is a function of the cost C_i, the lower bound O_i and goal g_i. (see Figure 2). User preferences can be easily expressed in goal vector G. For example, by decreasing the goal value g_i, the subsequent membership value $\mu_i(x)$ for objective i will decrease.

3 Tabu Search

TS starts with an initial solution s selected randomly or using any constructive algorithm. It then defines a subset $V^*(s)$, called candidate list, of its neighborhood $\aleph(s)$. The algorithm selects the best solution in $V^*(s)$ (in terms of an evaluation function) call it s^*, to be considered as the next solution. If the short term memory does not define the move leading to s^* as *tabu*, it is accepted as the new solution even if it is worse than the current solution in terms of the evaluation function (see Equation 1). However, if the move leading to s^* is *tabu*, the solution is not accepted unless a certain criteria, known as *aspiration criteria* is satisfied. Aspiration criterion is used to check whether the tabu solution is accepted or not [1].

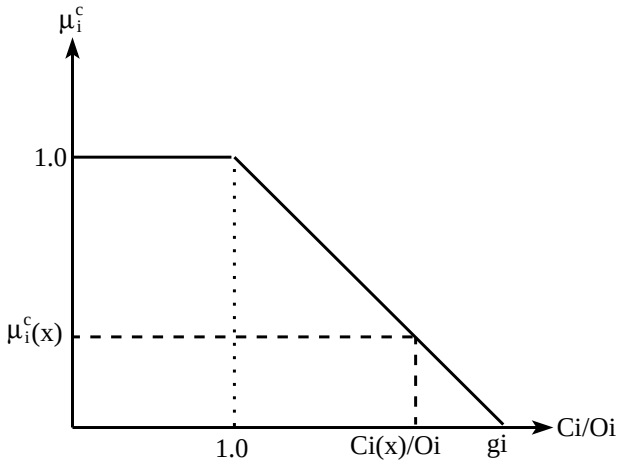


Figure 2: The membership function *within acceptable criterion i*.

A move consists of the swapping of two cells; m pairs of cells are trial swapped and the best swap among them is accepted. A compound move can be made d times where each time m other pairs are tested, where d is the desired move depth, and the best move is taken each time (steps 4 to 16 of Figure 3 are repeated d times). The algorithm checks if the move is tabu by considering only the two cells that were swapped first in the compound move. If the move is found tabu, the *aspiration criterion* is checked. In this work, best cost aspiration criteria is used [11, 12], where a tabu move is accepted if the cost is better than the best cost seen thus far. If the move satisfies the aspiration criteria, it is accepted; else it is rejected, and the process repeats. *Tabu tenure* i.e., number of iterations the move remains tabu is a parameter of the circuit size. The basic description of TS is shown in Figure 3 [1].

4 Literature Review

According to Crainic et. al taxonomy [26], a possible parallelization strategy of *tabu search* is to distribute the computation that requires the most CPU time on available machines (*functional decomposition*). For example, in the case of VLSI cell placement this corresponds to the evaluation of a candidate list of placement configurations. Another strategy is to perform many independent searches (*multi-search threads*). A third strategy, is to decompose the search space among processes (*domain decomposition*). Using a different taxonomy, Crainic et. al., classify TS along three dimensions. The first dimension is **control cardinality** where the algorithm is either *1-control* or *p-control*. In a *1-control* algorithm, one processor executes the search and distributes numerically intensive tasks on other processors. In a *p-control*

```
Algorithm Tabu_Search;
        X     :  Set of feasible solutions.
        s     :  Current solution.
        s*    :  Best admissible solution.
        C     :  Objective function.
        ℵ(s)  :  Neighborhood of s ∈ X.
        V*    :  Sample of neighborhood solutions.
        TL    :  Tabu list.
        AL    :  Aspiration Level.

1.      Start with an initial feasible solution s ∈ X.
2.      Initialize tabu lists and aspiration level.
3.      For fixed number of iterations Do
4.          Generate neighbor solutions V* ⊂ ℵ(s).
5.          Find best s* ∈ V*.
6.          If move s to s* is not in TL Then
7.              Accept move and update best solution.
8.              Update tabu list and aspiration level.
9.              Increment iteration number.
10.         Else
11.             If C(s*) < AL Then
12.                 Accept move and update best solution.
13.                 Update tabu list and aspiration level.
14.                 Increment iteration number.
15.             EndIf
16.         EndIf
17.     EndFor
End. (* Tabu_Searcb *)
```

Figure 3: Algorithmic description of tabu search (TS).

algorithm, each processor is responsible for its own search and the communication with other processors. The second dimension is **Control and communication type** where the algorithm can follow a *rigid synchronization (RS)*, a *knowledge synchronization (KS)*, a *collegial (C)*, or a *knowledge collegial (KC)* strategy. RS and KS correspond to synchronous operation mode where the process is forced to exchange information at specific points; C and KC correspond to asynchronous operation modes where communication occurs at regular intervals. Collegial approaches exchange more information than non-collegial ones. The third dimension is **search differentiation** where the algorithm can be *single point single strategy (SPSS)*, *single point different strategies (SPDS)*, *multiple points single strategy (MPSS)*, or *multiple points different strategies (MPDS)* [26].

In [27], a mechanism for parallelizing *tabu search* by dividing neighborhood examination among a number of slaves for solving flow shop sequencing problem was presented. A master process is used to send an initial solution to all slaves. At each iteration, each one of the slaves examines part of the neighborhood and reports the best move to the master process. The master process chooses the best move among all and sends it to all slaves as the next move to be performed if it is not tabu. The process then continues for a fixed number of iterations or until no improvement is observed for a given number of iterations. The algorithm presented uses *domain decomposition*

strategy. It is a *1-control*, *RS*, *SPSS* algorithm.

Garcia et. al [28] presented a parallel implementation of *tabu search* for vehicle routing problem. In their work, a master process applies TS and calls slaves which (with the master) investigate the neighborhood of the current solution. Each process identifies its best move and sends it to the master. The master process selects a set of the best moves and broadcasts them to all slaves. Processes exchange only sequence of moves rather than exchanging complete solutions which causes redundant communication overhead. The algorithm uses *domain decomposition* strategy. It is a *1-control*, *RS*, *SPSS* algorithm.

In [29, 30], evolution principles were included to improve parallel Tabu Search. In the given strategy, short term memory *tabu search* was applied on a set of machines. After a specific number of iterations, each machine exchanges best solutions with its neighbors. At each machine, if the received solution is better than the local best, it replaces it. The algorithm uses *multi-search threads* strategy. It is a *p-control*, *C*, *MPSS* algorithm.

Niar and Freville [31] proposed a parallel TS algorithm for the 0-1 multidimensional knapsack problem. In their algorithm, they had a master process that generates initial solutions for slaves. The initial solution for process i is taken as its previous best solution except in two cases: (i) if the quality of the best solution is less than a fraction α of the overall best solution or, (ii) if the best solution of process i has not been modified for a given number of iterations. For each slave, the master generates a different strategy where the strategy is represented by the tabu list size, the number of local iterations and the number of successive drops at each iteration. The algorithm uses *multi-search threads* strategy. It is a *p-control*, *RS*, *MPDS* algorithm.

In [32], a parallel *tabu search* algorithm for voltage and reactive power control in power systems was presented. Two schemes were implemented in that work. In the first one, the neighborhood was decomposed for parallel processing at each iteration. This is a *domain decomposition strategy*. The algorithm is *1-control RS SPSS* search. In the second scheme, a *multi-search threads* strategy is followed, where *tabu search* was replicated with various tabu list sizes for different processes. The algorithm is *p-control*, *RS*, *SPDS* search.

In [33], TS was applied to quad-partitioning VLSI Macro-cell placement problem. The objective was to minimize the interconnection length which in turn minimizes the delay. The results showed that using fuzzy cost function provided up to 43% improvements in the cost.

The work described in the paper reports a parallelization of TS algorithm on a network of heterogeneous workstations using PVM [16]. The problem tackled is the standard cell VLSI placement, a hard constrained multiobective optimizatoin problem with an exponential search space.

5 Parallel Tabu Search for Standard-Cell Placement

The proposed parallel Tabu search algorithm consists of three types of processes: (i) a master process, (ii) Tabu Search Workers (TSWs), and (iii) Candidate list Workers (CLWs). The algorithm is parallelized on two levels simultaneously. The upper one is at the TS process level where a master starts a number of TSWs and provides each with the same initial solution (*multi-search threads*). The lower level is the *Candidate List* construction level (local neighborhood search) where each TSW starts a number of CLWs, this is *functional decomposition*. The general structure of the proposed parallel algorithm is shown in Figure 4.

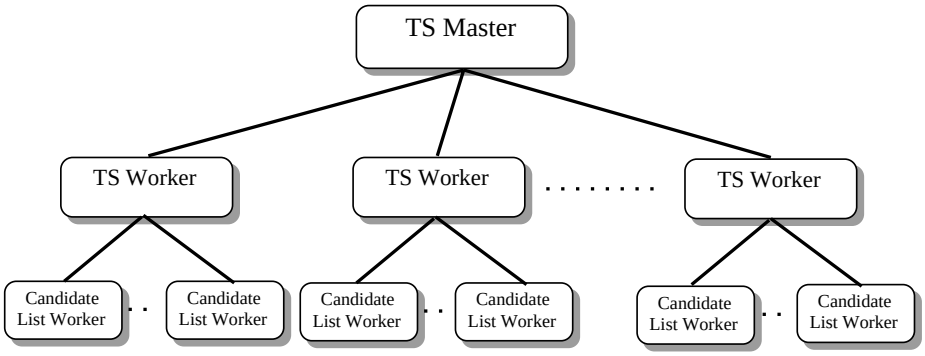


Figure 4: General structure of tabu search parallel implementation.

