MULTILEVEL HEURISTICS FOR TASK ASSIGNMENT IN DISTRIBUTED SYSTEMS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Murat İkinci

June, 1998

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Cevdet Aykanat(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Atilla Gürsoy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Özgür Ulusoy

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

ABSTRACT

MULTILEVEL HEURISTICS FOR TASK ASSIGNMENT
IN DISTRIBUTED SYSTEMS

Murat İkinci
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Dr. Cevdet Aykanat
June, 1998

Task assignment problem deals with assigning tasks to processors in order to minimize the sum of execution and communication costs in a distributed system. In this work, we propose a novel task clustering scheme which considers the differences between the execution times of tasks to be clustered as well as the communication costs between them. We use this clustering approach with proper assignment schemes to implement two-phase assignment algorithms which can be used to find suboptimal solutions to any task assignment problem. In addition, we adapt the multilevel scheme used in graph/hypergraph partitioning to the task assignment. Multilevel assignment algorithms reduce the size of the original problem by collapsing tasks, find an initial assignment on the smaller problem, and then projects it towards the original problem by successively refining the assignment at each level. We propose several clustering schemes for multilevel assignment algorithms. The performance of all proposed algorithms are evaluated through an experimental study where the assignment qualities are compared with two up-to-date heuristics. Experimental results show that our algorithms substantially outperform both of the existing heuristics.

Key words: Task assignment, distributed systems, task clustering, multilevel task assignment methods, Kernighan-Lin Heuristic.

ÖZET

DAĞITIK SİSTEMLERDE ÇOK DÜZEYLİ
GÖREV ATAMA ALGORİTMALARI

Murat İkinci
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doç. Dr. Cevdet Aykanat
Haziran, 1998

Görev atama probleminin amacı bir dağıtık sistemdeki görevlerin işlemcilere yürütme ve iletişim giderlerinin toplamını en küçük yapacak biçimde atamaktır. Bu çalışmada, görevlerin iletişim zamanlarının yanı sıra yürütme zamanları arasındaki farkı da dikkate alan yeni bir topaklama yöntemi önerilmiştir. Bu topaklama yöntemi uygun atama yöntemleri ile birlikte her türlü görev atama problemine en iyiye yakın çözümler bulabilecek olan iki-evreli atama algoritmaları oluşturmak için kullanılmıştır. Bunlara ek olarak, çizge/hiperçizge parçalamada kullanılan çok düzeyli çizenek görev atama problemine uyarlanmıştır. Çok düzeyli atama algoritmaları görevleri birleştirerek asıl problemi küçültür, en küçük problem için bir baslangıç ataması bulur, sonra bu atamayı her düzeyde iyileştirerek asıl probleme doğru yansıtır. Bu çalışmada çok düzeyli atama algoritmaları için bir çok topaklama çizeneği önerilmiştir. Bütün önerilen algoritmalar iki güncel algoritma ile karşılaştırılmış ve başarımları bir deneysel çalışma ile değerlendirilmiştir. Deney sonuçları göstermiştir ki önerilen algoritmalar varolan iki algoritmadan da daha iyi çalışmaktadır.

Anahtar kelimeler: Görev atama, dağıtık sistemler, görev topaklama, çok düzeyli görev atama yöntemleri, Kerninghan-Lin algoritması

To my family

ACKNOWLEDGMENTS

I am very grateful to my supervisor, Assoc. Prof. Dr. Cevdet Aykanat for his invaluable guidance and motivating support during this study. His instruction will be the closest and most important reference in my future research. I would also like to thank Yücel Saygın who was always with me with his invaluable moral support, my family for their moral support and patience during the stressful moments of my work, and last but not the least, Ümit V. Çatalyürek, who was always ready for help with his priceless technical knowledge and experience.

Finally, I would like to thank the committee members Asst. Prof. Dr. Atilla Gürsoy and Asst. Prof. Dr. Özgür Ulusoy for their valuable comments, and everybody who has in some way contributed to this study by lending moral and technical support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to the great advances in VLSI technology and the advent of high speed communication links, there has been a rapid increase in the number of the distributed computing systems in the past few years. The assignment of tasks to processors is an essential issue in exploiting the capabilities of a distributed system. In a careless assignment, processors may spend most of their time communicating with each other instead of performing useful computations. The task assignment problem in distributed systems deals with finding a proper assignment of tasks to processors such that total execution and communication costs are minimized.

The problem was first introduced and solved by Stone [1]. Stone reduced the task assignment problem to multiway cut problem by which the optimal assignments can be found in polynomial time for two-processor systems. Unfortunately the task assignment problem is known to be *NP-complete* [2] for three and more processors systems in general. Stone extends his method to more than two processors. He examines an auxiliary two-processor problem where a certain processor is singled out and all other processors are merged into a new one. He then shows that the tasks assigned to single processor retain this assignment in some optimal solution. This method is effective when many tasks are assigned to single processor. But computational results show that this is not the case especially for large problems. Lo [3] uses this method to reduce the number of tasks to be assigned. Her algorithm then completes

the assignment by using a greedy approach.

For the general task assignment problem, efficient branch-and-bound algorithms such as presented by Chern et. al. [4] and Magirou and Milis [2] can be used to find the optimal assignments, but they are infeasible in terms of computation time. So, several heuristic based algorithms have been proposed to produce suboptimal assignments effectively. Most of those assignment algorithms use clustering approaches in which the highly interacting tasks are merged to reduce the original problem into a smaller and easier one. The assignment algorithms which use clustering approaches can be classified into two groups; *single-phase assignment algorithms* and *two-phase assignment algorithms*. In single-phase assignment algorithms such as presented by Magirou [5] and Kopidakis et. al. [6], the processors are also considered for clustering as well as tasks. In those algorithms, clustering a processor and a task effectively represents assignment of that task to that processor. Two tasks can be merged to form a new cluster but two processor is not considered for clustering. Two-phase assignment algorithms such as presented by Efe [7], Williams [8] and Bowen et.al. [9] consists of two consecutive phases as; clustering phase and assignment phase. In the clustering phase, the highly communicating tasks are merged to form new clusters, and those clusters are then assigned to processors according to a heuristic in the assignment phase. Traditional clustering algorithms do not consider the differences between the execution characteristics of clustered tasks. They usually tend to form clusters of highly communicating tasks. In those clustering algorithms, clustering of dissimilar tasks can not be avoided. In this work, we present a clustering scheme which considers the difference between the execution times of tasks as well as the communication costs between them.

Multilevel approaches [10] are widely used for graph/hypergraph partitioning problems. In this work, we adapted the multilevel scheme used in graph/hypergraph partitioning problem to the task assignment problem to find suboptimal solutions. In this scheme, the original task assignment problem is reduced down to a series of smaller task assignment problems by clustering tasks, and then an initial assignment is found for the smallest task assignment problem. This initial assignment is then projected back towards the original

problem by periodically refining the assignments. Since the original problem has more degrees of freedom, such refinements decrease the cost of assignments. A class of local refinement algorithms that tend to produce very good results are based on the Kernighan-Lin (KL) heuristic [11]. For task assignment problem, we exploit the refinement scheme presented by Fiduccia-Mattheyses (FM) [12] which is a commonly used variation of KL. In our case, FM, starting from an initial assignment, performs a number of passes until it finds a locally minimum assignment. Each pass consists of a sequence of task reassignments and may have a linear time complexity in terms of the graph size by using appropriate data structures.

The organization of the thesis is as follows. The formal definition of task assignment problem and previous work is presented in Chapter 2. In this chapter, we also give the key points for the motivation of this work. In Chapter 3, we present the proposed clustering and assignment schemes for two-phase task assignment approaches. A two-phase task assignment algorithm (AC2) which uses those clustering and assignment schemes is also presented in Chapter 3. A multilevel approach based on FM refinement along with the different clustering schemes is presented in Chapter 4. Finally, experimental results obtained by the proposed algorithms are summarized in Chapter 5.

Chapter 2

Problem Definition and Previous Work

In this chapter, we define the task assignment problem and we mention about the previous work carried out to solve it. At the end of this chapter, we give the basic motivation behind our work.

2.1 Problem Definition

Let's begin with the following model of task-processor system and try to find a task assignment that minimizes total execution and communication costs. Formally, consider a set of n heterogeneous processors labelled as $P = \{p_1, p_2, p_3, ..., p_n\}$ and a set of m tasks labelled as $T = \{t_1, t_2, t_3, ..., t_m\}$. From now on, indices h, i, j, k and ℓ will be used to represent tasks, whereas indices p, q, r will be used to represent processors.

Let's assume that we have a task interaction graph (TIG), $G = (T, E)$ whose nodes represent tasks. The edges of G represent the interactions between the pair of tasks in T, i.e., the edges in G are defined as:

$E = \{(i, j) |$ *some data needed to be transferred between tasks* i *and* $j\}$

In some applications such as scheduling, the direction of the edges in G is important. However, in our context, direction makes no difference and we consider G to be an undirected graph. Each edge (i, j) in TIG is associated with a communication cost c_{ij} which is the cost to incur when tasks i and j are assigned to different processors. Since we consider identical communication links between processors, c_{ij} will be constant for all pairs of processors that tasks i and j are assigned to. That is, the communication costs do not depend on the processors that the tasks are assigned to. In addition, assigning tasks i and j to the same processor does not introduce any communication cost. In other words c_{ij} will be 0, if we assign tasks i and j to the same processor.

Let x_{ip} be the execution cost of task i on processor p. The execution costs of the same task on different processors need not to be equal because of the different capabilities of heterogeneous processors in the system. Let X_i be the sum of the execution costs of task i on each processor $p \in P$. In other words;

$$X_i = \sum_{p \in P} x_{ip}$$

The objective of the task assignment problem is to find an assignment function $A : T \rightarrow P$ that minimizes the sum of execution and communication costs. More formally, task assignment problem can be formulated as a minimization problem;

$$Min \left(\sum_{i=1}^{m} \sum_{p=1}^{n} a_{ip} x_{ip} + \sum_{(i,j) \in E} \sum_{p=1}^{n} a_{ip}(1 - a_{jp}) c_{ij} \right) \ subject\ to$$

$$\sum_{p=1}^{n} a_{ip} = 1, \quad i = 1, 2, 3, ..., m$$

$$a_{ip} \in \{0, 1\}, \quad p = 1, 2, 3, ..., n, i = 1, 2, 3, ..., m \ .$$

Here, $a_{ip} = 1$, if task i is assigned to processor p and $a_{ip} = 0$ otherwise. The constraint $\sum_{p=1}^{n} a_{ip} = 1$ enforces the fact that each task i should be assigned to one processor. As it can be realized from the formulation, task assignment problem is very similar to some other well known *NP-complete* problems such as graph partitioning [10] and quadratic assignment [13]. In addition to those similarities, Stone [1] and Magirou [5] find a close correspondence between task assignment problem and multi-way cut problem.

2.2 Previous Work

Numerous studies have been performed to solve the task assignment problem. One of the first is by Stone [1], who used network flow algorithms with a graph theoretical approach to solve the problem for two-processor systems in polynomial time. Stone's algorithm begins with modification of the TIG by adding two nodes labelled as S_1 and S_2 that represent processors P_1 and P_2 respectively. S_1 and S_2 represent unique source and unique sink nodes in the flow network. For each task node, an edge is added from the specific node to each of S_1 and S_2. The weight of an edge between a task and S_1 is equal to the execution cost of that task on the other processor P_2, and the weight of an edge between a task and S_2 is equal to the execution cost of that task on the other processor P_1. In the modified graph (Stone calls it *commodity flow network*), each two-way cut that separates the distinguished nodes S_1 and S_2, represents a solution to the task assignment problem and the weight of the cutset represents the total cost for that assignment. The minimum weight cutset obtained by the application of the maximum network flow algorithm correspond to an optimal solution to the task assignment problem. Stone extended his algorithm to more than two processors using a heuristic. For an n-processor system, Stone's algorithm adds a distinguished node for each processor to TIG. For each task node, an edge from that task node to each distinguished node is also added to the TIG. In this case, the weight of the edge between task node i and distinguished node p is equal to;

$$\frac{X_i}{(n-1)} - x_{ip}.$$

An n-way cut partitions the nodes of *commodity flow network* into n disjoint subsets in such a way that each subset contains exactly one distinguished node. Any n-way cut represents a solution to the task assignment problem. To find an n-way cut, Stone reduced the n-processor problem to several two-processor problems. However, this method is unable to find a complete solution to the problem in most of the cases.

After Stone's work, researchers tried to find exact assignment algorithms for restricted cases. Bokhari [14] presents an $O(mn^2)$ algorithm for TIGs that

have a tree topology, and Towsley [15] presents an $O(mn^3)$ algorithm for *serial-parallel* TIGs by generalizing Bokhari's approach. In another work, Fernandez-Baca [16] presents an $O(mn^{k+1})$ algorithm for the problem where the TIG is a *k-ary tree*. For other cases, the problem is known to be *NP-complete* in general [2].

For general problems, several heuristics have been proposed. Lo's algorithm [3] is one of the well known heuristics. It consists of three phases : *grab, lump* and *greedy* with complexities $O(nm^2|E|log\ m)$, $O(m^2|E|log\ m)$ and $O(nm^2)$ respectively. In the grab phase, Lo [3] uses Stone's [1] approach to find a partial assignment of tasks to processors. The partial assignment found in grab phase is the prefix of all optimal solutions [3]. If the assignment is complete then it is optimal. If there are some tasks remaining unassigned then the lump phase tries to find an optimal assignment by assigning all remaining tasks to one processor. If the lump phase fails to assign all remaining tasks to a processor, then greedy phase is invoked. The greedy phase tries to find the clusters of heavily interacting tasks. To do this, the greedy phase modifies TIG by eliminating the edges whose weight is smaller than the average weight of the edges in TIG. Then, any connected component of the modified TIG is used as a cluster of tasks. Those clusters are then assigned to their *best processors*. Here, and hereafter, we will refer the processor which executes a task or a cluster of tasks with minimum execution cost, as the best processor of that task or task cluster. Lo's algorithm seems to work well in systems that has small number of processors (e.g. n=3,4). However, in the case of medium-to-large number of processors (e.g. $n \geq 5$), the performance of the grab phase degrades drastically. That is, the number of tasks grabbed drastically decreases with increasing n. Furthermore, the performance of the clustering approach used in the greedy phase degrades substantially with increasing n.

