JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **42,** 21–29 (1997)
ARTICLE NO. PC971311

On the Task Assignment Problem: Two New Efficient Heuristic Algorithms

Y. Kopidakis, M. Lamari, and V. Zissimopoulos [1]

Laboratoire de Recherche en Informatique, CNRS-URA 410 "Al Khowarizmi," Université de Paris Sud, Centre d'Orsay, 91405 Orsay, France

We study the problem of task allocation in heterogeneous distributed systems. The objective is the minimization of the sum of processor execution and intertask communication costs. We transform the problem to a maximization one, where we try to determine and avoid large communication costs and inefficient allocations. After an appropriate graph transformation, we propose two fast algorithms, the Matching based and Max Edge heuristics, in order to consider tradeoffs between task clustering and task processor assignment. Their performance is evaluated through an experimental study where solution quality is compared with one of the best up-to-date heuristics for the problem. Results prove that algorithm Max Edge provides greatly improved solutions within short computational time even for large size instances.

1. INTRODUCTION

The performance of heterogeneous distributed systems heavily depends on the way tasks composing a parallel program are allocated to processors. Significant research has been done on the task allocation problem and different models have been proposed. In this paper, we consider the problem of assigning tasks to processors in a distributed system in order to minimize the sum of interprocessor communication and task processing costs. Processors are considered heterogeneous, having different processing capabilities depending on task nature and each processor communicates with all others through identical communication channels. The problem was first introduced by Stone in [18] and is known to be NP-hard in general [2]. Nevertheless, polynomial time optimal algorithms exist for two-processor systems [18] and in the case where the intertask communication graph is a tree [2], a series-parallel graph [20] or a k-tree [6]. For the general problem, branch and bound exact algorithms have been studied ([1, 13]) and several heuristic methods have been proposed in [11, 14, 17]. From the theoretical point of view, Fernandez-Baca [6] showed the unfeasibility of a polynomial time approximation scheme, but the question of existence of any approximation guarantee for the problem is still open. Recently, efforts towards this direction are reported. In [9], an approximation scheme is presented in the case of complete communication of constant value, along with polynomial algorithms for bivalued execution times or bounded number of processors. An interesting average case analysis of random versions of the problem with special value costs is presented in [10]. In addition, in [16], a heuristic based on partial task allocations is proposed and evaluated.

In this paper, we transform the initial problem of minimization of total communication and execution costs into a maximization one, where we try to determine and avoid large communication and execution penalties. After the model description in Section 2, we present in Section 3 an appropriate graph transformation for the maximization problem in question. In Section 4, a new heuristic based on maximum weight graph matching is introduced in order to determine efficient solutions, considering tradeoffs between task grouping and task processor allocations. In Section 5, we describe the Max Edge heuristic, an alternative algorithm that functions on the transformed graph in a more careful and constructive way. Finally, in Section 6 we present experimental results which prove the quality and the efficiency of the two methods proposed.

2. MODEL DESCRIPTION

Various assumptions made about the distributed computing system, considered in this paper, are described hereafter.

(1) The processors in the system are heterogeneous. In other words, a task if executed on different processors, will require different amounts of running time.

(2) We consider identical communication links between processors and thus, uniform communication costs. That is, the task communication times are processor independent and two tasks assigned to the same processor do not introduce any communication cost.

Let $P = \{P_j,\ j = 1, 2, \ldots, n\}$ be the set of processors of a distributed computing system, $T = \{T_i,\ i = 1, 2, \ldots, m\}$ the set of tasks to be allocated and $G(V, E)$ be the intertask communication graph. Let e_{ip} represent the cost of executing task i on processor p, c_{ij} the communication cost between tasks i and j and x_{ip}, $i = 1 \cdots m$, $p = 1 \cdots n$, a 0, 1 variable, equal to 1 if task i runs on processor p and 0 otherwise.

[1]E-mail: vassilis@lri.lri.fr.