The parallel search proceeds as follows. The master initiates a number of TSWs to perform TS starting from the given initial solution. A TSW gets all parameters and the initial solution from the master. It then performs a diversification step where each TSW diversifies with respect to a different subset of cells so as to enforce that TSWs don't search in overlapping areas. Diversification is performed by moves done within the TSW range to a specific depth such that a different initial solution is used at each TSW. Then each TSW starts a number of CLWs to investigate the neighborhood of the current solution (initial solution after diversification). It sends the parameters and the initial solution to each CLW. It also gives each CLW a range of cells to search the neighborhood with respect to those cells. For every move it makes, the CLW has to choose one of the cells from its range and the other cell from anywhere in the whole cell space. Therefore, the probability that two CLWs perform the same move is equal to $\frac{1}{(n-1)^2}$ where n is the number of cells. The probability that more than two CLWs select the same two cells is 0. This means that the probability that k CLWs make the same move is eliminated completely if $k > 2$.

Each CLW makes a compound move of a predetermined depth and keeps computing the gain. If the current cost is improved before reaching the

```
Algorithm Parallel_TS_Master_Process;
    N_i  :  Number of iterations.
    X    :  Set of feasible solutions.
    bs   :  Current best solution.
    bc   :  Current best cost.
    TL   :  Tabu list.
    N_w  :  Number of workers.
1.  Start with an initial feasible solution bs ∈ X.
2.  Initialize TL and bc.
3.  Spawn N_w TSW workers to perform Tabu Search.
4.  Send(bs, TL, bc) to all TSWs.
5.  For N_i Do
6.          Wait for best cost from all workers.
7.          Ask for bs and TL from the worker
            that has the overall best.
8.          Receive(bs, TL).
9.          Update bc.
10.         Send(AL, bs, TL, bc) to all workers except sender.
11.         Increment iteration number.
12. EndFor
End. (*Parallel_TS_Master_Process*)
```

Figure 5: Algorithmic description of master process of parallel TS.

maximum depth, the move is accepted without further investigation. After finding the compound move that improves the cost the most (or degrades it the least), the CLW sends its best solution to its parent TSW. The TSW selects the best solution from the CLW that achieves the maximum cost improvement (or the least cost degradation). It then checks if the move is tabu. If it is not, it accepts it. Otherwise, the cost of the new solution is checked against the *aspiration criterion* and the process continues for a number of local iterations. At the end of the local iteration count, each TSW sends its best cost to the master process. The master gets the overall best solution and broadcasts it to all TSWs and the process continues for a fixed number of global iterations. The completion of all iterations by the TSWs and selection of new current solution by the TS master is considered one global iteration. The TS iterations executed by each TS worker are called local iterations

The processes described in Figures 5, 6, and 7, work together to get a high quality solution with minimum communication between them. A TSW process and a CLW process exchange only the best solution between them while the master and TSW exchange the best solution as well as the associated tabu list.

The overall execution flow of the various processes is abstracted in Figure 8. The effect of the number of TSWs and CLWs on the quality of solution and the execution time of the algorithm is discussed later in Section 6. Although the proposed parallel algorithm has been applied to the VLSI standard cell placement problem, it may very easily be applied to any optimization problem where functional decomposition will be beneficial.

```
Algorithm TSW;
        GI    :  Number of global iterations.
        LI    :  Number of local iterations.
        cs    :  Current solution.
        bs    :  Current best solution.
        bsi   :  Best solution sent by CLW (i).
        TL    :  Tabu list.
        AL    :  Aspiration Level.

1.      Receive(cs, TL) and a range from master.
2.      For GI Do
3.              Perform a diversification step.
4.              For LI Do
5.              Send cs and a unique range of cells to each CLW.
6.              Receive bsi from all CLWs.
7.              Check if the best bsi is tabu.
8.                      if no, update cs to the best bsi.
9.                      if yes, check if it satisfies aspiration criterion.
10.                             if yes, update cs to the best bsi else cs is not changed.
11.             EndFor
12.             If the master asks for bs  Then
13.                     Send(bs, TL) to master.
14.             Else
15.                     Receive(bs, TL) from master.
16.                     cs = bs.
17.             EndIf
18.     EndFor
End. (*TSW*)
```

Figure 6: Algorithmic description of TSW.

```
Algorithm CLW;
        LI    :  Number of local iterations.
        cs    :  Current solution.
        TL    :  Tabu list.
        AL    :  Aspiration Level.

1.      Receive(cs, TL) and a range of cells from TSW.
2.      For LI Do
3.              Try m random pairs of cells of which one has to belong to the range.
4.              Perform best swap and update cs.
5.              Send cs to TSW.
6.              If the TSW asks for cs  Then
7.                      Send(cs, TL) to TSW.
8.              Else
9.                      Receive(cs, TL) from TSW.
10.             EndIf
11.     EndFor
End. (*CLW*)
```

Figure 7: Algorithmic description of CLW.

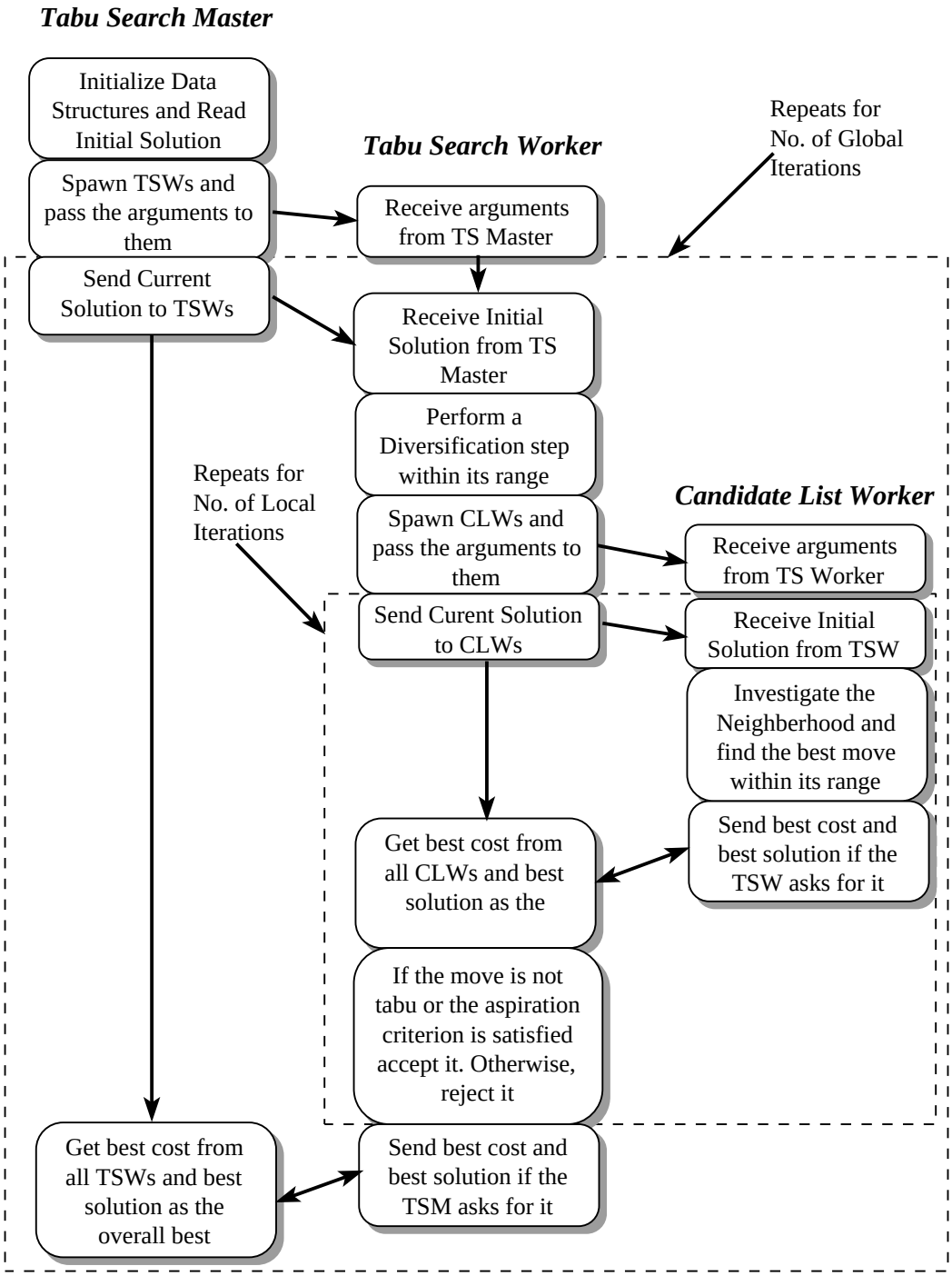


Figure 8: A scenario of the *master*, the *TSW* and the *CLW* processes.

5.1 The Algorithm in a Heterogeneous Environment

Normally, a network of workstations is composed of heterogeneous machines. Heterogeneity can be of various types such as the machine architecture, data format, computational speed, network type, machine load, and network load. PVM can take care of machine architecture heterogeneity and data format conversion.

In our implementation of parallel *tabu search*, we account for speed and load heterogeneity by letting the master receive the best cost from any TSW that has finished the local iterations. Once the number of TSWs that gave their best cost to the master reaches half the total number of TSWs, the master sends a message to all other TSWs forcing them to report whatever best cost they have achieved. TSWs check for such a message in their buffers periodically (every 10 iterations). Once they receive the message, they kill the currently running CLWs and report to the master their best achieved costs.

The same approach is followed in the communication between TSWs and their own CLWs that check for a message from their parents frequently. That message either kills them, if it is the TS master that is asking the TSW to report, or asks them for their best achieved solutions if half of the CLWs have reported their best. This approach is followed in order to account for the heterogenity in workstation's speeds and and loads as well as the varying network load.

Experiments are conducted on three different speed levels of machines and four different architectures. These architectures are IPX/SPARC, SparcStation 10, LX/SPARC and UltraSparc 1. All machines have the same operating system (Solaris 2.5).

5.2 Classification of the Proposed Algorithm

As mentioned earlier, the algorithm is parallelized on two levels simultaneously. The upper one is at the *tabu search* process level where a master starts a number of TSWs and provides them with the same initial solution. This is a *multi-search threads* approach where each TSW performs its own search. The lower level is the *Candidate List* construction level where each TSW starts a number of CLWs.. This level belongs to the strategy of *functional decomposition* because CLWs are spawned only to investigate the neighborhood of the current solution.