Another recent heuristic is presented by Kopidakis et. al. [6]. They transform the minimization of total execution and communication costs into a maximization problem as;

$$Max\left(\sum_{(i,j)\in E} c_{ij}\left(\sum_{p=1}^{n} a_{ip}a_{jp}\right) + \sum_{i=1}^{m}\sum_{p=1}^{n}(1-a_{ip})x_{ip}\right)\ subject\ to$$

$$\sum_{p=1}^{n} a_{ip} = 1, \quad i = 1,2,3,...,m$$
$$a_{ip} \in \{0,1\}, \quad p = 1,2,3,...,n, \quad i = 1,2,3,...,m$$

By doing so, they try to treat processor-to-task edges (pt-edges) and task-to-task edges (tt-edges) in a common framework. In their approach, TIG is augmented to include each processor as a node, and the weight of each pt-edge $(i,p) \in E$, $i \in T, p \in P$ is set to;

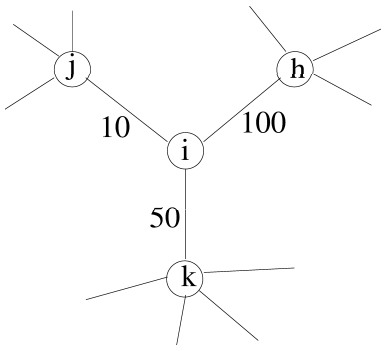
$$c_{ip} = \frac{X_i - x_{ip}}{n-1}$$

to express the term;

$$\sum_{p=1}^{n} x_{ip}(1 - a_{ip})$$

in the maximization problem. Kopidakis et. al. present an $O(m(m+n)^2)$ time task assignment algorithm by using the above formulation and graph model. Their algorithm is a pure clustering algorithm in which contraction of a pt-edge means an assignment and contraction of a tt-edge means clustering of tasks. The scaling between the weights of pt-edges and tt-edges is the main problem in their algorithm. The averaging on the execution times of tasks is not a good solution to this problem. Assume that some of the processors in the system is very slow relative to the others. Then, the weight of the pt-edges between fast processors and tasks will be high relative to tt-edges. Then, averaging will not provide a normalization between pt-edges and tt-edges in heterogeneous systems. So, their approach still suffer from lack of a proper scaling between tt-edges and pt-edges for comparison.

In most of the heuristic models, researchers tried to form task clusters with a minimum cost of intercluster communication. Efe [7] and Bowen et. al. [9] proposed clustering heuristics for the task assignment problem. The main problem in their approaches is that, the difference between the execution costs of the clustered tasks on the same processors is not taken into consideration.



Task	x_{*1}	x_{*2}	x_{*3}
i	2	200	400
j	1	100	200
k	200	2	400
h	400	200	2

Figure 2.1: Clustering alternatives of task i in $G = (T, E)$

2.3 Motivation

Most of the task assignment algorithms using clustering approach tend to minimize the intercluster communication costs first, and then they find a local optimal solution to task assignment problem by assigning those task clusters to their best processors. Since they don't consider the difference between the execution times of tasks in a cluster on the same processors, they also tend to form clusters of tasks that are not similar to each other.

For the sample TIG given in Fig. 2.1, traditional clustering algorithms tend to merge tasks i and h since $(i, h) \in E$ is the edge with maximum weight. Let's investigate the validity of this decision by looking at the different clustering alternatives for task i.

- If we cluster tasks i and j then;
  - 10 units of communication cost is saved,
  - but at least $min_{p \in P} \{x_{ip} + x_{jp}\} = (2 + 1) = 3$ units of execution cost is introduced.
- If we cluster tasks i and k then;
  - 50 units of communication cost is saved,
  - but at least $min_{p \in P} \{x_{ip} + x_{kp}\} = (2 + 200) = 202$ units of execution cost is introduced.
- If we cluster tasks i and h then;

- 100 units of communication cost is saved,
- but at least $min_{p \in P} \{x_{ip} + x_{lp}\} = (200+200) = 400$ units of execution cost is introduced.

So it seems that there is some deficiency in clustering tasks i and h together. This deficiency can not be avoided without taking the execution times of tasks into the consideration.

In addition to this observation, we can say that a task is usually assigned to one of the processors that executes it with low costs relative to the other processors. In other words, a task is rarely assigned to its worst processor in an optimal solution in terms of execution costs. For example, task i is not very likely to be assigned to P_3 in an optimal solution of the sample problem given in Fig. 2.1. So averaging approaches adopted in the schemes presented by Stone [1], Lo [3] and Kopidakis et. al. [6] make some wrong decisions while assigning tasks to the processors. Because, execution times of a task on some processors may be very high relative to the majority of the processors. If we use an averaging scheme, then we have to eliminate those processors from the calculation.

In a clustering approach, the communication cost between a task i and a cluster is equal to the sum of communication costs between task i and all tasks in that cluster. In most of the traditional assignment algorithms that use clustering approach, clusters are formed iteratively (i.e., new clusters are formed one at a time) based on the communication costs between tasks and clusters. This approach corresponds to *agglomerative clustering* in clustering classification. In those approaches, the communication cost between a task and a cluster would automatically create a large volume of communication and iterative clustering algorithms proceed in the next step by contracting an edge neighbour to one just contracted. This problem is known as the *polarization problem* in general. Kopidakis et. al. [6] proposed two solutions for this problem. First solution is to use hierarchical clustering approaches such as matching-based algorithms instead of the iterative algorithms. In hierarchical clustering algorithms, several new clusters may be formed simultaneously. This approach solves the polarization problem, but the experimental results given

in [6] show that it generally leads to decrease in the assignment quality. Other solution presented by Kopidakis et. al. is that they set the communication cost between a task i and a cluster equal to the maximum of the communication costs between task i and the tasks in that cluster instead of sum of them. Choosing the maximum communication cost prevents polarization towards the growing cluster. However, this scheme causes unfairness between clusters and usually, it does not yield good clusters in terms of communication costs.

According to the first observation, if we find a clustering scheme that considers the similarities of tasks while looking at the communication costs, it will give better clusters than the traditional clustering approaches. Second observation says that the assignment algorithm should be optimistic up to a point. That is, while looking at the execution times of a task on different processors, we have to eliminate the worst processors. Finally, third observation displays the need for a clustering scheme which avoids polarization during agglomerative clustering. These observations are the key points for the motivation of the proposed work.

Chapter 3

Single Level Assignment Algorithms

Task assignment algorithms which use clustering approaches usually consist of two phases; *clustering phase* and *assignment phase*. In clustering phase, highly communicating tasks are merged to form new clusters, and then, those clusters are assigned to their best processors in the assignment phase. Many work show that the assignment order of clusters affects the assignment quality of a task assignment algorithm. In this chapter, we present new clustering and assignment approaches for two-phase task assignment algorithms.

3.1 Clustering Phase

In most of the previous clustering approaches to the task assignment problem, such as algorithms proposed by Efe [7] and Bowen et. al. [9], clustering phase and assignment phase are strictly separated from each other. In those algorithms, clustering phase is usually followed by the assignment phase. Clustering phase, as the first phase of those algorithms, has more flexibility than assignment phase, so success of assignment phase heavily depends on the success of clustering phase. Main decisions about the solution are given in the

clustering phase and assignment phase usually completes the solution by using a straightforward heuristic, such as assigning all the clusters to their best processors as in Lo's greedy part [3]. The problem with clustering approach is that, the optimal solution to the reduced problem is not always an optimal solution to the original graph. This is because of the wrong decisions made in the clustering phase of the algorithms. In such algorithms, total intertask communication costs within the clusters are tried to be maximized to minimize the communication costs between clusters. However, this approach does not give good clusters, especially when the processors are heterogeneous. In this section, we will present a new clustering approach that considers the differences between execution costs of tasks on the same processors.

Let's assume that $(i,j) \in E$ for tasks i and j in G. If tasks i and j are assigned to different processors, then their contribution to the total cost with edge (i,j) will be at least;

$$c_{ij} + min_{p \in P}\{x_{ip}\} + min_{p \in P}\{x_{jp}\}$$

where the last two terms are the minimum execution costs of tasks i and j. If tasks i and j are assigned to the same processor, then their contribution to the total cost with edge (i,j) will be at least;

$$min_{p \in P}\{x_{ip} + x_{jp}\}.$$

Let α_{ij} be the profit of clustering tasks i and j together. Generally, tasks i and j are decided to be in the same cluster, if the cost of assigning them to different processors is more than the cost of assigning them to same processor. We can derive an optimistic equation for α_{ij} by subtracting those two costs;

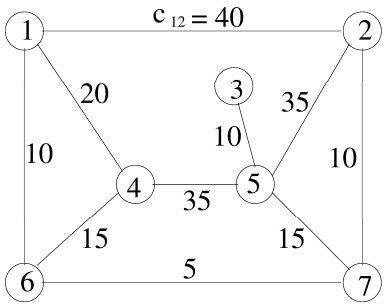
$$\alpha_{ij} = c_{ij} + min_{p \in P}\{x_{ip}\} + min_{p \in P}\{x_{jp}\} - min_{p \in P}\{x_{ip} + x_{jp}\}. \quad (1)$$

In our clustering approach, we consider the clustering of tasks i and j whose clustering profit α_{ij} is maximum. The profit metric in Eq. 1 can be rewritten as;

$$\alpha_{ij} = c_{ij} - d_{ij} \quad (2)$$

where d_{ij} effectively represents the dissimilarity between tasks i and j in terms of their execution characteristics. That is,

$$d_{ij} = min_{p \in P}\{x_{ip} + x_{jp}\} - (min_{p \in P}\{x_{ip}\} + min_{p \in P}\{x_{jp}\}).$$



Tasks	x_{i1}	x_{i2}	x_{i3}
1	65	30	**15**
2	50	**45**	100
3	100	**5**	100
4	85	45	**10**
5	**10**	95	100
6	85	**30**	95
7	35	**25**	90
2,5	**60**	140	200
1,4	150	75	**25**
2,5,7	**95**	165	290

Figure 3.1: TIG and execution times for a sample task assignment problem.

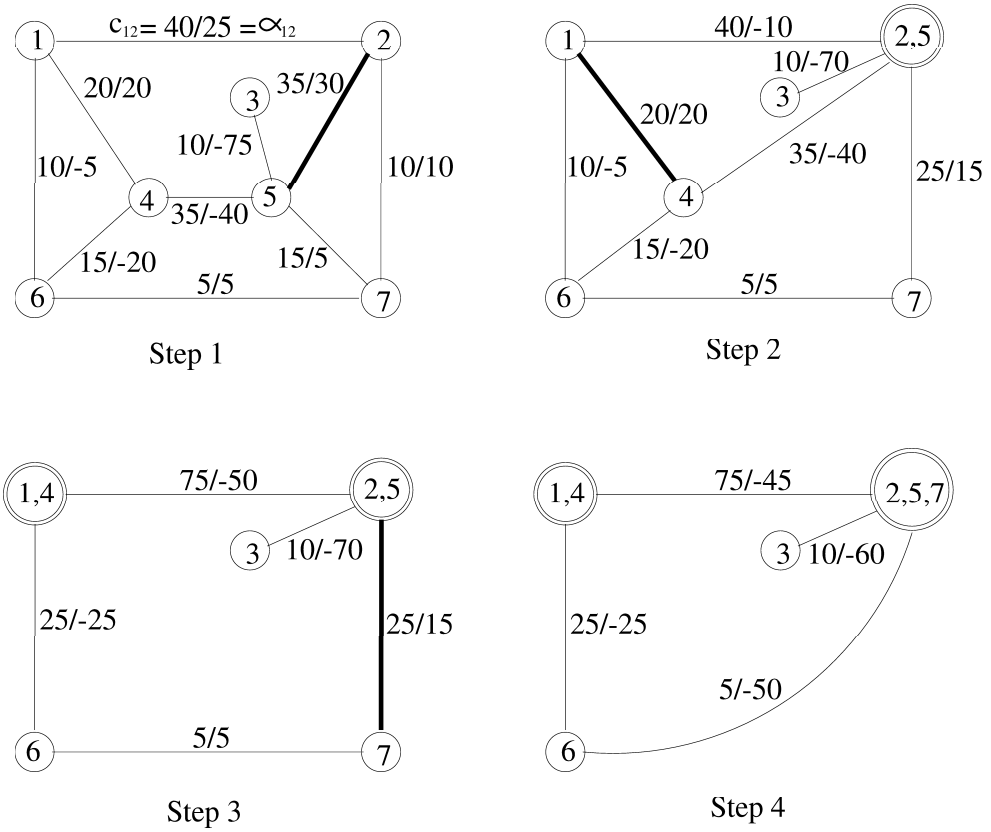


Figure 3.2: Clustering steps of sample TIG given in Fig. 3.1

Note that $d_{ij} \geq 0$ since;

$$min_{p \in P}\{x_{ip} + x_{jp}\} \geq min_{p \in P}\{x_{ip}\} + min_{p \in P}\{x_{jp}\}, \quad \forall i, j, p.$$

In other words, sum of the minimum execution costs of tasks i and j on their best processors is always less than or equal to the minimum of sum of execution costs of tasks i and j on the same processors. Dissimilarity metric achieves its minimum value of $d_{ij} = 0$ when both tasks i and j have the minimum execution cost on the same processor, i.e. when their best processors are the same. As seen in Eq. 2, the profit of a clustering decreases by increasing dissimilarity between the respective pair of tasks. Hence, unlike the traditional clustering approaches, our clustering profit does not only depend on the intertask communication costs but also depends on the similarities of tasks to be clustered. It is an optimistic metric, but it is worth to be optimistic up to a point.