0743-7315/97 $25.00

Based on the above assumptions, the cost function and the constraints for the task assignment problem are described in the following:

$$(\mathrm{P})\begin{cases} \text{Min } \sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip}x_{ip} \\ \quad + \sum_{(i,\,j)\in E}\sum_{p=1}^{n} c_{ij}x_{ip}(1-x_{jp}) \\ \sum_{p=1}^{n} x_{ip} = 1, \qquad i = 1\cdots m \quad (1) \\ x_{ip}\in\{0,\,1\}, \qquad i = 1\cdots m,\ p = 1\cdots n \end{cases}$$

The constraint (1) expresses the fact that each task must be assigned to one processor.

Note that the described assignment problem is similar to other assignment problems, such as the allocation of researchers into departments [7] and the famous quadratic assignment problem [15]. In addition, it is closely related to graph partitioning problems: the multiway-cut problem, [3], the k-cut problem [8], and the k-partition problem [5]. This relation is exploited in [12] for a task assignment formulation with slightly different objectives. In the same work, the maximum weight matching algorithm is applied, in a different perspective than in our work.

3. GRAPH TRANSFORMATION

We now consider the objective function of the previous section:

$$\min \sum_{(i,\,j)\in E}\sum_{p=1}^{n} c_{ij}x_{ip}(1-x_{jp}) + \sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip}x_{ip}. \quad (1)$$

In what follows, a transformation of the problem is proposed. The minimization of total communication and execution costs can be viewed as a maximization problem where we try to locate and avoid two kinds of penalties: heavy communication between tasks and inefficient allocations. In other words, instead of determining low cost allocations with small communication costs, we are searching for high cost allocations and task communications to avoid. Formally, (1) is equivalent to

$$\min \sum_{(i,\,j)\in E} c_{ij}\sum_{p=1}^{n} x_{ip} - \sum_{(i,\,j)\in E} c_{ij}\sum_{p=1}^{n} x_{ip}x_{jp} - \sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip}(1-x_{ip}) + \sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip}. \quad (2)$$

Let $y_{ij} = \sum_{p=1}^{n} x_{ip}x_{jp}$, where y_{ij} determines if two tasks are allocated to the same processor. Clearly, $y_{ij} = 1$ if tasks i, j are grouped on the same processor and $y_{ij} = 0$ otherwise. Given that $\sum_{p=1}^{n} x_{ip} = 1$, (2) can be reformulated as follows:

$$\min \sum_{(i,\,j)\in E} c_{ij} - \sum_{(i,\,j)\in E} c_{ij}y_{ij} + \sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip} - \sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip}(1-x_{ip}). \quad (3)$$

Clearly, $\sum_{(i,\,j)\in E} c_{ij}$ and $\sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip}$ are two large value constants, for total communication and total execution costs respectively. So we can transform the initial minimization problem to an equivalent maximization problem:

$$\max \sum_{(i,\,j)\in E} c_{ij}y_{ij} + \sum_{i=1}^{m}\sum_{p=1}^{n} e_{ip}(1-x_{ip}) \quad (4)$$

Since y_{ij} equals 1 only if tasks i and j are assigned to the same processor, the first term in (4) corresponds to the total communication cost for clustered tasks. Thus, in the transformed problem, we try to maximize the communication of clustered tasks, which is similar to minimizing the communication cost of tasks not clustered in the initial problem. At the same time, in (4), we try to maximize the competitive second term, which sums the cost of allocations. Clearly, the new maximization problem is NP-complete, since it is equivalent to the initial NP-complete minimization problem.