The algorithm falls into *p-control* class at the higher parallelization level because the search control is distributed among all TSWs. The lower level parallelization belongs to the *1-control* class because the TSW controls the search done by its CLWs.

On the *control and communication type* dimension, the algorithm follows *rigid synchronization* because the master waits for its children or stops them. It is a *multiple points single strategy (MPSS)* search on the *search differentiation* dimension because TSWs diversify from the initial solution

at each global iteration using the diversification scheme proposed by Kelly et. al [13].

6 Experiments and Discussion

In this section, we present and discuss various experiments that are performed using the proposed parallel *tabu search* algorithm for VLSI standard cell placement. In Sections 6.1 and 6.2, we study the effect of the degree of low-level and high-level parallelization on the algorithm performance, namely quality of best solution and speedup. The definition of speedup for non-deterministic algorithms such as TS is different from that used for deterministic constructive algorithms. For this category of algorithms, speedup is defined as follows

$$Speedup_{(n,x)} = \frac{t_{(1,x)}}{t_{(n,x)}} \tag{2}$$

where $t_{(1,x)}$ is the time needed to hit an x-quality solution using one CLW (or TSW) and $t_{(n,x)}$ is the time needed to hit the same solution quality using n CLWs (or TSWs). $Speedup_{(n,x)}$ in this case can be greater than n because investigating the neighborhood with n CLWs (or making n independent searches) can cause the stochastic search to hit an x-quality solution more than n times faster. In Section 6.3 we study the effect of diversification performed by TSWs by comparing the results obtained with and without diversification.

In our experiments, we used eight *ISCAS-89* benchmark circuits. Table 1 shows the number of cells, the number of I/O (input/output) pins, the number of rows, the layout height, the routing channel height, the optimum wire length, the optimum delay, and the optimum area for all circuits used.

Circuit			Layout			Optimal Costs		
Name	Cells	IOs	Rows	LH	Avg. RCH	O_{wl}	O_{delay}	O_{area}
highway	56	11	3	284	55.0	7156	3.91	512
fract	149	24	5	556	66.5	28207	7.97	784
c499	283	73	6	750	80.4	43583	8.17	1176
c532	395	43	7	703	49.5	64674	17.83	1152
c880	784	86	9	1034	64.0	123616	16.8	1824
c1355	1451	73	13	1557	66.9	273138	13.62	2304
struct	1952	64	15	2102	88.0	432096	13.54	3280
c3540	2243	72	17	2480	93.4	500157	23.26	3096

Table 1: Characteristics of circuits and layouts used. ($LH =$ layout heights and $Avg.\ RCH =$ average routing channel height in microns).

6.1 Effect of Degree of Low-level Parallelization

In this experiment, different number of CLWs are tried, from 1 to 4, for each circuit. The change in the best solution quality is monitored as the number of CLWs is changed. All other algorithm parameters are fixed. The number of TSWs is 4 in all experiments. Twelve machines are used as a parallel virtual machine. The number of global iterations (GIs), number of local iterations (LIs), the number of cells swaps per move the depth of compound move (d), diversification depth, and the T-tenure, for all the circuits, are fixed as given in Table 2.

Name	Cells	GI	LI	m	d	Div_Depth	T_Tenure
highway	56	500	37	7	3	7	7
fract	149	500	61	12	3	18	7
c499	283	160	70	17	4	70	8
c532	395	120	100	20	4	100	9
c880	784	120	200	28	5	200	9
c1355	1451	80	120	9	2	60	9
struct	1952	80	120	9	2	60	10
c3540	2243	80	120	9	2	60	10

Table 2: Parameters of experiments (GI = global iterations and LI = local iterations).

Figure 9 shows the effect of changing the number of CLWs on the best solution quality for 6 circuits. For most of the circuits, it is clear that increasing the degree of low level parallelization is beneficial. For *highway*, the circuit size is small. That makes adding CLWs beyond 2 not useful. Using 4 CLWs gives bad solution quality. A reason for this is the restriction imposed on choosing cells when more CLWs are used (as mentioned in Section 5). The cost used here is the fuzzy cost mentioned in Section 1, based on the optimum values shown in Table 1 for wire length, delay and area.

Figure 10 shows the time needed to achieve a specific solution quality for all circuits. This quality is chosen according to the best quality achieved by the worst run. Adding more CLWs resulted in reaching better solutions in less time except for *highway* and *c499* where increasing the number of CLWs to more than 2 was counterproductive and was causing communication overhead more than speeding up the search.

Figure 11 shows the speedup achieved in reaching a specific solution quality for all circuits. It is clear from the figure that in most of the experiments, as the number of CLWs increases from 1 to 4, the speedup increases. The sharpness of the speedup increase depends on the circuit size and the goodness of the initial solution. For *c532*, the initial solution is too far from the best reached. As a result, increasing the number of CLWs results in a sharper change in the speedup. For *c1355*, the circuit size is moderate and

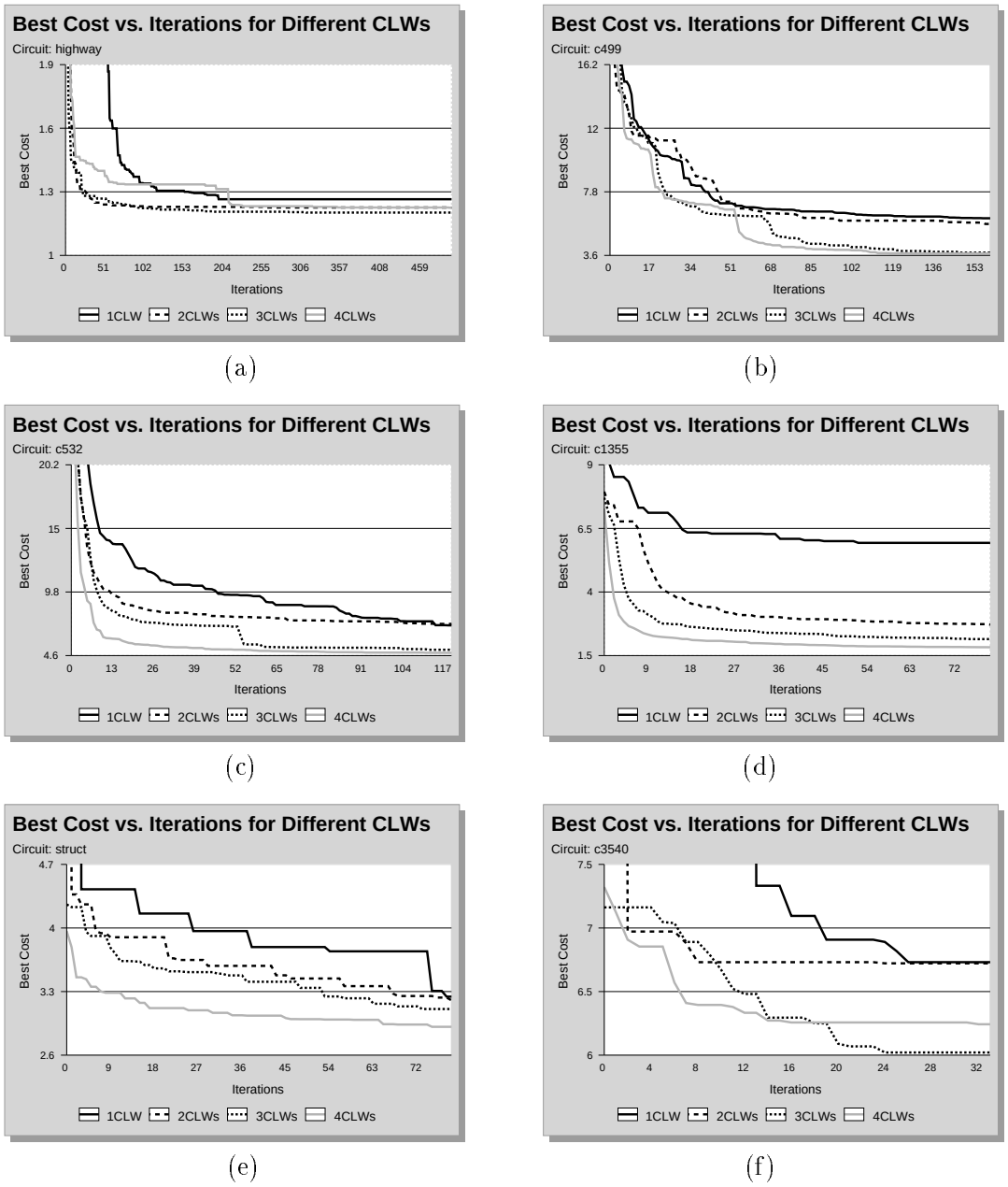
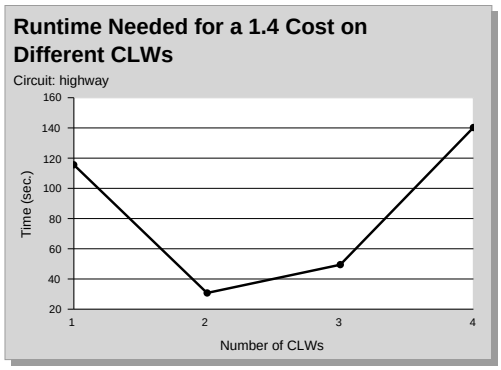
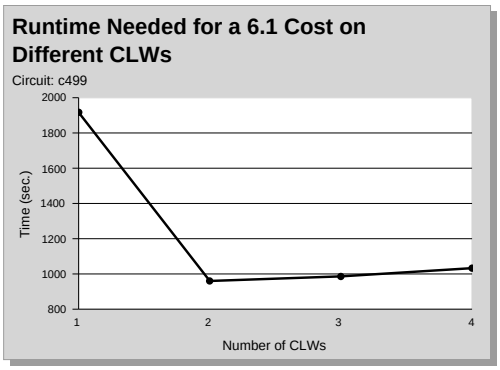


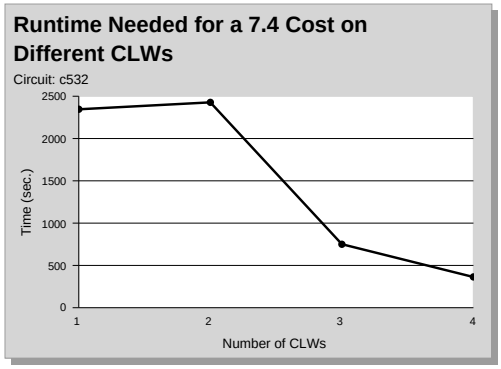
Figure 9: Effect of number of CLWs on solution quality.