Figure 3.2 presents the steps of our clustering algorithm for the sample task assignment problem defined in Fig. 3.1. The execution costs of the new clusters are also presented in Fig. 3.1. Our clustering algorithm stops when all of the clustering profits are negative. At the end of our clustering algorithm, two new clusters are formed; first one is formed by merging tasks 1 and 4, and second one is formed by merging tasks 2, 5 and 7. By doing so, two decisions are given in the clustering phase; tasks 1 and 4 should be assigned to the same processor, and tasks 2, 5 and 7 should be assigned to the same processor. With this decisions, the original problem is reduced to a smaller problem by contracting the clustered tasks together. We found the optimal solution to the problem in Fig. 3.1 by using the *branch-and-bound* [2] algorithm presented by Magirou and Milis [2]. The cost of optimal solution for the sample problem is 255 units. We observe that a straightforward assignment on the coarsest TIG obtained at the end of Step 4 of Fig. 3.2 achieves the same optimal solution. Here, straightforward assignment corresponds to assigning each task cluster to its best processor. That is, task clusters {1,4}, {2,5,7}, {3} and {6} are assigned to their best processors P_3, P_1, P_2 and P_2 respectively. This result shows that our clustering algorithm produced perfect clusters for the sample problem. It means that decisions which are given in the clustering phase are completely correct for the sample problem. Lo's algorithm [3] give a solution whose cost is 275 units while the algorithm proposed by Kopidakis et. al. [6]

give a solution whose cost is 285 units for the same problem.

We have presented a profit metric for clustering two tasks, but we can extend our metric to clusters of k-tasks ($2 \leq k \leq m$) by preserving the general principles of our approach. Let S be the set of tasks to be considered for clustering, and α_S be the clustering profit of tasks in S. Then,

$$\alpha_S = \frac{1}{2}\sum_{i\in S}\sum_{j\in S} c_{ij} + \sum_{i\in S} min_{p\in P}\{x_{ip}\} - min_{p\in P}\left\{\sum_{i\in S} x_{ip}\right\}.$$

We apply pure agglomerative clustering algorithm in our clustering approach. TIG is initially considered to have n clusters of exactly one task each. At each pass, the algorithm merges a set of clusters into a new cluster. Let S be the set of clusters that are decided to be merged in our clustering algorithm into a new cluster labelled as k. Then the execution times of the new cluster on each processor p is;

$$x_{kp} = \sum_{i\in S_k} x_{ip},\ \forall p = 1, 2, ..., n.$$

All external edges of the tasks in S are merged to form the adjacency list of new cluster k while deleting the internal edges. Then, algorithm continues in the same way as far as the largest clustering profit remains above zero.

Our clustering scheme is iterative, but it inherently solves the polarization problem. Because, our clustering scheme does not only consider the communication costs of tasks but it also considers the difference between the execution times of the tasks being clustered. As in most of the clustering algorithms, the communication cost between a task and a cluster is large relative to that of a pair of single tasks in our clustering scheme. But the difference between the execution times of a task and a cluster is also large relative to that of a pair of single tasks. So our clustering gain metric does not degenerate when the clusters get bigger.

3.2 Assignment Phase

Clustering phase does not give any solution to the task assignment problem, so we must somehow assign the clusters of tasks to processors after the clustering

phase of the algorithm. Numerous research on iterative assignment algorithms have shown that quality of an assignment heavily depends on the order in which the tasks are assigned. There are a lot of assignment heuristics that try to find a reasonable order in the assignment of tasks. One of them is by Williams [8]. Williams sorted tasks by their sum of communication costs and than assigned tasks in that order to their best processors. This algorithm is a straightforward but efficient algorithm. In this section, we present two new assignment heuristics that are used to determine the assignment order; *assignment according to clustering loss* and *assignment according to grab affinity*. In both of the heuristics, each cluster selected for assignment is assigned to its best processor.

3.2.1 Assignment According to Clustering Loss

In the previous section, we presented a profit metric α_S for clustering a set (S) of tasks into a new cluster. If α_S is positive, clustering the tasks in S may be a good decision. Let assume that all clustering profits of task i with other tasks is negative. Then forming a cluster including task i is meaningless, so it is better to assign task i to its best processor. But if there are more than one tasks that have negative clustering profits for all their clustering alternatives, then the order in which clusters are assigned may affect the solution quality of the algorithm. Our experiments showed that assigning the task with most negative clustering profits first gives better solutions to the task allocation problems. This is reasonable, because the task with most negative clustering profits is the most independent task in general. So, in case of faulty assignment, other tasks will not be affected very much.

3.2.2 Assignment According to Grab Affinity

The word grab is first used by Lo [3] to identify the first phase of her algorithm. In grab phase, Lo's algorithm tries to find a prefix to optimal solution by using maximum flow algorithm on *commodity flow network*. In each iteration of the grab phase, a number of tasks may be grabbed by a processor, and these tasks

are then assigned to a processor. Assume that only one task (task i) is grabbed by a processor p in a step of grab phase. Then, the following inequality must hold;

$$\frac{X_i}{n-1} - x_{ip} \geq \sum_{(i,j)\in E} c_{ij} + x_{ip}$$

For any task i, let;

$$r_i = \frac{X_i}{n-1} - 2x_{ip} - \sum_{(i,j)\in E} c_{ij}$$

where p is the best processor of task i. If r_i is greater than 0, then task i is assigned to its best processor in any optimal assignment. For $r_i \leq 0$, a greater r_i means that task i is more likely to be assigned to its best processor in an optimal solution. Due to this observation, selecting the task i with greatest r_i for assigning first, is more likely to give better solutions to task assignment problem. We use this criteria to determine the cluster of tasks to be assigned first in the assignment phase of our algorithms.

After assigning task i to a processor, if we assign another task j adjacent to task i to the same processor, then there will be a communication cost which is saved. So, after assigning a task to a processor, we must adjust the execution times of the tasks which are adjacent to that task in TIG. In this case, Lo [3] proposed a method for adjusting the execution costs of tasks. We also used this method in our algorithms. Assume that task i is assigned to processor p and task j is an unassigned task which is adjacent to task i in TIG. Then, new execution cost of task j on processor $q \in P$ such that $(q \neq p)$ is;

$$x^*_{jq} = x_{jq} + c_{ij}, \tag{3}$$

and execution cost of task j on processor p will not change.

3.3 AC2 Task Assignment Algorithm

In one phase algorithms such as the one presented by Kopidakis et.al. [6] scaling and polarization problems generally leads to bad solutions. In this section, we present a two-phase assignment algorithm (AC2) which has a loose asymptotic upper bound of $O(|E|^2 n + |E| m \log m)$ in worst case. AC2 consists of two phase; *clustering phase* and *assignment phase*.

```
AC2 (G, x)
  Q ← ∅
  for each task i ∈ T do
      compute clustering profit α_ik for each task k ∈ Adj[i] according to Eq. 1
      choose the best mate j ∈ Adj[i] of task i with α_ij = max_{k∈Adj[i]} {α_ik}
      INSERT (Q, i, α_ij)
      mate[i] ← j
  while Q ≠ ∅ do
      i ← MAX (Q)
      if key[i] > 0 then
          i ← EXTRACT-MAX (Q)
          CLUSTER (G, Q, x, i, mate[i])
      else
          select the task i with maximum assignment affinity
          ASSIGN (G, Q, x, i)
```

Figure 3.3: AC2 task assignment algorithm

In the clustering phase, our algorithm uses pure agglomerative clustering approach to form the clusters of tasks by using the clustering profit described above. In the assignment phase, one of the two assignment criteria can be used to determine the task to be assigned. In our implementation, we use assignment with grab affinity as the assignment criterion. The task which is selected for assignment is assigned to its best processor according to the modified execution times of tasks. The pseudo codes for clustering phase and assignment phase of our algorithm are given in Fig. 3.4 and Fig. 3.5 respectively.

In the AC2 assignment algorithm given in Fig. 3.3, the property $Adj[i]$ for task i represents the set of all tasks which are adjacent to task i. The property $mate[i]$ for task i contains the best clustering alternative of task i among all adjacent unassigned tasks and property $key[i]$ contains the queuing key for task i which is equal to the clustering profit of tasks i and $mate[i]$. The algorithm continuously forms supertasks by merging pairs of tasks whose clustering profits are positive. When the clustering profits of all task pairs are negative, then a task which is selected according to one of our assignment criteria, is assigned

CLUSTER (G, Q, x, i, j)
  DELETE(Q, j)
  merge tasks i and j into a new supertask k
  construct $Adj[k]$ by performing weighted union of $Adj[i]$ and $Adj[j]$
  update $Adj[h]$ accordingly for each task $h \in Adj[k]$
  for each processor $p \in P$ **do**
    $x_{kp} \leftarrow x_{ip} + x_{jp}$
  for each $h \in Adj[k]$ **do**
    compute clustering profit $\alpha_{hk} = \alpha_{kh}$
    if $key[h] \leq \alpha_{hk}$ **then**
      INCREASE-KEY (Q, h, α_{hk}) with $mate[h] = k$
    elseif $mate[h] = i$ or $mate[h] = j$ **then**
      recompute the best mate $\ell \in Adj[h]$ of task h
      DECREASE-KEY $(Q, h, \alpha_{h\ell})$
  choose the best mate $\ell \in Adj[k]$ for task k
  INSERT $(Q, k, \alpha_{k\ell})$ with $mate[k] = \ell$

Figure 3.4: AC2 clustering algorithm

ASSIGN (G, Q, x, i)
  DELETE(Q, i)
  assign task i to its best processor
  for each task $j \in Adj[i]$ **do**
    $Adj[j] \leftarrow Adj[j] - \{i\}$
    for each processor $q \in P - \{p\}$ **do**
      $x_{jq} \leftarrow x_{jq} + c_{ij}$

Figure 3.5: Assignment algorithm for task i to processor p

to its best processor. Assignment of a supertask to a processor effectively means assignment of all its constituent tasks to that processor. Note that after the assignment of a task, the clustering profits of some unassigned task pairs may become positive. If so, the algorithm forms intermittent clusters. Our algorithm terminates when all tasks are assigned to processors. In AC2, we use a priority queue (Max-heap) to get the pair of task with maximum clustering profit.

After each clustering and assignment phase of AC2 the key values for unassigned tasks are changed. So, the key values of the tasks in the priority queue must be updated appropriately after each clustering and assignment of tasks. The update operations on the priority queue is achieved by using *increase-key* and *decrease-key* operations. When a task pair (i, j) is clustered into a supertask k then, we have to update clustering profits of adjacent tasks on the priority queue. If the clustering profit of an adjacent task h with new task k is greater than the old key value of task h, then task k will be the best mate for task h with a greater key value which is equal to α_{hk}. Otherwise, the algorithm recomputes the best clustering profit of task h, only if the old best mate of task h is either task i or task j. In this case, the key value of task h have to be decreased. In all other cases, the key value and best mate of task h will remain unchanged. When a task i is assigned to its best processor, the execution times of all unassigned tasks adjacent to task i are updated according to Eq. 3. Although, TIG seems to be updated in the algorithm for the sake of simplicity of presentation, the topology of TIG is never changed in our clustering algorithm. In addition, pt-edges are not explicitly considered in our implementation for run-time efficiency, instead they are considered implicitly.

In this work, we implemented another assignment algorithm (AC3) in which at most three tasks are clustered instead of two. AC3 is able to find the heavily communicating triple tasks in the TIG in one iteration of the clustering algorithm. With this characteristic, it has a more powerful clustering scheme than AC2. The implementation of AC3 is very similar to AC2, but it needs substantially more computation time than AC2.

Chapter 4

Multilevel Task Assignment Algorithms

Multilevel graph partitioning methods have been proposed leading to successful graph partitioning tools such as Chaco [17] and MeTiS [10]. These multilevel heuristics consist of three phases, namely *coarsening, initial partitioning* and *uncoarsening*. In the first phase, multilevel clustering is successively applied starting from the original graph by adopting various clustering heuristics until the number of tasks in the coarsened graph reduces below a predetermined threshold value. In the second phase, the coarsest graph is partitioned using various heuristics. In the third phase, the partition found in the second phase is successively projected back towards the original graph by refining the projected partitions on intermediate level graphs using several heuristics. In this chapter, we try to adopt this multilevel scheme to task assignment problem. Our multilevel algorithms also have these three phases. In the first phase, TIG will be coarsened by using several clustering heuristics. In the second phase, an initial assignment will be found on the reduced task assignment problem. In the third phase, the assignment found in the second phase is successively projected back to the original problem by refining the assignment in each intermediate level. Since the original problem has more degrees of freedom, such refinements decrease the cost of assignments at each level.