For the above maximization problem, a graph transformation is proposed where a special node for each processor is added to the initial task graph. We define from $G(V,\,E)$ a new graph $G'(V',\,E')$, where $V' = V \cup P$. For each task–task edge in E', we consider weight $c'_{ij} = c_{ij}$, $i,\,j \in T$, corresponding to the first term of (4). For each task–processor couple $(i,\,k)$, $i \in T$, $k \in P$, we add an edge in E' with weight c'_{ik}. This weight should express the term $\sum_{p=1}^{n} e_{ip}(1-x_{ip})$ of (4), corresponding to the penalty that the assignment probably introduces. However, in order to be able to consider tradeoffs between task clustering and task–processor allocations, we normalize weights for task–processor edges towards the size of a single execution cost, preserving however the relative order of cost introduced. Thus, we define edge weight c'_{ik} as follows:

$$c'_{ik} = \frac{\sum_{p=1}^{n} e_{ip} - e_{ik}}{n-1}. \quad (5)$$

Weight c'_{ik} can be viewed as the mean penalty in the case where task i is assigned to one of $1,\,\ldots,\,k-1,\,k+1,\,\ldots,\,n$ processors.

The technique of additional processor nodes was first proposed by Stone in [18] and has been exploited in [11, 13, 16]. Weight $((\sum_{p=1}^{n} e_{ip})/(n-1)) - e_{ik}$ was associated to edge connecting task i to processor k. In that case, the problem is equivalent to the well-known multiway cut problem where,

given a graph and a set of terminal nodes, one tries to find a minimum weight set of edges (cut) which disconnects each terminal from the others. However, it fails in providing any approximation guarantee for the task allocation problem, even though there exists a $2-2/n$ approximation algorithm for the n-cut problem ([3]). The reason is that the weights defined above could take negative values.

In our approach, we defined task processor edge weights in a different perspective. They incorporate tradeoffs between task–task clusterings and task–processor allocations. In addition, the algorithms proposed in what follows exploit the fact that c'_{ik} in (5) are always positive.

Figure 1 illustrates this model in the case where the distributed computing system is composed of 3 tasks $\{T_1, T_2, T_3\}$ and 2 processors $\{P_1, P_2\}$. On the initial graph G, edge weights represent the intertask communication cost. Execution times on processors P_1 and P_2 are respectively $T_1(7, 2)$, $T_2(1, 6)$, $T_3(7, 2)$.

4. MATCHING BASED HEURISTIC

In this section, a graph matching approach is proposed for solving the task assignment problem. We first apply the graph transformation presented above. Then, in order to make independent and disjointed decisions for grouping and assignment, we use the maximum weight matching algorithm which functions in a fast and parallel way, eliminating several edges and allowing multiple decisions at a time.

Let $G(V, E)$ be an undirected graph, where V is the set of vertices and E is the set of edges. Let P_{ij} the weight of the edge $(i, j) \in E$. A matching M of a graph $G(V, E)$ is a subset of the edges $M \subseteq E$ with the property that no two edges of M share the same node. Edges of M are called *matched* edges. A maximum weighted matching is a matching of G with the largest possible sum of weights, i.e., $\sum_{(i,j)\in M} p_{ij}$ is maximum.

At each step k of the algorithm, we determine the maximum weighted matching M in the graph G^k ($G^1 = G'$). Each matched edge is then contracted. A (*task, task*) type edge contracted defines the clustering of the two tasks and a (*task, processor*) type edge defines an allocation of the task to the processor. In the first case, we avoid the penalty that would be introduced in the objective function by the task communication cost and in the second case, the penalty that would be introduced by the nonallocation of the task to the processor.

This contraction reduces the graph G^k to a graph G^{k+1}, where new edge weights c^{k+1} are defined from c^k. For each processor, the execution cost of two grouped tasks is equal to the sum of the execution costs of the two tasks. The communication cost between a task and the clustered tasks is equal to the sum of the corresponding communication costs. The edge strength between a task t and a task–processor matched edge (i, p), c^{k+1}_{tp}, is equal to the execution cost of the task t on the processor p, c^k_{tp}, plus the intertask communication cost c^k_{it}. After the allocation, task i and processor p constitute one node, corresponding to p. Thus, all edges connecting i and the other processors are deleted.

MATCHING BASED ALGORITHM.