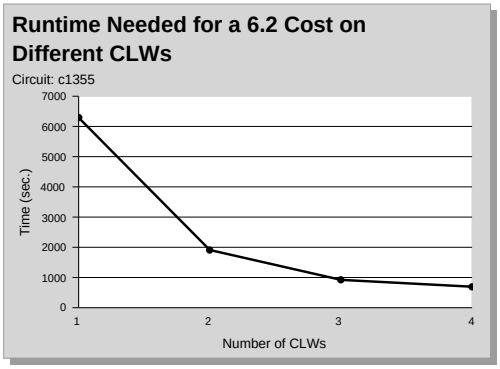
(a)



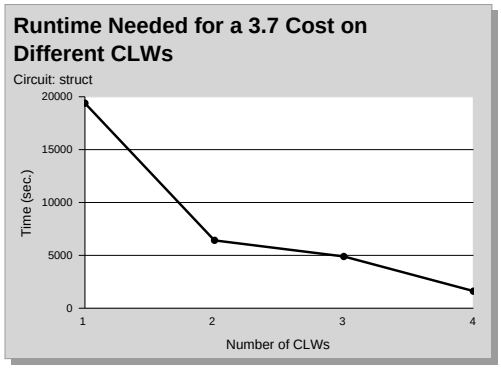
(b)



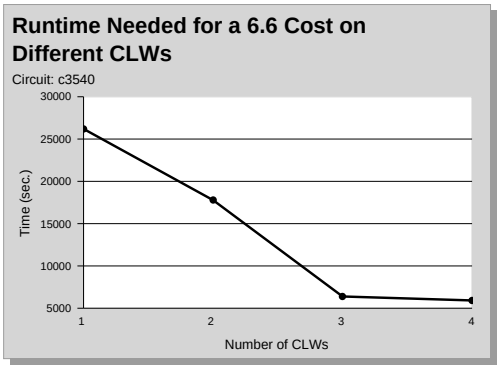
(c)



(d)



(e)



(f)

Figure 10: Runtime needed to achieve a solution of cost less than x for different numbers of CLWs.

the initial solution is not that far from the best reached as in the case of *c532*. Because of communication overhead, the rate of change in the speedup goes down as the number of CLWs is increased. In all experiments except *highway* and *c499*, the critical point, where the speedup starts to degrade, is not reached but it is clear in some curves that it is being approached. For *highway* and *c499*, the critical point occurred at 2 CLWs because the 2 circuits are small.

6.2 Effect of Degree of High-level Parallelization

In this experiment, different numbers of TSWs are tried, from 1 to 8, for each circuit. The change in the best solution quality is monitored as the number of TSWs is changed. The number of CLWs per TSW is fixed to 1 in all experiments. As mentioned earlier, 12 machines are used as a parallel virtual machine with the same parameters as shown in Table 2.

Figure 12 shows the effect of changing the number of TSWs on the best solution quality for all circuits. For *c499* and *c1355*, it is clear that using more TSWs gives better solution quality in all iterations. For *highway*, the circuit size is small. This makes adding TSWs beyond 4 not useful. For *c532*, best results are obtained with 4 TSWs. Increasing the number of TSWs to 8 did not lead to any noticeable improvement. Recall that no functional decomposition is done (only 1 CLW per TSW). Relying on multi-thread search alone may not always result in improvement of speedup.

Figure 13 shows the time needed to achieve a specific solution quality for all circuits. This quality is chosen according to the best quality achieved by the worst strategy. Adding more TSWs proved to be beneficial with respect to runtime except for *highway* and *c532* where increasing the number of TSWs to more than 4 was causing communication overhead more than speeding up the search.

Figure 14 shows the speedup achieved in reaching a specific solution quality for all circuits. For *highway* and *c532*, the critical point, occurred at 4 TSWs. Adding more TSWs degraded the speedup. For the other three circuits, the critical point was approached but not reached.

In general, increasing the number of CLWs performed better than increasing the number of TSWs, i.e., low-level parallelization seems to be more beneficial than high-level parallelization. This is due to the fact that a larger number of CLWs leads to a greater functional decomposition of the exploration of a solution neighborhood, which is the most time consuming component in VLSI placement. Further, increasing the number of TSWs meant more communication overhead. Therefore, too much increase in the number of TSWs can be counter-productive (See Figure 14, (a) and (c)). Recall that the algorithm was run on a cluster of workstations on an Ethernet segment and hence, inter-station communication can be very slow.

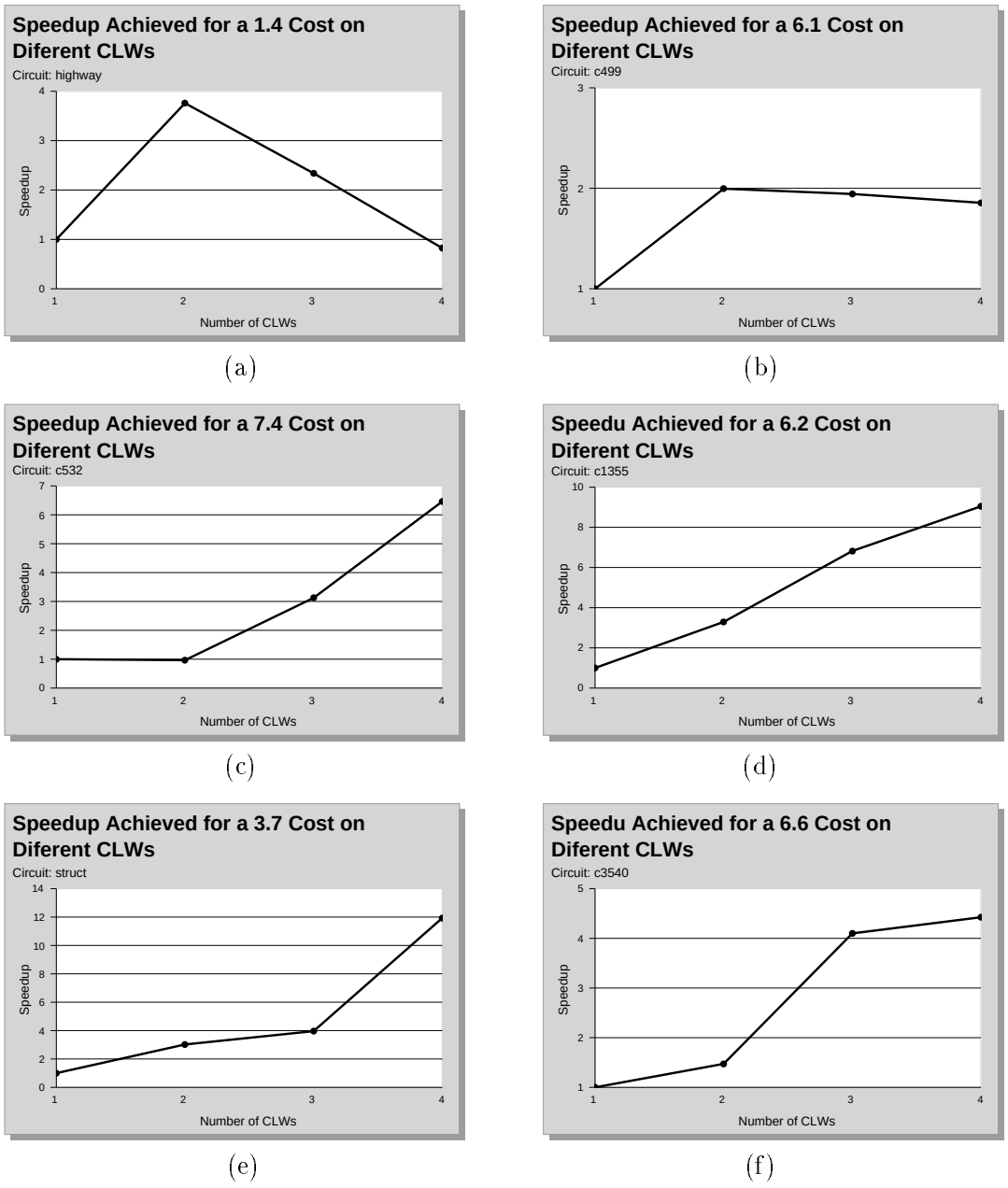


Figure 11: Speedup achieved in reaching a solution of cost less than x for different numbers of CLWs.