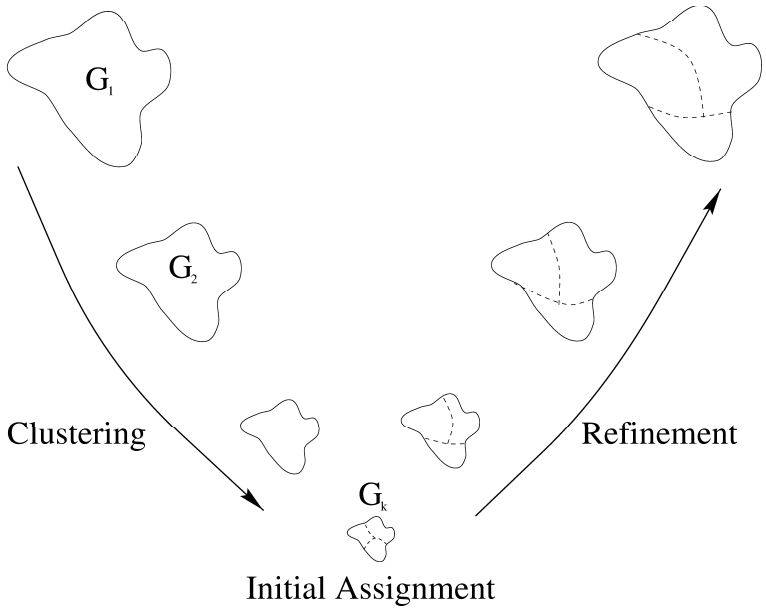


Figure 4.1: Multi level assignment for a system of three processors

4.1 Clustering Phase

In this phase, the given TIG $G = G_0 = (T_0, E_0)$ is coarsened into a sequence of smaller TIGs $G_1 = (T_1, E_1)$, $G_2 = (T_2, E_2), ..., G_k = (T_k, E_k)$ satisfying $|T_0| > |T_1| > |T_2| > ... > |T_k|$. This coarsening is achieved by coalescing disjoint subsets of tasks of TIG G_i into *supertasks* such that each supertask in G_i forms a single task of G_{i+1}. The execution time of each task of G_{i+1} on a processor becomes equal to the sum of its constituent tasks of the corresponding supertask in G_i. The edge set of each supertask is set equal to the weighted union of the edge sets of its constituent tasks. Coarsening phase terminates when the number of tasks in the coarsened TIG reduces below the number of processors (n) or reduction on the number of tasks between successive levels is below 90 percent (i.e., $0.90|T_k| < |T_{k+1}|$). In the clustering phase, we apply our clustering profit metric presented in Section 2.1. We present five heuristics to reduce TIG; *matching-based clustering (MC), randomized semi-agglomerative clustering (RSAC2), semi-agglomerative clustering (SAC2), agglomerative clustering (AC2)* and *multi-multi level assignment(MLA).*

4.1.1 Matching-Based Clustering

Matching-based clustering works as follows. For each edge $(i,j) \in E_\ell$ in the TIG G_ℓ the clustering profit α_{ij} for tasks i and j is calculated. Then, each pair of adjacent tasks i and j are visited in the order of descending clustering profit α_{ij}. If both of the adjacent tasks are not matched yet, then those two adjacent tasks are merged into a cluster. By doing so, our clustering algorithm tries to form clusters of tasks that provide maximum clustering profits over all tasks. If the clustering profit of tasks i and j is less than 0, then those two tasks are not matched and the matching algorithm terminates at this point. At the end, unmatched tasks remain as singleton clusters. This matching scheme does not give the maximum weighted matching in terms of edge clustering profits, because it is very costly to find maximum weighted matching on a graph. Our scheme only tries to find a matching close to the maximum matching by using a heuristic. Matching-based clustering allows the clustering of only pairs of tasks in a level. In order to enable the clustering of more than two tasks at each level, we have provided agglomerative clustering approaches.

4.1.2 Randomized Semi-Agglomerative Clustering

In this scheme, each task i is assumed to constitute a singleton cluster, $C_i = \{i\}$ at the beginning of each coarsening level. Here, C_i also denotes the cluster containing task i during the coarse of clustering. Then, clusters are visited in a random order. If a task i has already been clustered (i.e. $|C_i| > 1$), then it is not considered for being the source of a new clustering. However, an unclustered task can choose to join with a supertask cluster as well as a singleton cluster. That is, all adjacent clusters of an unclustered task are considered for selection. A task i is tried to be included in an adjacent cluster C_j which has the maximum clustering profit with task i among all adjacent clusters of task i. Selecting the cluster C_j adjacent to task i corresponds to including task i in the cluster C_j to grow a new multitask cluster $C_i = C_j = C_j \cup \{i\}$. For this case, if the clustering gains of a task i are all negative, then task i remains unclustered. That is, task i will be a singleton cluster for the next level. The clustering quality of this scheme is not predictable, because it highly depends on the order in which the

clustered tasks are visited. That is, at each run, this clustering scheme gives different clusters of tasks. So, it is not used in an assignment algorithm, but instead, we used this clustering scheme in a randomized assignment algorithm which we run many times to find solutions to a task assignment problem whose best result is expected to be quite close to an optimal solution.

4.1.3 Semi-Agglomerative Clustering

This version of clustering approach is very similar to the randomized semi-agglomerative clustering approach. The only difference is that, a single task to be clustered is not selected randomly, instead, a single task with the highest clustering profit among others is selected as the source of the clustering. The solution quality obtained by the semi-agglomerative clustering approach is more predictable. In fact, it gives relatively better solution quality than the average solution quality of the randomized version. But it is also very likely to be stuck on a local optimal solution whose refinement is not easy.

4.1.4 Agglomerative Clustering

In semi-agglomerative clustering approaches, single tasks are enforced to be included in a cluster. In those approaches, some very good clustering alternatives that can be obtained by merging two multitone clusters are not considered. In the agglomerative clustering, two multitone clusters can be merged together in a single level. By doing so, we try to eliminate the deficiencies in semi-agglomerative clustering approaches. This clustering approach is very similar to the AC2 clustering algorithm presented in Section 3.1. But, in this case, it is adopted to the multilevel scheme. We do not use a randomization scheme for agglomerative clustering approach.

4.1.5 Multi-Multi Level Assignment

In all of the above algorithms, the original TIG is reduced by clustering tasks into a single task. Another approach to reduce the original problem could be to assign some of tasks in each level of the algorithm. In this section, we present a multilevel algorithm which reduces the original problem by successively assigning some of the tasks in each level. Let's assume that we have a multilevel assignment algorithm which uses the randomized semi-agglomerative clustering approach. It is obvious that, at each run, this algorithm will give different assignments for the same task assignment problem. If we run this algorithm for sufficiently large times, the cost of the best assignment obtained in those runs can be expected to be very close to the cost of optimal solution to that task assignment problem. In this algorithm, 5 different assignments are found for a given task assignment problem by using a randomized multilevel assignment algorithm. From those 5 assignments, we choose the best 4 assignments to eliminate the negative effects of significantly bad assignments. If task i is assigned to the same processor p in all of the 4 assignments, then it is assigned to processor p at the current level. Then, task i and all edges of task i are deleted from the TIG for the next levels. In next levels, task i will not be considered as a task in any phase. But in the refinement phase, task i will be free to be assigned to any other processor at higher levels. After this assignment, we have to adjust the execution costs of the adjacent tasks to reflect the assignment. For any edge $(i, j) \in E$, we add c_{ij} to all execution times of task j on all processors except processor p. This approach gives very good assignments for any task assignment problem, but it has a relatively high running time. This tradeoff can be lowered by using less than 5 assignments at a time, but in that case, it is likely to get worse solutions.

4.2 Initial Assignment Phase

The aim of this phase is to find an assignment for the task assignment problem in the coarsest level. We can find the initial assignment by using our single level task assignment algorithms as well as Lo's [3] algorithm. It is obvious that

good initial assignments usually lead to better solutions to the original task assignment problem. So, in our multilevel algorithms, we use the two-phase task assignment algorithm AC2 described in Section 3.3 to find the initial assignment for a task assignment problem.

4.3 Uncoarsening Phase

At each level ℓ, assignment A_ℓ found on the set T_ℓ is projected back to an assignment $A_{\ell-1}$ on the set $T_{\ell-1}$. The constituent tasks of each supertask in $G_{\ell-1}$ is assigned to the processor that the respective supertask is assigned to in G_ℓ. Obviously, this new assignment $A_{\ell-1}$ has the same cost with the previous assignment A_ℓ. As the next step, we refine this assignment by using a refinement algorithm starting from the initial assignment $A_{\ell-1}$. Note that, even if the assignment A_ℓ is at a local minima (i.e. reassignment of any single task does not decrease the assignment cost), the projected assignment $A_{\ell-1}$ may not be at a local minima. Since $G_{\ell-1}$ is finer, it has more degrees of freedom that can be used to further improve the assignment $A_{\ell-1}$ and thus decrease the assignment cost. Hence, it may still be possible to improve the projected assignment $A_{\ell-1}$ by local refinement heuristics.

Kernighan and Lin (KL) [11] proposed a refinement heuristic which is applied in refinement phase of the graph partitioning tools because of their short run-times and good quality results. KL algorithm, starting from an initial partition, performs a number of passes until it finds a locally minimum partition. Each pass consists of a sequence of vertex swaps. Fiduccia and Mattheyses (FM) [12] introduced a faster implementation of KL algorithm by proposing vertex move concept instead of vertex swap. This modification as well as proper data structures, e.g., bucket lists, reduced the time complexity of a single pass of KL algorithm to linear in the size of the graph. In coarsening phase of our assignment algorithm, we use FM approach with some modifications to refine the assignments in intermediate levels. In this version of FM, we propose task reassignment concept instead of vertex move in graph/hypergraph partitioning.

Let task i be assigned to processor p in an assignment. The *reassignment*

gain of task i from processor p to another processor q is the decrease in the cost of assignment, if task i is assigned to the processor q instead of processor p. In other words, reassignment gain for task i from processor p to processor q is equal to :

$$g_{i,p\rightarrow q} = \left(x_{ip} + \sum_{j \in Adj[i], a[j]=q} c_{ij} \right) - \left(x_{iq} + \sum_{j \in Adj[i], a[j]=p} c_{ij} \right),$$

where $a[j]$ denotes the current processor assignment for task j. Our FM algorithm begins with calculating the maximum reassignment gain for each task i in current TIG. Those tasks are inserted into a priority queue according to their maximum reassignment gains. Initially all tasks are unlocked, i.e., they are free to be reassigned to the other processors. The algorithm selects an unlocked task with the largest reassignment gain from the priority queue and assigns it to the processor which gives the maximum reassignment gain. After the reassignment of a task i, the algorithm locks task i and recalculates the reassignment gains of all tasks adjacent to task i. Note that, our algorithm does not allow the reassignment of the locked tasks in a pass since this may result in trashing. A single pass of the algorithm ends when all of the tasks are locked, i.e, (all tasks have been reassigned). At the end of a FM pass, we have a sequence of tentative task reassignments and their respective gains. Then from this sequence, we construct the maximum prefix subsequence of reassignments with the maximum sum which incurs the maximum decrease in the cost of the assignment. The permanent realization of the reassignments in this maximum prefix subsequence is efficiently achieved by rolling back the remaining moves at the end of the overall sequence. Now, this assignment becomes the initial assignment for the next pass of the algorithm. The roll-back scheme in FM provides hill-climbing ability in refinement. So, FM does not stuck to a trivial local optimal assignment. The overall refinement process in a level terminates if the maximum prefix sum of a pass is not positive. In the case of multilevel assignment algorithms, FM refinement becomes very powerful, because the initial assignment available at each successive uncoarsening level is already a good assignment.

Chapter 5

Experimental Results

5.1 Data Sets

We have evaluated the performance of the proposed algorithms for randomly generated problem instances. We can classify the set of problem instances which are used in this work into two groups according to topologies of their respective TIGs. In the first group, we have generated problem instances whose TIGs are trees and in the second group, we have generated problem instances whose TIGs are general graphs. Optimal assignments can be effectively obtained by using Bokhari's task assignment algorithm [14] for the problem instances with tree TIGs and so, the performance of the proposed algorithms can be determined accurately. On the other hand, it is infeasible to compute the optimal assignments for the problem instances with general TIGs. In this case, the assignments of the proposed algorithms are compared to the best known assignment for any specific problem instance. The best known assignments for the problem instances with general TIGs are determined by running our randomized multilevel assignment algorithm described in Chapter 4 for 100000 times on each problem instance and choosing the best assignment among the results of these 100000 runs.

In generation of problem instances with general TIGs, the topologies of TIGs are selected from the set of DWT symmetric matrices in Harwell-Boing matrix

			Edge Degrees				
Topology	m	$\|E\|$	min	max	avg	σ	cov
DWT59	59	104	1	5	3.53	7.40	2.10
DWT66	66	127	1	5	3.85	5.15	1.34
DWT72	72	75	1	4	2.08	5.24	2.52
DWT87	87	227	1	12	5.22	21.61	4.14
DWT162	162	510	1	8	6.30	22.09	3.51
DWT198	198	597	1	11	6.03	30.10	4.99
DWT209	209	767	3	16	7.34	32.57	4.44
DWT221	221	704	3	11	6.37	23.87	3.75
DWT234	234	300	1	9	2.56	22.48	8.77
DWT245	245	608	1	12	4.96	40.63	8.19
DWT307	307	1108	5	8	7.22	19.14	2.65
DWT310	310	1069	3	10	6.90	25.94	3.76
DWT361	361	1296	3	8	7.18	25.52	3.55
DWT419	419	1572	5	12	7.50	35.68	4.75
DWT492	492	1332	2	10	5.41	43.17	7.97

Table 5.1: Properties of DWT symmetric matrices

collection. We have used 15 different topologies in generation of the problem instances. Properties of the matrices which are used as general graph topologies are summarized in the Table 5.1. During the test data generation, the execution costs are randomly selected integers which follow a uniform distribution within interval $[1, I_e]$. Similarly, the communication costs are randomly selected integers which follow a uniform distribution within interval $[1, I_c]$. In all of our experiments, we have fixed I_c to 100 for each edge in TIG. Since the generation of realistic problems is critical for the validation of the proposed algorithms, we have tried to avoid generation of trivial problems. To do this, we have used different I_e's for each task i according to the following equation.

$$I_e = \frac{2}{r_{com}} \sum_{j \in Adj[i]} c_{ij}$$

In this way, we have tried to keep execution costs and communication costs comparable with each other. In this equation, r_{com} is the communication ratio which is varied in order to estimate the impact of the relative size of execution and communication costs. In our experiments, we have generated problem instances for 3 different values of r_{com} which are 0.7, 1.0 and 1.3. Another parameter in generation of the problem instances is the number of processor

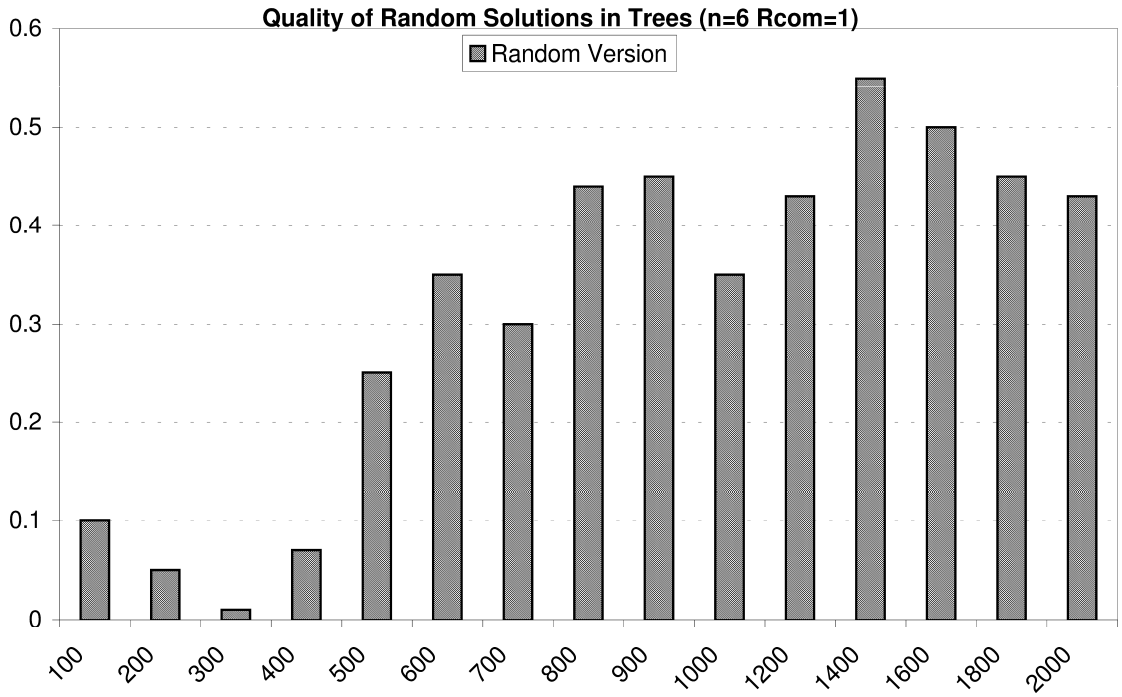


Figure 5.1: Percent relative distance of the best solutions provided by 100000 runs of the randomized multilevel assignment algorithm on trees

(n). In our experiments, we have generated problem instances with 3, 6, 9, 12, 15 and 18 processors. For each different combination of parameters (TIG, n, r_{com}), 20 different random problem instances are generated and solved by proposed algorithms

In generation of problem instances with tree TIGs, we have created random tree topologies as follows. First, we have created completely connected graphs with m nodes (m is the number of tasks) whose edges are randomly weighted. Then, we have found the minimum spanning trees of those graphs. These minimum spanning trees are used as random tree topologies in our experiments. For a fixed m, we have generated 20 different tree topologies to avoid the effects of tree diameters on the assignment qualities. As in the case of general graphs, we have generated 20 random problem instances for each different tree topology, i.e. 400 problem instances are generated for each fixed m. For this case, n and r_{com} are chosen from the same sets described above and execution costs and communication costs are assigned in the same manner.

As a measure of the solution quality, the relative distance from the best known solution to a problem instance (best known solutions are optimal solutions for trees) is calculated for each of the implemented algorithms. The percent relative distance for an algorithm A is equal to the;

$$100 \times \frac{S_A - S_B}{S_B},$$

where S_B is the quality of the best known solution and S_A is the quality of the solution provided by algorithm A. By using optimal solutions provided by Bokhari's algorithm on trees, we have tested the solution qualities of the best known solutions provided by our randomized multilevel assignment algorithm. As it can be seen from Fig. 5.1, it provides solutions which are very close to their optimal solutions in quality for trees. So, it shows that our quality metric is reliable for general graphs.

5.2 Implementation of the Algorithms

We have implemented two single level task assignment algorithms; AC2 and AC3 according to clustering and assignment schemes described in Chapter 3. The solutions provided by AC2 and AC3 are refined by using a two-level FM scheme in order to see the effects of refinement on the single level algorithms. In the first level of FM, solutions are refined by reassigning only the clusters of tasks which are formed by the clustering schemes of AC2 and AC3. In second level of FM, refined solutions are projected back into the original problem and the projected solutions are refined by reassigning the tasks of the original problem.

In addition to those single level algorithms, we have implemented 4 multilevel task assignment algorithms which use the clustering schemes; matching-based clustering (MC), semi-agglomerative clustering (SAC2), agglomerative clustering (AC2) and multi-multi level assignment (MLA) described in Chapter 4. For the sake of ease of presentation, we call multilevel algorithms with the name of their clustering schemes. For example, multilevel task assignment algorithm which uses MC is called as MC-ML (ML stands for multi level).

```
KLZ (G, x)
  modify the TIG G = (E, T) into G' = (E', T ∪ P)
  sort the edges of G' according to their weights in descending order
  for each edge e ∈ E' in decreasing order of their weights do
    if e = (p, i) is a pt-edge where p ∈ P and i ∈ E then
      if task i is not assigned then
        assign task i and all tasks clustered with task i to processor p
    else if e = (i, j) is a tt-edge where i, j ∈ T then
      if task i is not assigned then
        if task j is not assigned then
          merge two clusters represented by task i and j together
        else
          let p be the processor to which task j is assigned
          assign task i and all tasks clustered with task i to processor p
      else if task j is not assigned then
        let p be the processor to which task i is assigned
        assign task j and all tasks clustered with task j to processor p
```

Figure 5.2: Asymptotically faster implementation proposed for the KLZ assignment algorithm

So, our 4 multilevel algorithms are MC-ML, SAC2-ML, AC2-ML and MLA-ML. We have used assignment with grab affinity to find an initial solution in the coarsest level of the multi level assignment algorithms. The refinement phase of all multilevel assignment algorithms are implemented as described in Chapter 4.

The algorithms proposed by Lo [3] and Kopidakis et. al. [6] are also implemented in this work for relative performance evaluation. The former and the latter algorithms are referred to here as VML and KLZ respectively. The implementation proposed by Kopidakis et. al. [6] for their MaxEdge algorithm leads to $O(m(m+n)^2)$ time computational complexity. In this work, we propose an asymptotically faster implementation for the KLZ algorithm. The proposed scheme displayed in Fig. 5.2 runs in $O((|E|+mn)\log(|E|+mn))$. The solutions provided by those algorithms are also refined. In the refinement

of the solutions provided by VML, the grabbed tasks are locked to prevent them from being reassigned to the other processors, since they are already assigned to their optimal processor.

5.3 Effects of the Assignment Criteria

In Chapter 3, we have presented two assignment heuristics to determine the cluster of tasks to be assigned first. We have tried both of the assignment heuristics along with the one proposed by Williams [8] in our AC2 algorithm for various task assignment problems to find their effects on the solution qualities. As seen in Fig. 5.3, both of the assignment heuristics give approximately same solution qualities on different problem instances. However, assignment according to grab affinity gives slightly better solutions on average. Lo's work [3] shows that when the number of processors in a task assignment problem is small, a task will have higher chance of being grabbed. So, it is likely that assignments according to grab affinity give better solution qualities for the systems that have small number of processors. However, if we use assignment according to grab affinity in our single level task assignment algorithms, we need a second priority queue to keep the grab affinities of unassigned tasks. Since our single level assignment algorithms (AC2 and AC3) calculates maximum clustering profits of unassigned tasks, we do not need a second priority queue in our algorithms which use assignment according to clustering loss. So there is a trade off. For the sake of uniformity, we use assignment according to grab affinity in all single level task assignment algorithms which we are tested.

5.4 Experiments with Tree TIGs

Figures 5.4, 5.5 and 5.6 illustrate the percent qualities of the proposed algorithms for 3, 9 and 15 processors systems with tree TIGs respectively. As seen in Fig. 5.4, VML performs substantially better than KLZ on 3-processor systems. However, Figs. 5.5 and 5.6 show that KLZ substantially outperforms VML for 9-processor and 15-processor systems. This situation is due to the

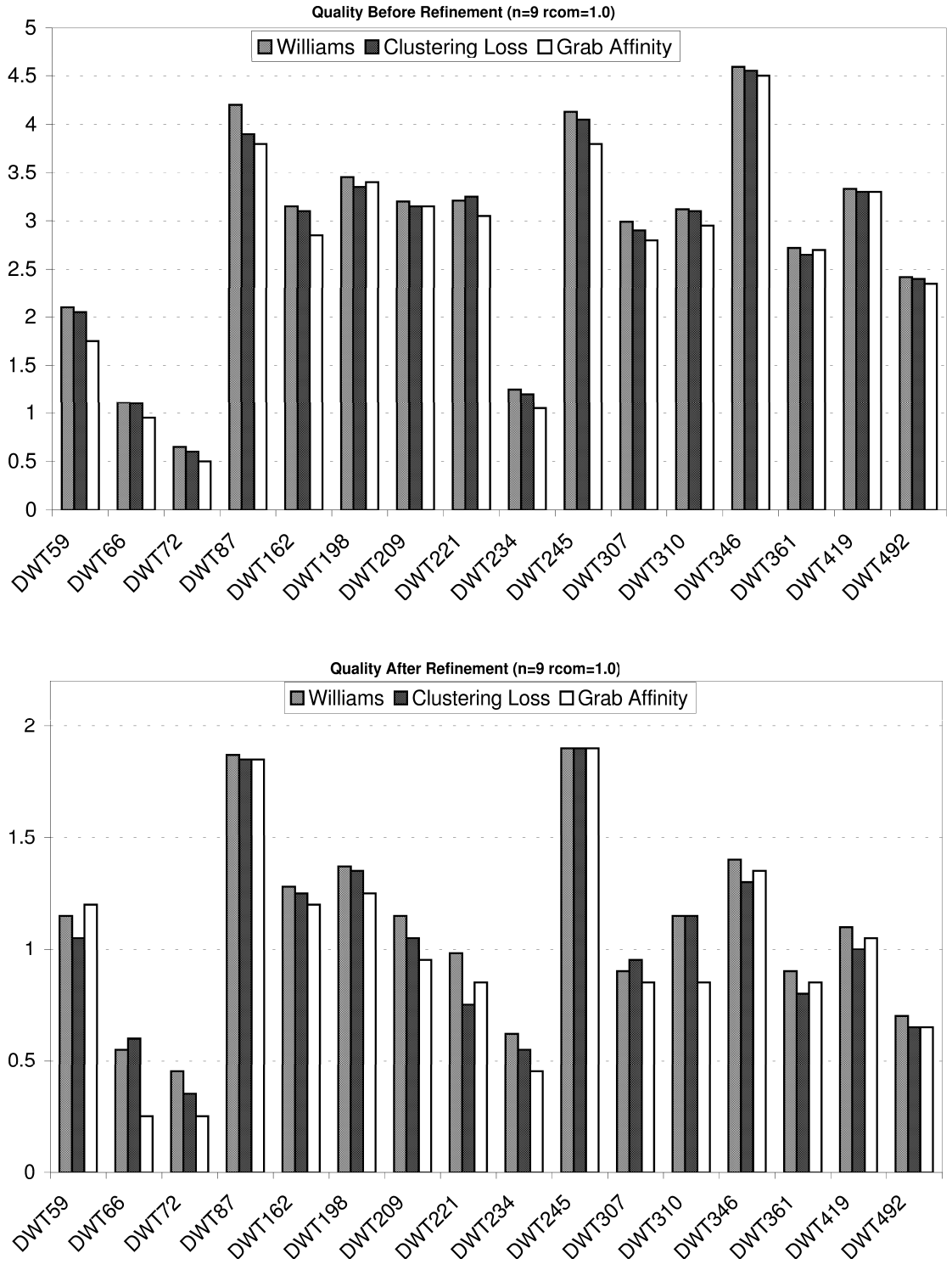


Figure 5.3: Percent relative performance of assignment heuristics applied to AC2 algorithm

fact that the grab phase of VML works only for small number of processors especially for 2 and 3 processors. But, as seen in Figs. 5.4-5.6, there is no clear winner among the refined versions of VML and KLZ. Figures 5.4-5.6 show that qualities of the solutions provided by the proposed algorithms are obviously and constantly superior in any case. As seen in Figs. 5.4-5.6, the assignment qualities of all proposed algorithms based on clustering approaches decrease with increasing number of tasks. However, the assignment quality of MLA-ML does not affected from the number of tasks. This finding can be most probably due to the fact that assignment gets importance over clustering in task assignment problems whose TIGs are sparse.

The relative solution qualities of experimented algorithms on tree TIGs are summarized in Tables 5.2 and 5.3. As seen in Table 5.2, our AC2 and AC3 algorithms produce substantially better solutions than other assignment algorithms. In Table 5.2, it is also observed that AC2 produces slightly better solutions than AC3. This finding can be attributed to the fact that forming clusters of 3 tasks in trees is not a good approach because, there is no 3-cliques in trees. So AC2 has an advantage in sparse graphs, although AC3 is more powerful in general. Another important observation in Table 5.2 is that the performances of all experimented algorithms get worse with increasing n and r_{com}. This situation can be most probably due to the fact that it becomes harder to find optimal solutions for the task assignment problems with large n and r_{com}.

If we look at the solution qualities of VML and KLZ in Table 5.2, we can see that VML gives better assignments than KLZ only for the 3-processor systems. The performance of VML drastically decreases with increasing number of processors. This is expected, because the performance of VML mainly depends on the success of its grab phase. The grab phase works only for small number of processors. For the task assignment problems with large number of processors, the assignments of VML is generally provided by the greedy phase. The results in Table 5.2 shows that straightforward clustering scheme in greedy phase of VML usually leads to bad assignments relative to other clustering schemes.