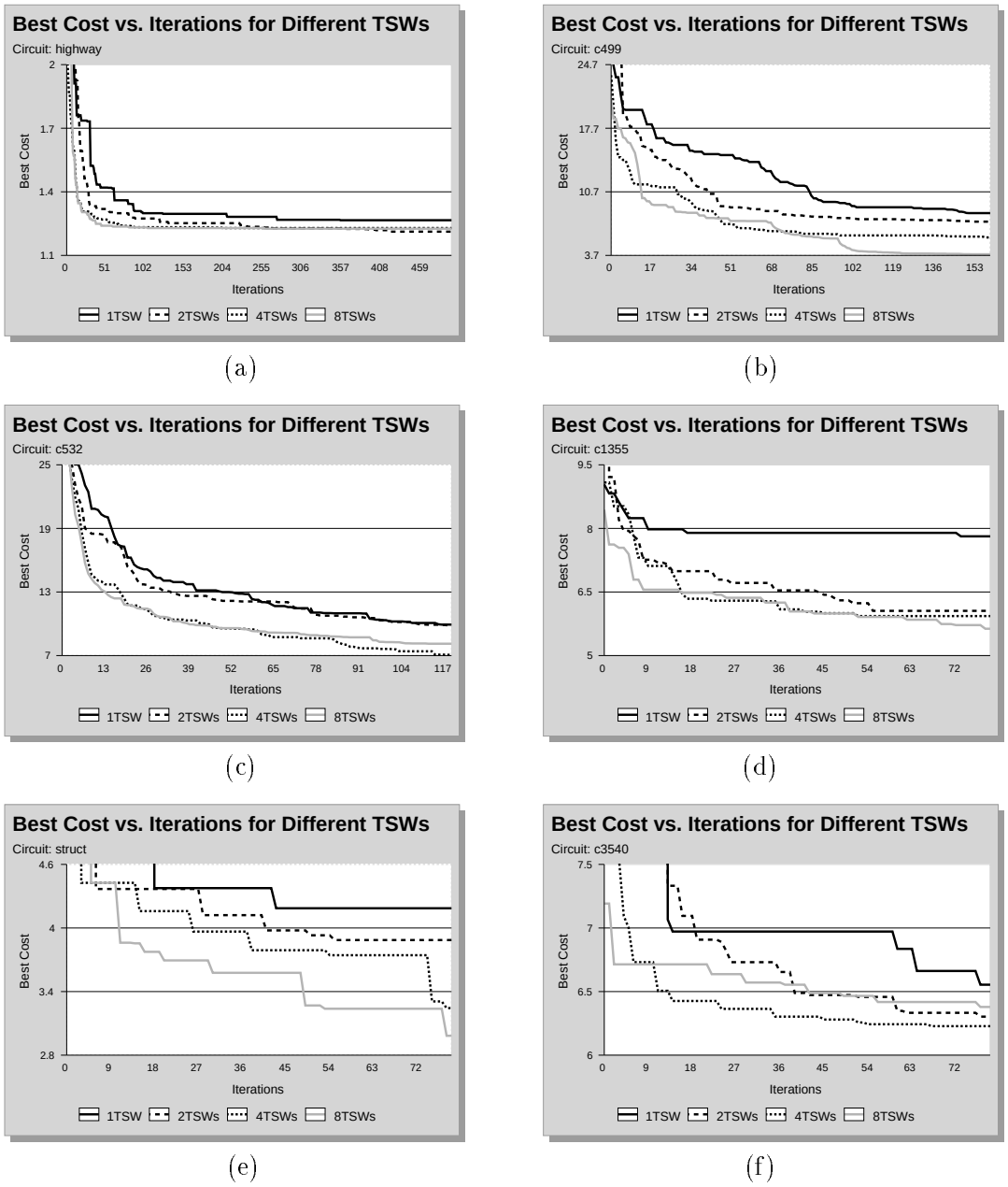


Figure 12: Effect of number of TSWs on the solution quality.

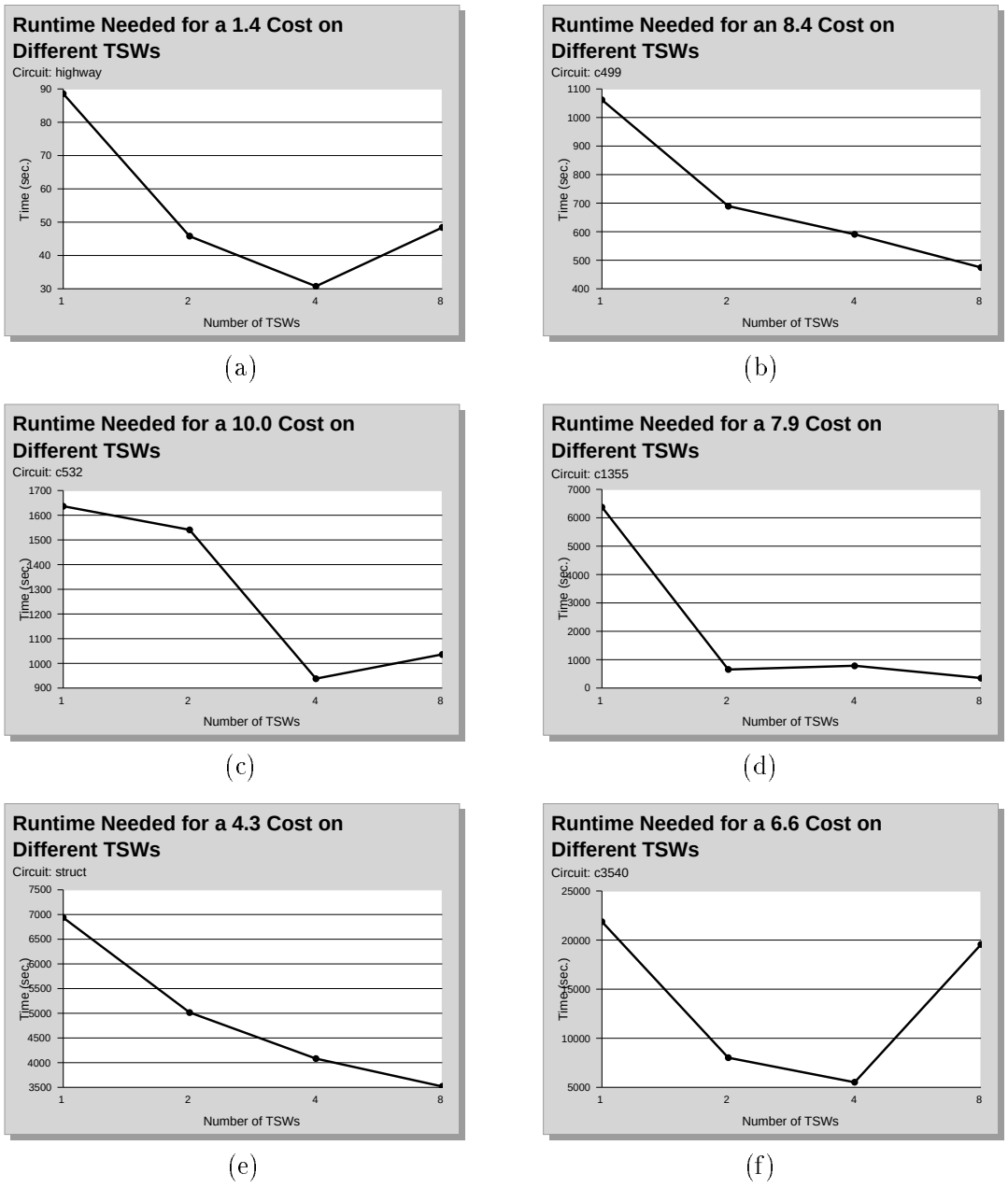


Figure 13: Runtime needed to achieve a solution of cost less than x for different numbers of TSWs.

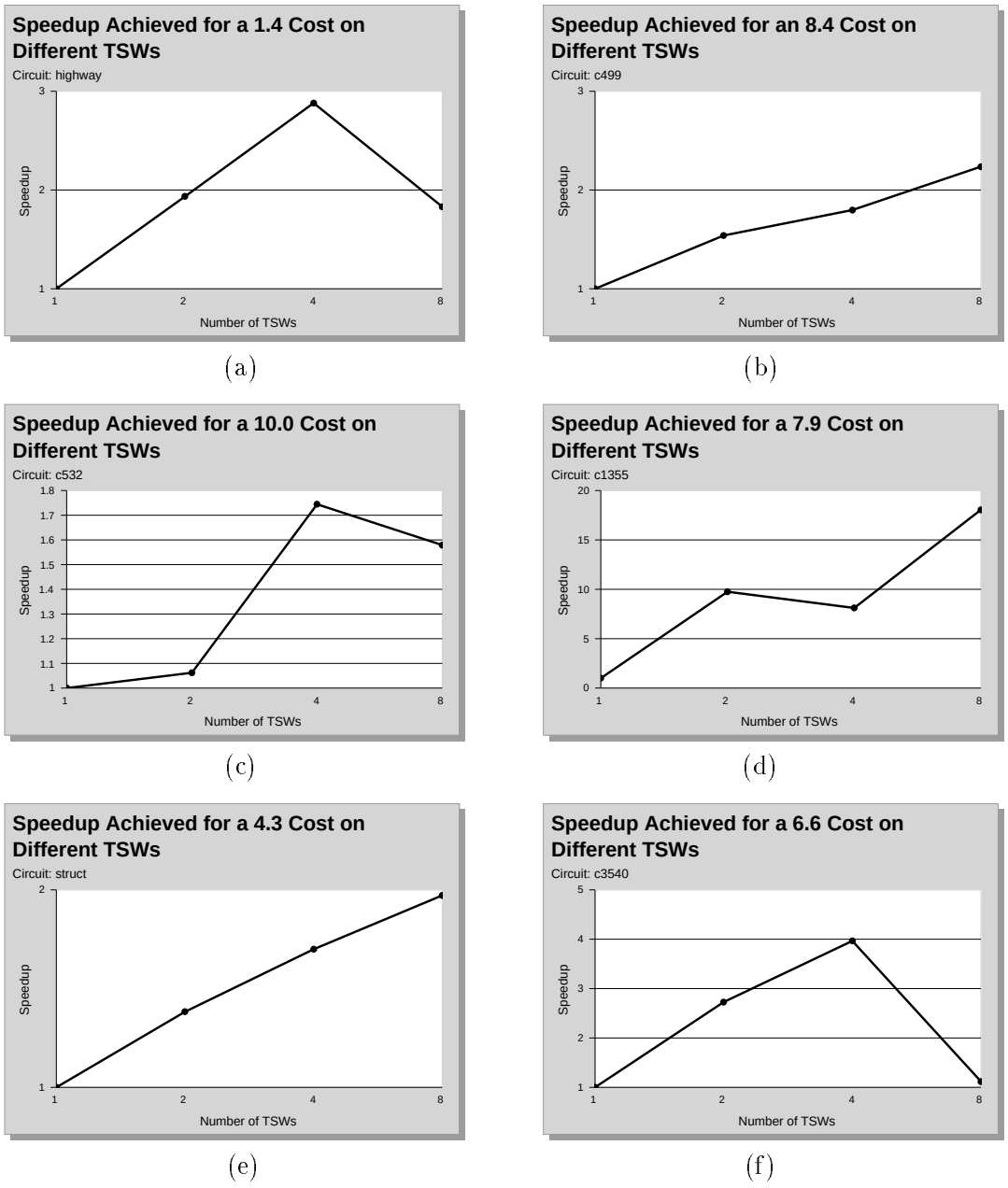


Figure 14: Speedup achieved in reaching a solution of cost less than x for different numbers of TSWs.