The relative assignment qualities of our multilevel algorithms and the refined versions of single level algorithms are summarized in Table 5.3. As seen

in Table 5.3, SAC2-ML and AC2-ML give substantially better solutions than others, and MLA-ML gives worst solution qualities among the multilevel algorithms. This is also expected, because the success of MLA-ML highly depends on the refinement phase of its random assignments. Numerous work has shown that the performance of FM schemes deteriorates for too sparse graphs. As seen in Table 5.4, the improvements which are provided by FM on the solutions of our algorithms are less than 1% for all cases. Although it works well on the solutions of VML and KLZ, the solution qualities of them are still worse than all of our algorithms even after the refinement. As seen in Table 5.4, solutions of multilevel algorithms are generally refined more than our single level algorithms. This is because of the fact that FM is more suitable for multilevel schemes, and its performance can be increased by imposing appropriate number of levels.

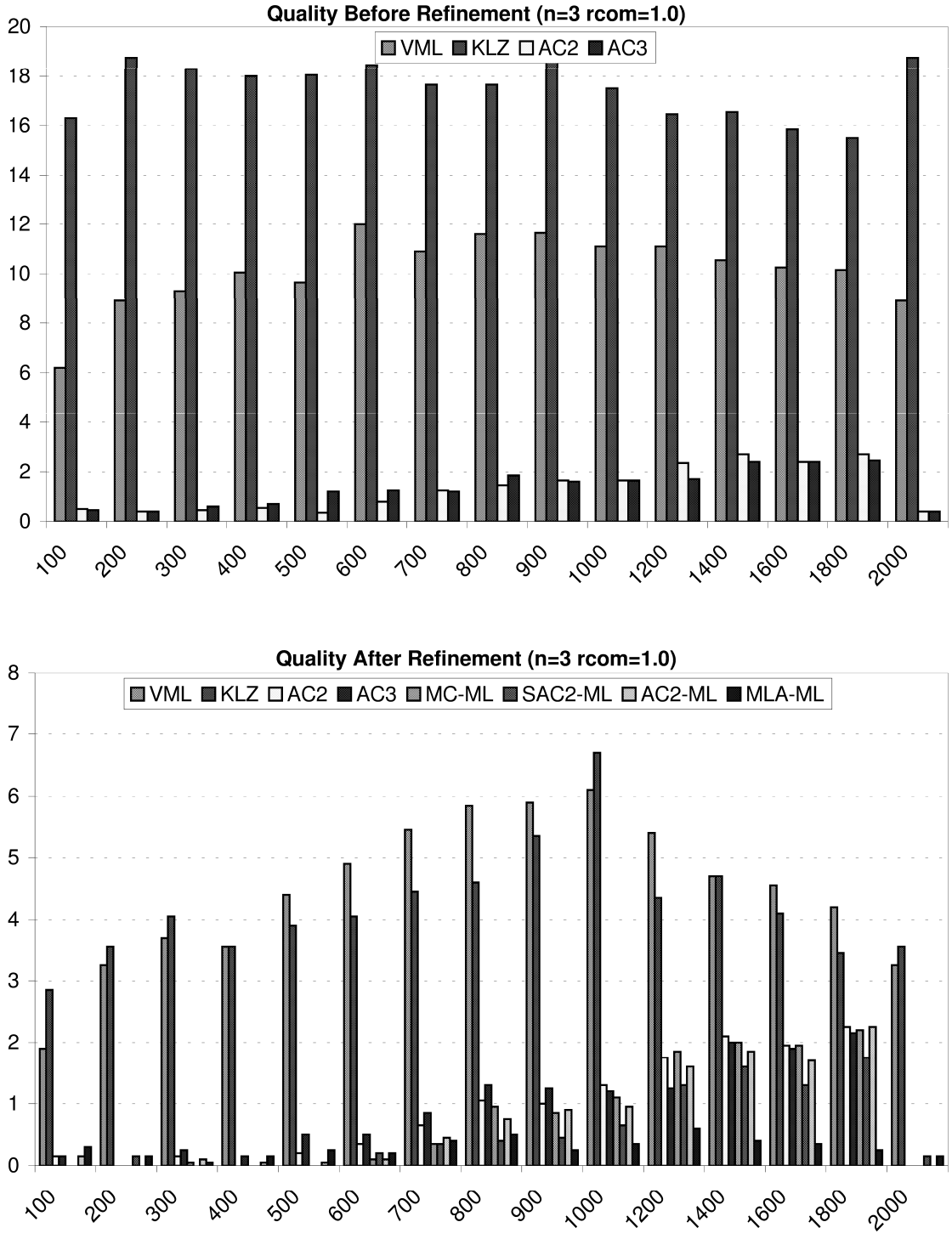


Figure 5.4: Percent qualities of algorithms for 3-processor systems in trees

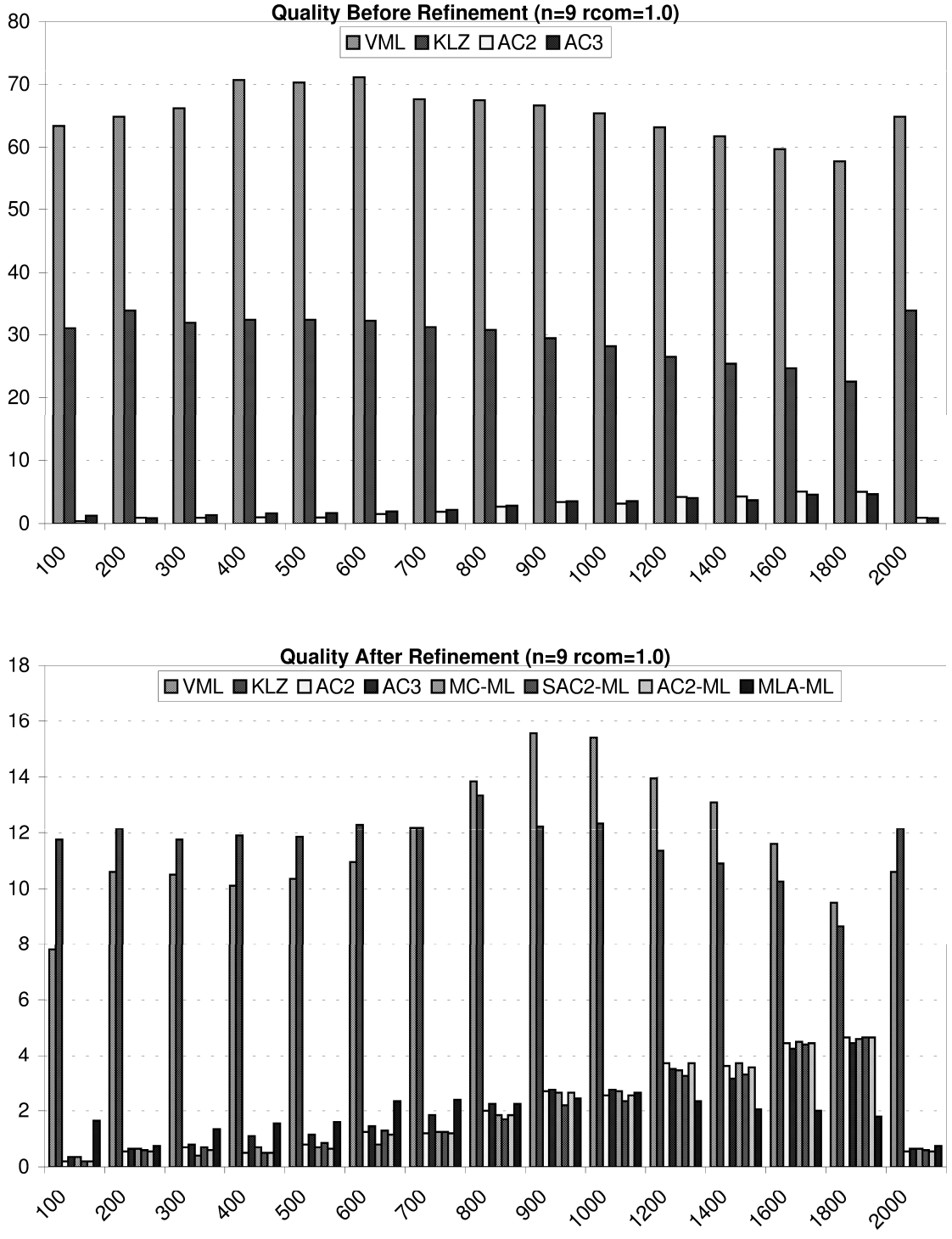


Figure 5.5: Percent qualities of algorithms for 9-processor systems in trees

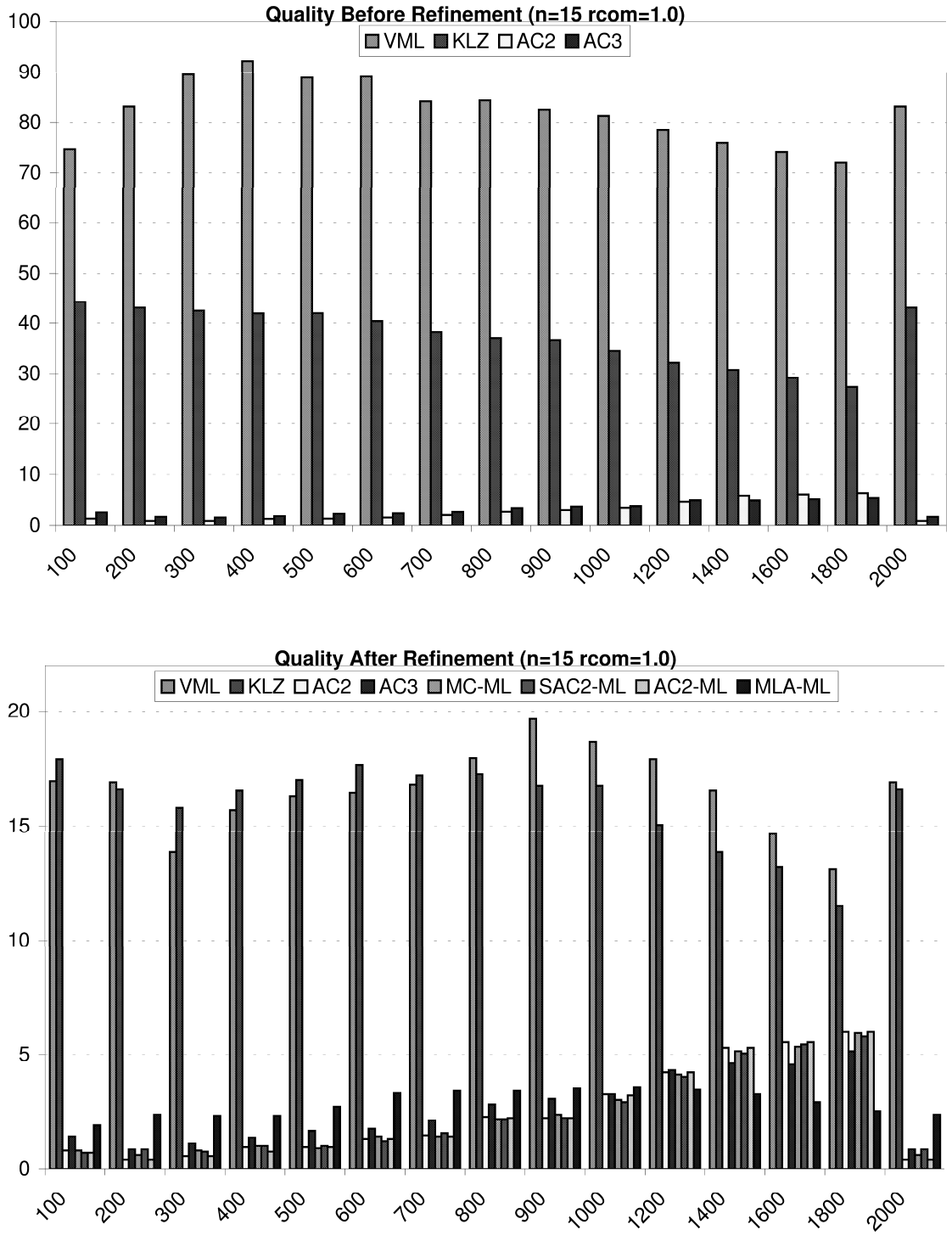


Figure 5.6: Percent qualities of algorithms for 15-processor systems in trees

		Existing Algorithms		Proposed Algorithms	
r_{com}	n	VML	KLZ	AC2	AC3
0.70	3	10.02	14.33	1.25	1.25
	6	66.78	22.34	1.92	1.77
	9	88.02	27.48	2.28	2.00
	12	102.04	31.16	2.29	2.23
	15	111.92	34.50	2.40	2.45
	18	120.94	37.63	2.52	2.56
1.00	3	10.15	17.49	1.31	1.35
	6	50.77	25.02	2.05	2.23
	9	65.41	29.74	2.40	2.55
	12	75.93	34.77	2.55	2.85
	15	82.25	37.53	2.80	3.18
	18	89.62	40.76	2.89	3.42
1.30	3	10.18	17.38	1.13	1.41
	6	41.49	24.47	1.89	2.29
	9	52.73	28.66	2.23	2.64
	12	61.00	33.24	2.33	2.92
	15	65.77	35.89	2.44	3.27
	18	71.61	39.16	2.49	3.37

Table 5.2: Averages of percent qualities of solutions provided by single level algorithms in trees

		Existing Algorithms		Proposed Algorithms					
r_{com}	n	VML	KLZ	AC2	AC3	MC-ML	SAC2-ML	AC2-ML	MLA-ML
0.70	3	2.41	1.54	0.81	0.76	0.74	0.68	0.73	0.30
	6	5.21	3.48	1.42	1.32	1.37	1.36	1.40	1.29
	9	6.72	5.15	1.80	1.61	1.74	1.78	1.81	1.91
	12	8.45	6.49	1.90	1.86	1.88	1.89	1.91	2.51
	15	10.10	7.84	2.03	2.07	2.07	2.08	2.02	2.94
	18	11.06	9.33	2.23	2.20	2.30	2.27	2.24	3.32
1.00	3	4.47	4.21	0.86	0.90	0.76	0.55	0.73	0.29
	6	8.96	8.38	1.48	1.64	1.45	1.41	1.41	1.21
	9	11.74	11.68	1.96	2.07	1.93	1.86	1.92	1.86
	12	14.08	13.95	2.14	2.29	2.20	2.13	2.12	2.40
	15	16.55	15.98	2.37	2.58	2.37	2.36	2.34	2.88
	18	17.48	17.77	2.49	2.82	2.66	2.61	2.49	3.07
1.30	3	5.64	6.54	0.76	0.93	0.68	0.38	0.62	0.23
	6	12.18	12.16	1.28	1.65	1.35	1.11	1.23	0.97
	9	16.01	15.54	1.73	2.08	1.76	1.49	1.65	1.57
	12	18.31	18.69	1.83	2.26	1.90	1.76	1.80	1.93
	15	21.47	20.76	1.96	2.60	2.05	1.82	1.92	2.41
	18	22.56	22.69	2.09	2.73	2.31	2.04	2.08	2.67

Table 5.3: Averages of percent qualities of refined solutions of the algorithms in trees

		Existing Algorithms		Proposed Algorithms					
r_{com}	n	VML	KLZ	AC2	AC3	MC-ML	SAC2-ML	AC2-ML	MLA-ML
0.70	3	7.10	12.33	0.09	0.11	0.34	0.19	0.16	0.00
	6	61.06	18.33	0.10	0.12	0.35	0.21	0.09	0.00
	9	80.79	21.81	0.09	0.05	0.39	0.16	0.14	0.00
	12	93.07	24.18	0.07	0.05	0.33	0.10	0.08	0.00
	15	101.23	26.20	0.04	0.05	0.27	0.06	0.04	0.00
	18	109.35	27.81	0.02	0.03	0.23	0.04	0.02	0.00
1.00	3	5.23	12.77	0.10	0.07	0.40	0.26	0.20	0.00
	6	41.34	16.16	0.17	0.16	0.49	0.28	0.25	0.00
	9	53.12	17.63	0.12	0.10	0.53	0.23	0.12	0.00
	12	61.36	20.34	0.07	0.13	0.40	0.12	0.08	0.00
	15	65.21	21.13	0.08	0.15	0.44	0.14	0.08	0.00
	18	71.53	22.51	0.08	0.17	0.35	0.11	0.08	0.01
1.30	3	3.99	10.40	0.04	0.07	0.39	0.31	0.09	0.00
	6	28.87	11.84	0.18	0.21	0.53	0.39	0.21	0.00
	9	36.25	12.64	0.12	0.13	0.67	0.27	0.15	0.00
	12	42.14	14.10	0.10	0.24	0.59	0.25	0.10	0.00
	15	43.80	14.62	0.15	0.25	0.60	0.29	0.18	0.00
	18	48.57	15.99	0.08	0.16	0.54	0.19	0.11	0.00

Table 5.4: Averages of percent refinements on the solutions of the algorithms in trees

5.5 Experiments with General TIGs

We have experimented the performance of the proposed task assignment algorithms on the problem instances with general TIGs. Figures 5.7, 5.8 and 5.9 illustrate the percent qualities of the proposed algorithms for 3, 9 and 15 processors systems with general TIGs respectively. Missing bars in Figs. 5.7-5.9 denote that the respective algorithm achieves the qualities of the best known assignments for all of the 20 problem instances with respective TIG topology. As in the case of tree topologies, VML performs better than KLZ for only 3-processor systems. As seen in Figs. 5.7-5.9, the proposed algorithms drastically outperform both VML and KLZ. The assignment qualities of the proposed algorithms are not affected from the number of tasks. As seen in Figs. 5.7-5.9, MLA-ML performs substantially better assignments than all of the proposed algorithms.

The relative solution qualities of all experimented algorithms are summarized in Tables 5.5 and 5.6. As seen in Table 5.5, our AC3 algorithm produces substantially better solutions than other single level algorithms. This is most probably due to fact that AC3 algorithm finds the 3-cliques of highly interacting tasks in TIGs. Usually, AC3 is expected to produce better clusters of tasks then AC2, and this power of AC3 improves the solution qualities as expected.

If we look at the Table 5.6, it can be noticed that MLA-ML produces solutions which are at most 0.88% worse than the best known solutions. This situation shows that FM scheme works well on task assignment problems whose TIGs are general graphs. This result is not surprising because, the experimental studies on graph partitioning problems [10] showed that FM works well for dense graphs in multilevel scheme. Another interesting observation in Table 5.6 is that the performance of MLA-ML monotonically decreases with increasing communication costs. This is most probably because of the fact that the coarsening gets more important in those problems. Since MLA-ML does not use coarsening to reduce the original TIG, it automatically produces worse solutions when r_{com} is increased.

As seen in Table 5.6, MC-ML produces better solutions than SAC2-ML and

AC2-ML. This is most probably because of the fact that MC-ML has more levels than both of SAC2-ML and AC2-ML for the same problem. So, it best uses the power of FM scheme in refinement steps to produce better solutions. In fact, as seen in Table 5.7, FM improves the initial assignments of MC-ML more than the initial assignments of SAC2-ML and AC2-ML.

As seen in Table 5.5, VML and KLZ produces substantially worse solutions than all other algorithms. Although FM improves the solutions of them very well, they both give the worst solution qualities even after the refinement in all of the cases.

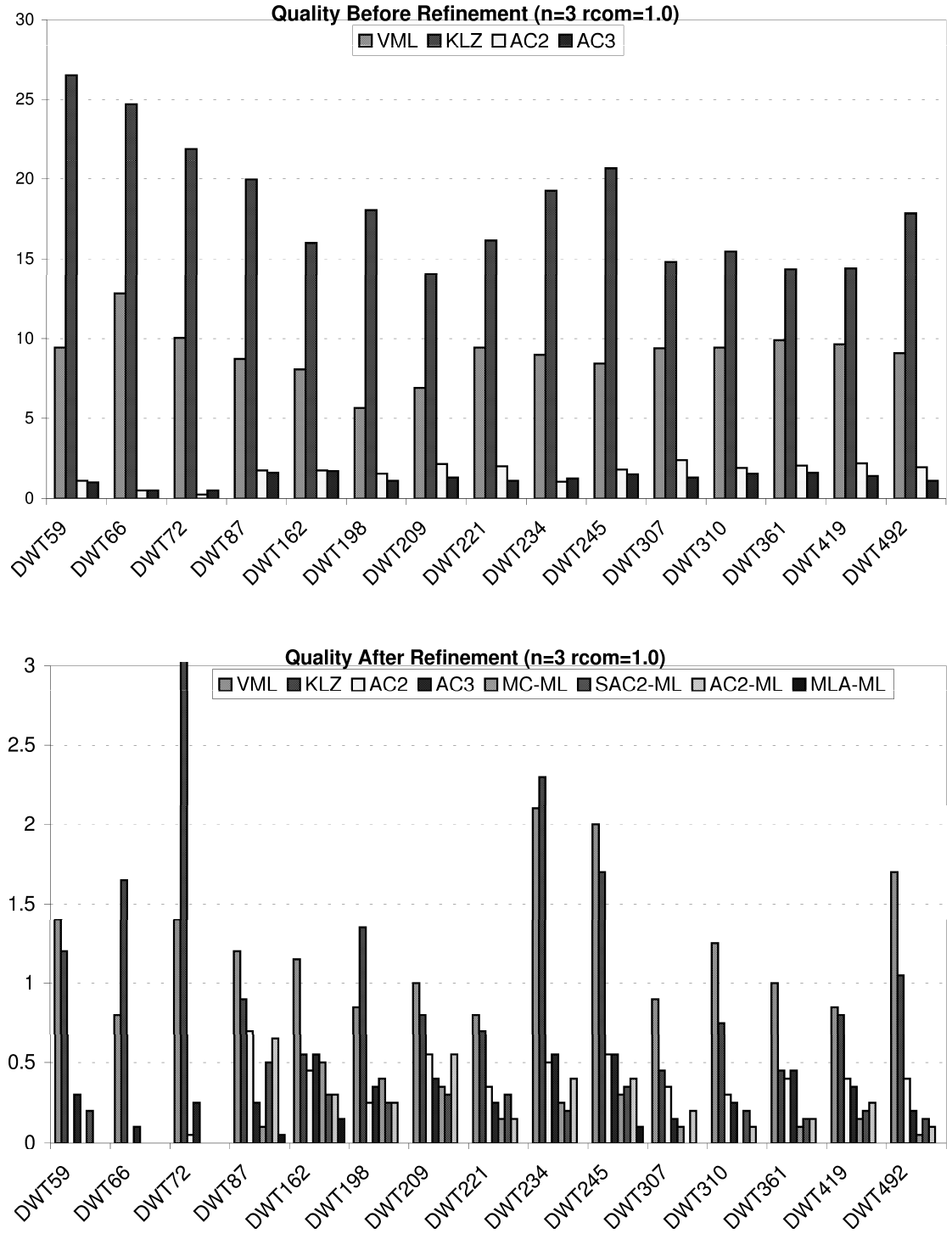


Figure 5.7: Percent qualities of algorithms for 3-processor systems in general graphs

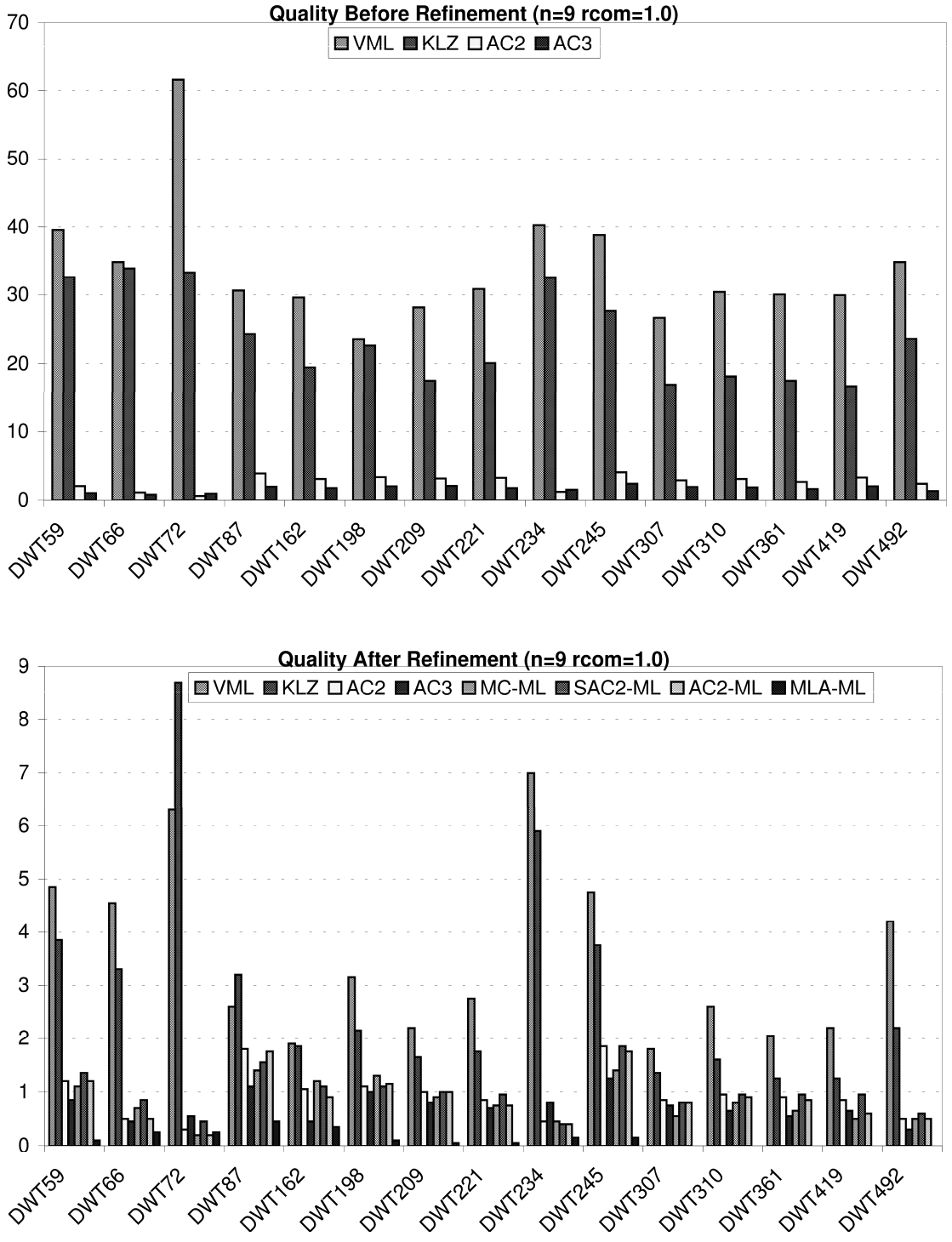


Figure 5.8: Percent qualities of algorithms for 9-processor systems in general graphs