6.3 Effect of Diversification

In this experiment, we try to see the effect of the diversification step performed by the TSWs at the beginning of each global iteration. As mentioned earlier, the diversification step is performed to make every TSW investigate a different region of the search space.

Each TSW performs a number of successive moves up to a predefined diversification depth (see Table 2) on the solution received from the master TS process. At each move, the TSW tries m different swaps and picks the most improving (or least degrading) one. One of the cells to be swapped has to be from the range belonging to the TSW.

Figure 15 shows a comparison between two runs of four TSWs and one CLW per TSW. In one run, diversification is done to the diversification depth mentioned in Table 2. In the other run, no diversification is performed. It is clear from the figure, that the diversified run outperforms the non-diversified run significantly. For *struct*, the non-diversified run outperforms the diversified run in the early iterations because it is more greedy. However, in the long term, the diversified run always performs better.

The message conveyed in Figure 15 is that some diversification is always useful, and too much diversification without enough local investigation might mislead the search by making it jump from place to another without enough investigation any where. The only way to decide how much local investigation versus diversification is enough is through experiments.

Figure 16 shows the results of an experiment where the number of global iterations is decreased as the number of local iterations is increased for all circuits. For GI and LI refer to Figure 6. GI refers to the number of diversification steps, that is, the number of "distinct" solution sub-spaces investigated. On the other hand LI refers to the number of iterations executed in the particular sub-space. As we increase the number of global iterations and decrease the number of local iterations, we make more diversification and less local investigation. It is clear from the figure that no general conclusion can be made about the best number of global iterations versus local iterations. It all depends on the problem instance itself. This experiment is used as a guide for the most suitable number of local and global iterations that should be used to continue searching for the best achievable solution and to achieve the highest speed.

6.4 Accounting for Workstations Heterogeneity & Network Load

In this experiment, we try to see the effect of accounting for speed and load heterogeneity of various machines by performing two runs. In the first one (heterogeneous run), we run the algorithm while accounting for speed and load heterogeneity by making the master ask for best solutions from all TSWs once half of them complete all assigned iterations, and report their best to their parent. TSWs do the same by asking their CLWs to submit their best solutions once half of them report their best to the parent.

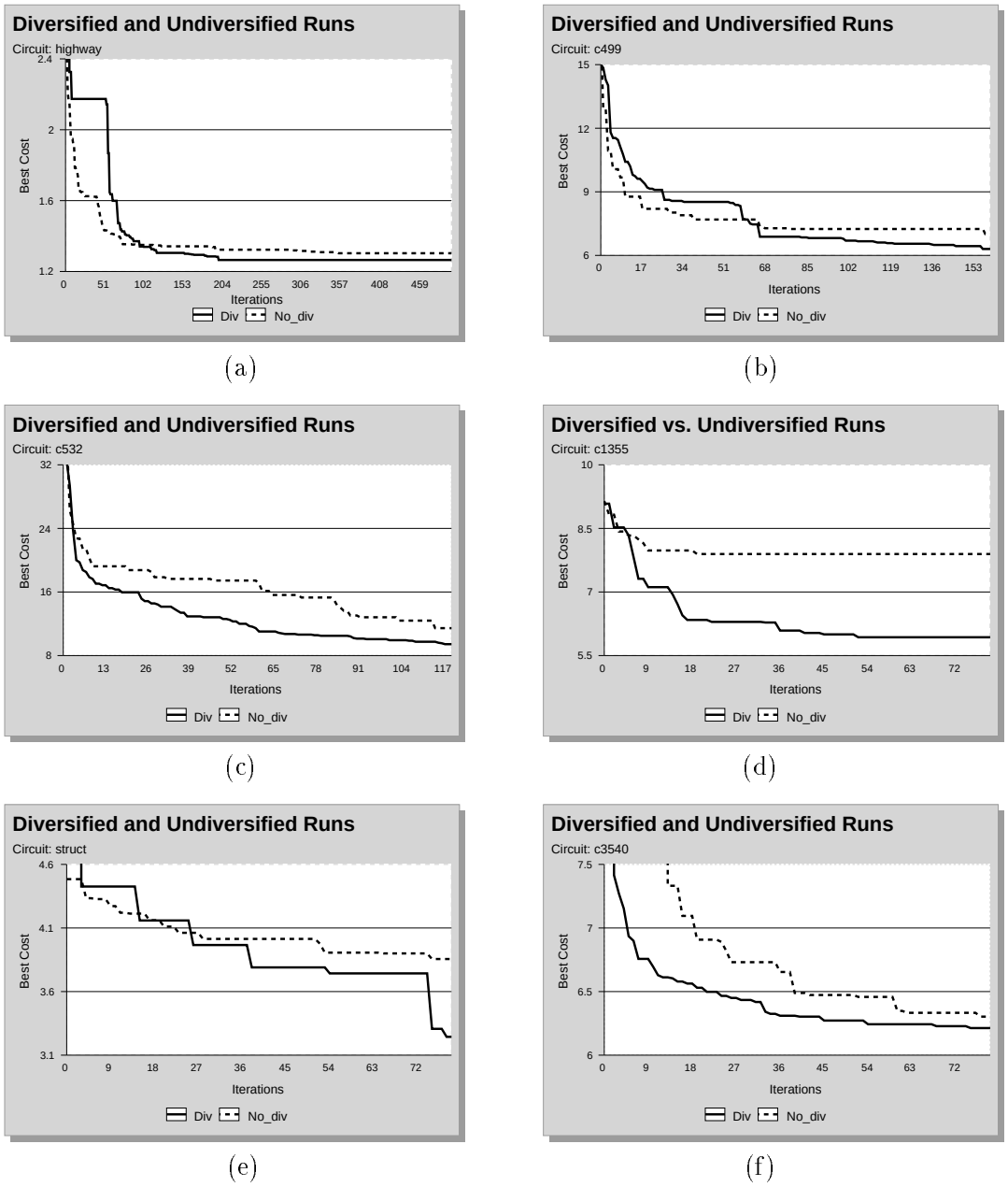


Figure 15: Effect of diversification.

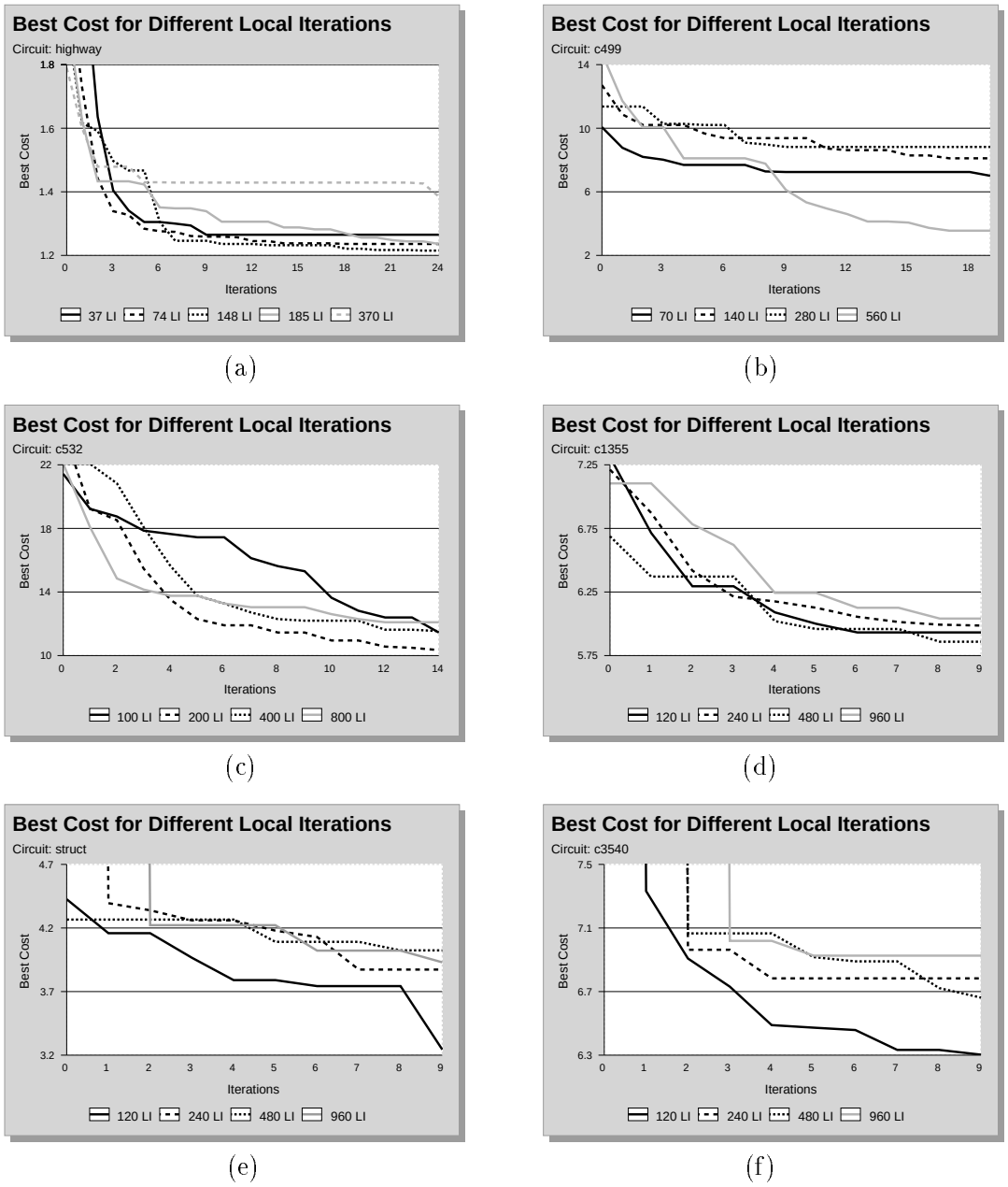


Figure 16: Local versus global iterations.

This is a knowledge collegial mode of operation with respect to the control and communication dimension. In the second run (homogeneous run), each parent waits for all its child processes to finish and return their new best. In all experiments we used twelve machines to make the Parallel Virtual Machine. These machines include seven high-speed machines (UltraSparc 1), 3 medium-speed machines (SparcStation 10), and 2 low-speed machines (LX/SPARC), all running the same operating system (Solaris 2.5) and interconnected by a 10BaseT Ethernet segment.

PVM takes care of distributing processes between machines. In both runs, we use 4 TSWs and 4 CLWs per TSW. The run that does not account for heterogeneity is supposed to give better solutions because the parent waits for all of its children to give their best solutions and does not force any one to stop searching because others have finished. However, since the number of global iterations is maintained the same for both cases, the heterogeneous run-time is expected to be far less than the homogeneous runtime.

Figure 17 shows the best quality of solution achieved versus global iterations for the homogeneous and heterogeneous runs. Except for one circuit (*c499*), we observed no noticeable difference in solution quality. For *c499*, there is a difference in solution quality but at the expense of larger runtime.

Table 3 shows the runtime needed for heterogeneous and homogeneous runs. The table clearly indicates that for all test cases the parallel implementation that accounts for heterogeneity results in a reduction of execution by a factor of 1.4 to almost 2.00 with respect to the homogeneous implementation. Figure 18 provides more accurate comparison between the homogeneous and heterogeneous implementations. It tracks the best cost achieved versus run time in seconds.

Circuit	Hom. Runtime	Het. Runtime	Improvement
highway	2316	1631	1.42
fract	11194	5626	1.99
c499	5722	3060	1.87
c532	8615	4839	1.78
c880	32361	19550	1.66
c1355	42560	27822	1.53
struct	76954	43332	1.78

Table 3: Runtime of homogeneous and heterogeneous runs in seconds.

Figure 18 shows that towards the end of the experiment, the heterogeneous run is doing either better than or at least as good as the homogeneous run, but never performs worse. For some circuits like *highway, c532* and *c1355*, the heterogeneous run keeps performing better than the homogeneous run throughout the execution. For *c499*, the heterogeneous run starts by performing worse and afterwards it outperforms the homogeneous run.

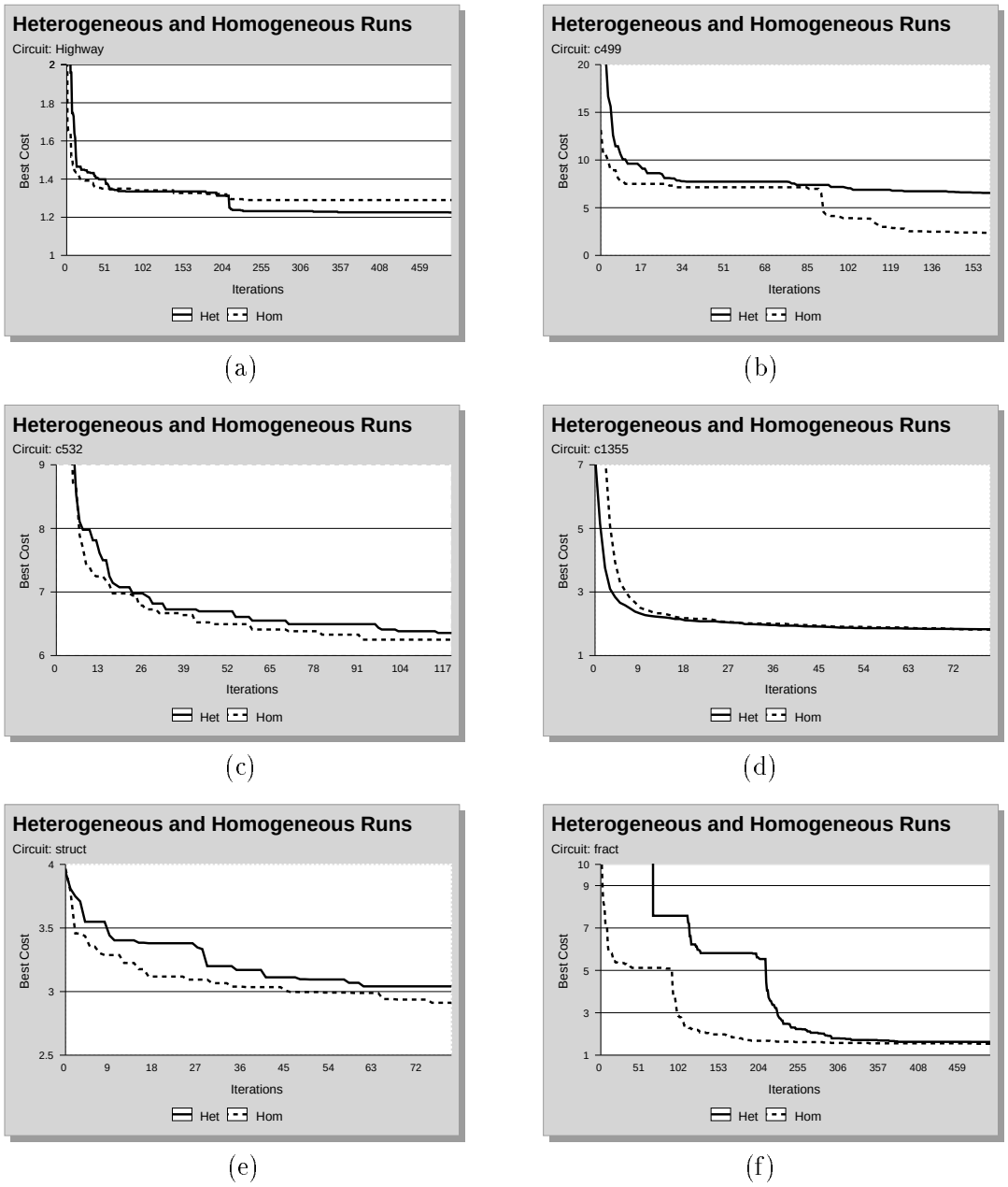


Figure 17: Heterogeneous versus homogeneous runs.

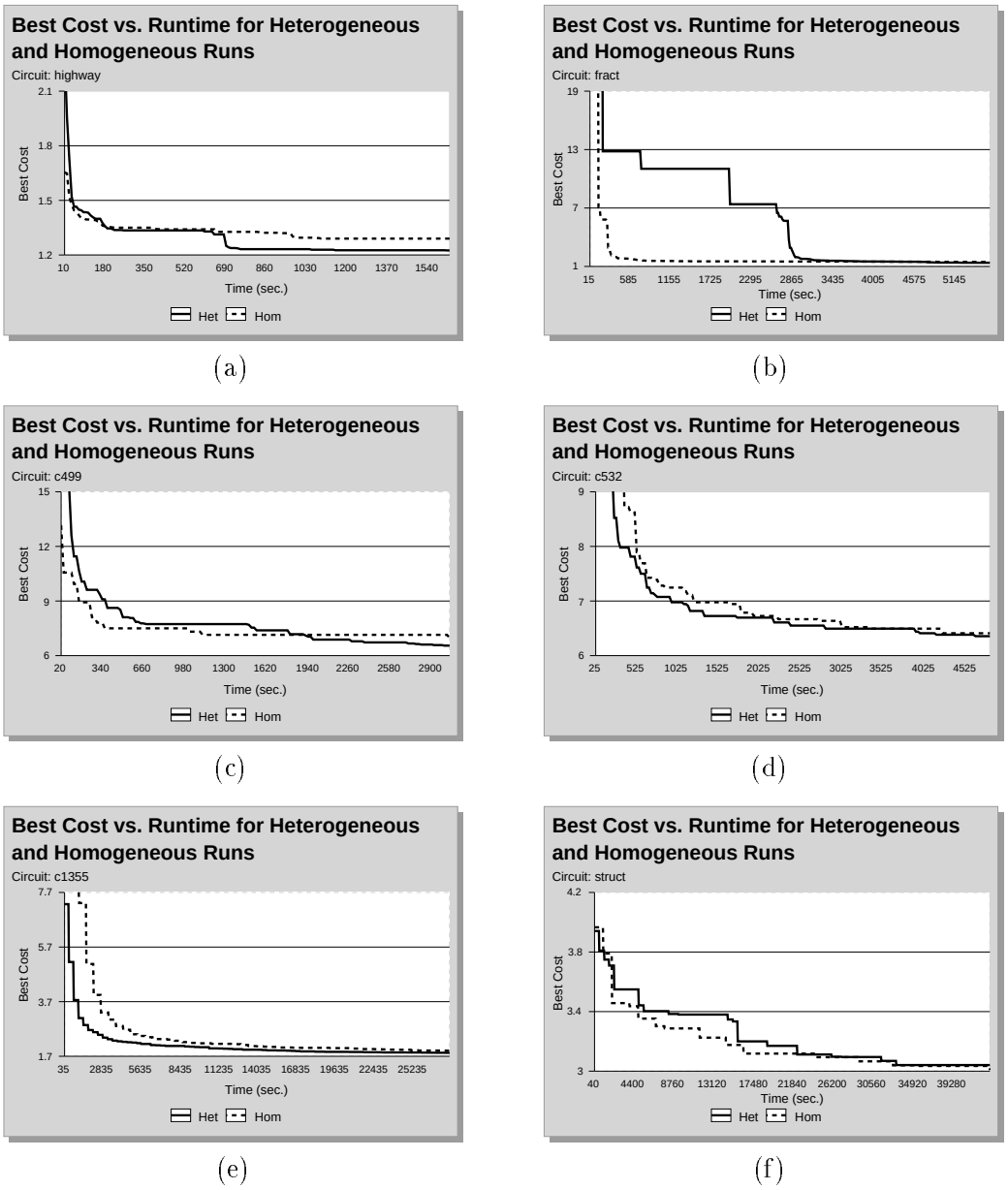


Figure 18: Best cost versus runtime for heterogeneous and homogeneous runs.