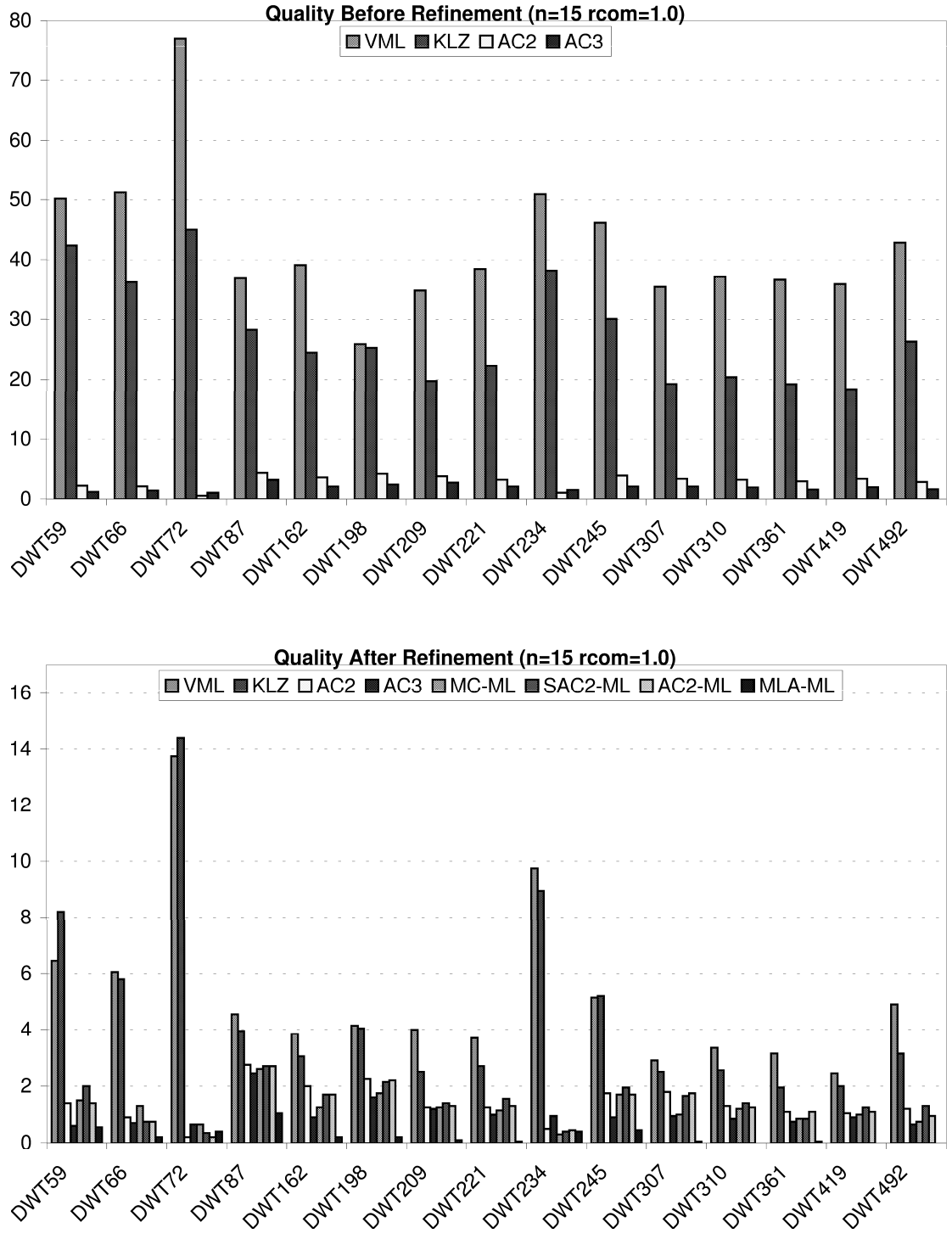


Figure 5.9: Percent qualities of algorithms for 15-processor systems in general graphs

		Existing Algorithms		Proposed Algorithms	
r_{com}	n	VML	KLZ	AC2	AC3
0.70	3	12.04	9.98	0.96	0.58
	6	45.35	12.61	1.20	0.78
	9	58.55	14.15	1.26	0.75
	12	66.81	15.68	1.36	0.83
	15	73.16	16.86	1.28	0.77
	18	77.98	18.12	1.35	0.79
1.00	3	9.08	18.27	1.63	1.23
	6	26.68	21.97	2.48	1.54
	9	33.98	23.79	2.67	1.66
	12	38.74	26.15	2.87	1.81
	15	42.52	27.68	2.98	1.91
	18	45.68	29.68	2.91	1.80
1.30	3	6.04	27.53	1.76	1.30
	6	16.72	32.52	3.30	2.04
	9	20.94	34.94	3.82	2.30
	12	24.07	37.72	3.87	2.50
	15	26.80	39.36	4.36	2.80
	18	29.01	41.66	4.07	2.66

Table 5.5: Averages of percent qualities of solutions provided by single level algorithms in general graphs

		Existing Algorithms		Proposed Algorithms					
r_{com}	n	VML	KLZ	AC2	AC3	MC-ML	SAC2-ML	AC2-ML	MLA-ML
0.70	3	0.31	0.21	0.07	0.03	0.03	0.05	0.04	0.01
	6	0.81	0.55	0.15	0.13	0.10	0.13	0.13	0.03
	9	1.20	0.91	0.24	0.15	0.19	0.23	0.24	0.04
	12	1.76	1.36	0.33	0.23	0.29	0.34	0.32	0.09
	15	2.24	1.80	0.33	0.21	0.36	0.38	0.32	0.09
	18	2.46	2.12	0.50	0.29	0.51	0.48	0.48	0.13
1.00	3	1.23	1.24	0.35	0.33	0.16	0.21	0.23	0.02
	6	2.75	2.15	0.73	0.50	0.47	0.64	0.63	0.10
	9	3.53	2.92	0.94	0.72	0.83	0.99	0.88	0.13
	12	4.38	3.84	1.16	0.85	1.08	1.16	1.10	0.21
	15	5.21	4.73	1.38	1.00	1.22	1.43	1.32	0.25
	18	5.89	5.11	1.45	1.00	1.27	1.44	1.41	0.27
1.30	3	2.13	2.63	0.51	0.49	0.35	0.36	0.43	0.08
	6	4.27	4.46	1.20	0.95	0.81	0.98	1.02	0.32
	9	5.94	5.71	1.69	1.16	1.46	1.61	1.61	0.48
	12	7.32	6.56	1.72	1.32	1.54	1.71	1.59	0.61
	15	7.99	8.22	2.18	1.64	1.87	2.07	2.04	0.88
	18	9.26	8.85	2.06	1.58	1.84	1.99	1.94	0.81

Table 5.6: Averages of percent qualities of refined solutions of the algorithms in general graphs

		Existing Algorithms		Proposed Algorithms					
r_{com}	n	VML	KLZ	AC2	AC3	MC-ML	SAC2-ML	AC2-ML	MLA-ML
0.70	3	11.40	9.35	0.66	0.29	2.06	1.05	0.68	0.00
	6	44.09	11.60	0.74	0.29	1.89	0.90	0.75	0.00
	9	56.90	12.73	0.62	0.27	1.59	0.76	0.62	0.00
	12	64.58	13.83	0.64	0.25	1.54	0.70	0.64	0.01
	15	70.38	14.56	0.55	0.25	1.47	0.63	0.56	0.00
	18	75.03	15.54	0.48	0.17	1.47	0.59	0.48	0.01
1.00	3	7.31	16.53	0.93	0.49	2.96	1.72	1.08	0.00
	6	23.46	19.37	1.38	0.64	3.33	1.83	1.49	0.00
	9	30.00	20.37	1.31	0.53	2.98	1.65	1.37	0.03
	12	33.81	21.85	1.29	0.57	2.83	1.62	1.37	0.02
	15	36.78	22.46	1.18	0.49	2.89	1.55	1.22	0.02
	18	39.31	24.09	1.07	0.37	2.76	1.42	1.11	0.07
1.30	3	3.46	24.37	0.91	0.48	2.77	1.72	0.98	0.01
	6	11.95	27.52	1.73	0.67	3.88	2.65	1.91	0.11
	9	14.53	28.76	1.72	0.74	4.02	2.42	1.82	0.25
	12	16.26	30.71	1.74	0.74	3.69	2.39	1.86	0.21
	15	18.28	30.64	1.68	0.73	3.79	2.36	1.83	0.30
	18	19.24	32.28	1.58	0.72	3.83	2.23	1.70	0.31

Table 5.7: Averages of percent refinements on solutions of the algorithms in general graphs

5.6 Run-Time Performance of The Proposed Algorithms

All of the algorithms are implemented and run on a workstation equipped with a 133 MHz PowerPC processor with 512-Kbyte external cache and 64 Mbytes of memory. The average running times of the proposed algorithms are summarized in Fig. 5.10. All running times are normalized according to average running times of KLZ. VML is not included in Fig. 5.10 for the sake of scaling because, its running time is very large relative to others. As seen in Fig. 5.10, average running times of our multilevel assignment algorithms are comparable with KLZ. Only MLA-ML needs substantially more computation time, but this can be reduced by using less than 5 random solutions at each level. Among the single level assignment algorithms, AC2 needs approximately 30% less computation time than KLZ on average. But AC3 needs approximately 25% more computation time on average. This is because of the search carried out to find the best triple clustering alternative for each task in AC3.

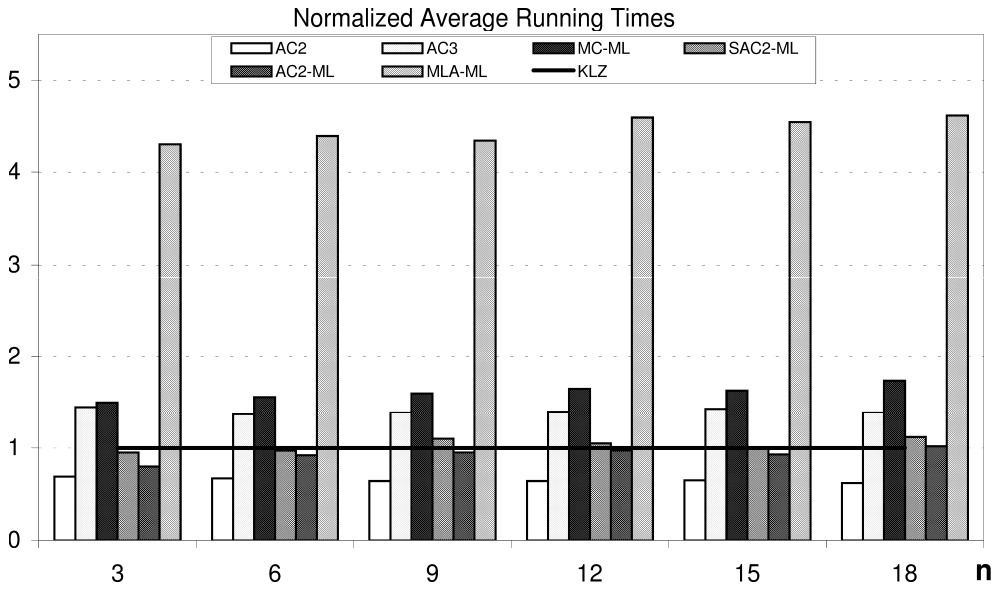


Figure 5.10: Normalized average running times of the proposed algorithms

Chapter 6

Conclusion

In this work, we have investigated the task assignment problem in distributed systems. We have proposed efficient clustering and assignment schemes for two-phase assignment algorithms based on an optimistic clustering metric which considers the differences between the execution times of tasks to be clustered. We have evaluated the validity of our clustering and assignment schemes through an experimental study. In these experiments, we have generated random problem instances whose TIGs are trees and general graphs. For the problem instances whose TIGs are trees, our algorithms produced assignments which are at most 3% worse than the optimal solutions. For general TIGs, our algorithms produced assignments which are very close to the optimal solutions. These results are very encouraging because, at significantly lower execution time, we obtain assignments with costs very close to the optimal assignments.

We have also adapted the multilevel scheme used in graph/hypergraph partitioning to task assignment. Experimental results indicated that the multilevel assignment algorithms perform well on a variety of task-processor systems. In the refinement phase of the multilevel algorithms, we have used a variation of FM. In this version of FM, we have introduced task reassignment concept instead of vertex move in graph/hypergraph partitioning. We have also used the FM algorithm in the refinement of the assignments provided by single level algorithms. Experimental results showed that our FM refinement significantly

improved the assignment qualities of proposed algorithms especially for relatively dense TIGs.

We should note that in this work, we have focused on only the problem of minimizing total execution and communication costs of an assignment. However, other issues such as load balancing, memory restrictions, queuing delays, precedence constraints and communication link loads can also be taken into consideration. An obvious extension to this research is to increase the complexity of the proposed methods to include such factors. Thus, task assignment problem continues to offer a wide variety of challenging problems.

Bibliography

[1] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. on Software Engineering*, vol. SE-3, january 1977.

[2] V. F. Magirou and J. Z. Milis, "An algorithm for the multiprocessor assignment problem," *Operations Research Letters*, December 1989.

[3] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE trans. on Computers*, vol. 37, November 1988.

[4] M. S. Chern, G. H. Chen, and P. Liu, "An lc branch-and-bound algorithm for module assignment problem," *Inform. Process. Lett.*, vol. 32, pp. 61–71, july 1989.

[5] V. F. Magirou, "An improved partial solution to the task assignment and multiway cut problems," *Operations research letters*, vol. 12, 1992.

[6] Y. Kopidakis, M. Lamari, and V. Zissimopoulos, "On the task assignment problem : Two new heuristic algorithms," *Journal of Parallel and Distributed Computing*, vol. 42, 1997.

[7] K. Efe, "Heuristic models of task assignment scheduling in distributed systems," *IEEE trans. on computer*, june 1982.

[8] E. A. Williams, "Design analysis and implementation of distributed systems from a performance perspective," *Ph.D. dissertation*, 1983.

[9] N. S. Bowen, C. N. Nikolaou, and A. Ghafoor, "On the assignment problem of arbitrary process systems to heterogeneous distributed computer systems," *IEEE trans. on computer*, vol. 41, march 1992.

[10] G. Karypis and V. Kumar, "Metis : Unstructured graph partitioning and sparse matrix ordering system." Dept. of Computer Science, University of Minesota. Http://www.cs.umn.edu/ karypis.

[11] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell system Technical Report*, vol. 49, 1970.

[12] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," *19th ACM/IEEE Design Automation Conf.*, 1982.

[13] V. Maniezzo and M. Dorigo, "Algodesk: an experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem," *European Journal of Operational Research*, vol. 81, 1995.

[14] S. H. Bokhari, "A shortest tree algorithm for optimal assignments across space and time in distributed processor system," *IEEE trans. on software Engineering*, vol. SE-7, no. 6, 1981.

[15] D. Towsley, "Allocating programs containing branches and loops within a multiple processor system," *IEEE Trans. on Software Engineering*, vol. SE-12, October 1986.

[16] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE trans. on software engineering*, vol. 15, november 1989.

[17] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," *Tech. Rep., Sandia National Laboratories*, 1993.