In another experiment, we tried to determine if it is useful (or not) to include slow machines in the parallel virtual machine, because the master keeps stopping them once the other machines report their best solutions. To see the contribution of the slow machines in our strategy, we conducted an experiment where one high-speed, one medium-speed, and one low-speed machine are used as a parallel virtual machine. A single TSW is spawned on each machine with one CLW per TSW. Once a TSW reports its best solution to the master, the master asks all others to stop and report their best solutions to it. By monitoring the number of solutions reported by each TSW within various cost ranges, we can tell which machine is contributing more to the search with useful results. Figure 19 shows the results of the experiment ran on *c499* for 500 global iterations. The results show that the contributions of the three machines are nearly equal in all cost ranges. This behavior is attributed to the non-deterministic nature of the search and to the synchronization step performed by the master process at the end of each global iteration.

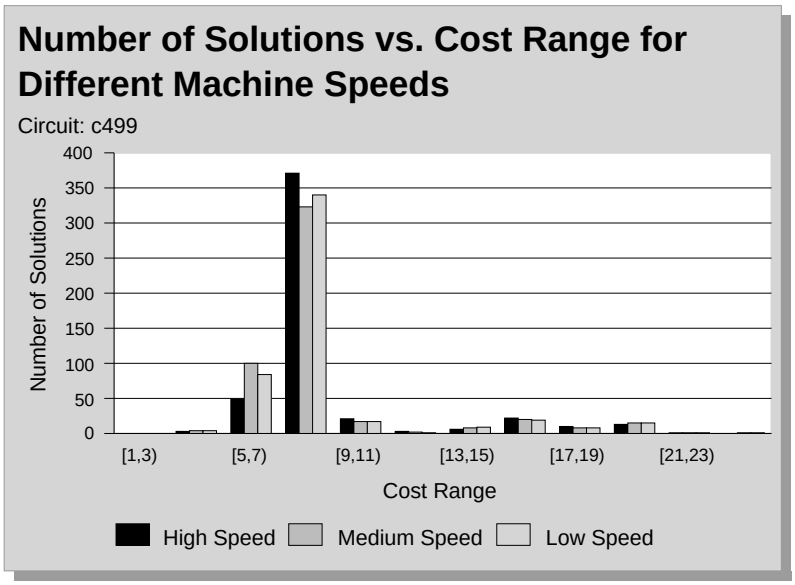


Figure 19: Number of solutions provided by machines of different speeds within various solution ranges.

7 Summary and Conclusion

In this paper, we presented the parallelization of tabu search and its implementation using PVM on a network of 12 heterogeneous SUN workstations. These workstations differ in speed and architecture: they are IPX/SPARC, SparcStation 10, LX/SPARC and UltraSparc 1, all running Solaris 2.5. Two parallelization strategies, functional decomposition and multi-search threads

strategy are integrated. Domain decomposition strategy is also implemented probabilistically. Experiments were conducted on several real instances of the VLSI cell placement problem.

The goal of parallelization is to speedup the search and to improve solution quality. Observations support that both parallelization strategies are beneficial, especially for large test cases, with functional decomposition producing slightly better results. Below we summarize our observations from extensive experiments carried out on circuits of various sizes.

- For most test circuits, increasing the degree of low level parallelization (functional decomposition) was beneficial and was dependent on the size of the circuit. Functional decomposition seems to be more beneficial for bigger circuits which have larger local and global search spaces.
- In order to achieve a specific solution quality, for all circuits, adding more CLWs resulted in reaching better solutions in less time. For small circuits, increasing the number of CLWs to more than 2 made the communication overhead offset the gain in speed.
- For large circuits, using more TSWs (instantiations of more search threads) resulted in improved solution quality in all runs, and resulted also in lower run-time requirement for achieving the same quality of solution. That is, high level parallelism is beneficial for large circuits. However, in general the most effective strategy seems to be a mix of high and low level parallelization. Low level or high level parallelization alone is not as effective.
- Speedup and load heterogeneity are taken into account by making the master ask for best solutions from all TSWs once half of them have completed all iterations. All solutions then report their best to the master. This resulted in higher speedup.
- An interesting observation is that all machines (independent of their speeds/architecture) contribute nearly equally to the search process. This is attributed to the non-deterministic implementation and diversification step of TS, as well as the global synchronization step of the algorithm.

Acknowledgment: The authors would like to thank King Fahd University of Petroleum and Minerals for all the support provided. Hassan R. Barada would also like to thank Etisalat College of Engineering, P.O. Box 980, Sharjah, UAE, for support.

References

[1] Sadiq M. Sait and Habib Youssef. *Iterative Computer Algorithms and their Applications in Engineering*. IEEE Computer Society Press, 1999.

[2] F. Glover, E. Taillard, and D. de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41:3–28, 1993.

[3] Sadiq M. Sait and Habib Youssef. *VLSI Design Automation: Theory and Practice*. McGraw-Hill Book Co., Europe, 1995 (also co-published by IEEE press).

[4] C. Sechen and A. L. Sangiovanni-Vincentelli. Timberwolf3.2: A new standard cell placement and global routing package. *Proceedings of 23rd Design Automation Conference*, pages 432–439, 1986.

[5] A. Casotto, F. Romeo, and A. L. Sangiovanni-Vincentelli. A parallel simulated annealing algorithm for the placement of macro-cells. *IEEE Transactions on Computer Aided Design*, CAD-6(5):838–847, September 1987.

[6] S. A. Kravitz and R. A. Rutenbar. Placement by simulated annealing of a multiprocessor. *IEEE Transactions on Computer-Aided Design*, CAD-6(4):534–549, July 1987.

[7] J. P. Cohoon and W. D. Paris. Genetic placement. *IEEE Transactions on Computer-Aided Design*, CAD-6:956–964, November 1987.

[8] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer Aided Design*, 9(5):500–511, May 1990.

[9] R. Kling and P. Bannerjee. ESP: A new standard cell placement package using simulated evolution. *Proceedings of 24th Design Automation Conference*, pages 60–66, 1987.

[10] Sadiq M. Sait, Habib Youssef, and Ali Hussain. Fuzzy simulated evolution algorithm for multiobjective optimization of VLSI placement. In *Proceedings of IEEE International Congress on Evolutionary Computation, Washington D.C.*, pages 91–97, July 1999.

[11] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, USA, 1997.

[12] F. Glover. Tabu search fundamentals and uses. Technical report, Graduate School of Business Administration, University of Colorado at Boudler, June 1995.

[13] J. P. Kelly, M. Laguna, and F. Glover. A study of diversification strategies for the quadratic assignment problem. *Computers Ops Research*, 21(8):885–893, 1994.

[14] R. Hübscher and F. Glover. Applying tabu search with influential diversification to multiprocessor scheduling. *Computers & Operations Research*, 21(8):877–884, 1994.

[15] F. Glover. Tabu search: A tutorial. Technical report, Graduate School of Business Administration, University of Colorado at Boudler, February 1990.

[16] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. The MIT Press, Cambridge, Massachusetts, London, England, 1994.

[17] M. Lewis and R. Cline. PVM communication performance in a switched FDDI heterogeneous distributed computing environment. In *IEEE Workshop on Advances in Parallel and Distributed Systems*, pages 13–19, October 1993.

[18] P. Crandall, E. Sumithasri, and M. Clement. Performance comparison of desktop multiprocessing and workstation cluster computing. In *5th IEEE International Symposium on High Performance Distributed Computing*, pages 272–281, August 1996.

[19] G. Geist and V. Sunderam. The PVM system: Supercomputer level concurrent computation on a heterogeneous network of workstations. In *6th Distributed Memory Computing Conference*, pages 258–261, May 1991.

[20] E. Horvath, R. Shankar, and A. Pandya. A parallel algorithm for standard cell placement. In *Seattle International Joint Conference on Neural Networks*, pages 896–897, July 1991.

[21] P. Cheung, C. Yeung, S. Tse, C. Yuen, and W. Ko. A new optimization cost model for VLSI standard cell placement. In *IEEE International Symposium on Circuits and Systems*, pages 1708–1711, June 1997.

[22] L. A. Zadeh. Fuzzy Sets. *Information Contr.*, 8:338–353, 1965.

[23] L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transaction Systems Man. Cybern.*, SMC-3(1):28–44, 1973.

[24] L. A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences*, 8:199–249, 1975.

[25] Ronald Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions Systems, man, and Cybernetics*, 18(1):183–190, January 1988.

[26] T. Crainic, M. Toulouse, and M. Gendreau. Towards a Taxonomy of Parallel Tabu Search Heuristics. *INFORMS Journal of Computing*, 9(1):61–72, 1997.

[27] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 417:65–74, 1990.

[28] Bruno-Laurent Garica, Jean-Yves Potvin, and Jean-Marc Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, 21(9):1025–1033, November 1994.

[29] I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro. Improving search by incorporating evolution principles in parallel tabu search. In *Proc. of the first IEEE Conference on Evolutionary Computation- ICEC'94*, pages 823–828, June 1994.

[30] I. De Falco, R. Del Balio, and E. Tarantino. An effective parallel heuristic algorithm for the mapping problem. In *Proc. of Australian New Zealand Intelligent Information Systems Conference- ANZIIS'94*, pages 160–164, November 1994.

[31] Smail Niar and Arnaud Freville, editors. *A Parallel Tabu Search Algorithm For The 0-1 Multidimensional Knapsack Problem*. 11th International Parallel Processing Symposium, Apr. 1997.

[32] H. Mori and T. Hayashim. New parallel tabu search for voltage and reactive power control in power systems. In *Proc. of the 1998 IEEE International Symposium on Circuits and Systems- ISCAS'98*, pages 431–434, May 1998.

[33] Carol A. Mackey and Jo Dale Carothers. Performance-driven macrocell placement. In *IEEE Fifteenth Annual International Phoenix Conference on Computers and Communications*, pages 427–433, March 1996.