$G^1 = G'$, $k := 1$, $c^k_{ij} := c'_{ij}$
repeat
— Find a maximum weighted matching M on G^k
— G^{k+1} is the new graph obtained by contracting edges $(i, j) \in M$
if $(i, j) \in M$ and $i, j \in T$ then group tasks i and j into a new task l. Let $c^{k+1}_{lt} := c^k_{it} + C^k_{jt}$, $\forall t \in T$, $t \neq i$, $t \neq j$ and $c^{k+1}_{lp} := c^k_{ip} + c^k_{jp}$ $forall\ p \in P$
if $(i, p) \in M$ and $i \in T$, $p \in P$ then assign the task i to the processor p. Let $c^{k+1}_{tp} := c^k_{tp} + c^k_{it}$, $\forall t \neq i$ and delete edges c^k_{iq}, $\forall q \in P$, $q \neq p$
— $k := k + 1$;
until all edges are removed (all tasks are allocated)

Obviously, the algorithm leads to a complete assignment of tasks to processors, since it deletes an edge only if the corresponding task (or cluster of tasks) is allocated. Thus, the algorithm always provides a feasible suboptimal solution for the transformed maximization problem and, consequently, for the initial minimization one.

Figure 2 presents the maximum weighted matching, drawn in bold, in the modified graph $G'(V', E')$ of Fig. 1. The edges (T_2, P_1) and (T_1, T_3) define the assignment of T_2 to P_1 and the clustering of T_1 and T_3 respectively. After edge contraction, the cluster (T_1, T_3) will be assigned to processor P_2 in the next step of the algorithm.

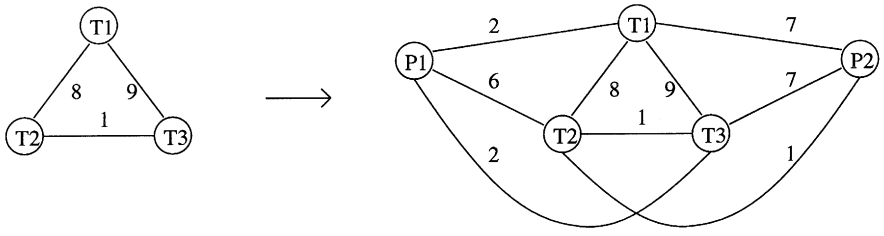

FIG. 1. The communication graph $G(V, E)$, transformed in $G'(V', E')$ on the right.

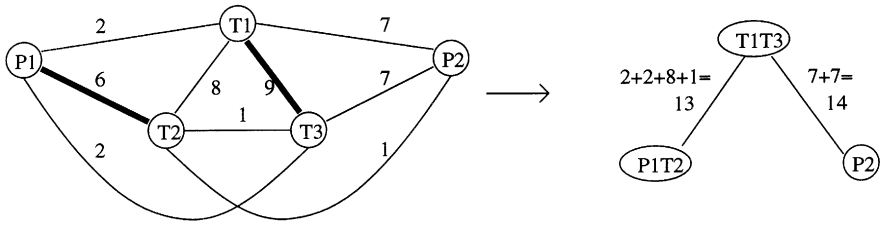

FIG. 2. Maximum weighted matching and edge contraction in $G'(V', E')$.

The complexity of the matching based algorithm described above is $O(m(m+n)^2)$, since the repeat loop is executed $O(m+n)$ times ($m+n$ is the total number of nodes in G') and the most costly operation inside the loop is graph edge update taking $O(m(m+n))$ time. We assume here that, for determining a maximum weighted matching, we use a simple algorithm which finds the edge of maximum weight, deletes its neighbors and proceeds in the same way. Assuming that edges are sorted in lists, all this can be done in $O((m+n)m)$ time, which is the maximum number of edges of G'.

5. THE MAXIMUM EDGE ALGORITHM

We now present an alternative method in order to determine clustering and allocation on the modified graph of Section 3. In the Matching based heuristic, we considered simultaneous clusterings and allocations provided by the determined matching at each step. In the Max Edge heuristic, we consider the clustering or allocation provided by the maximum weight edge of the transformed graph, contracting only one graph edge at a time and thus proceeding in a greedy, but more careful and slow way at each step. The description of the Max Edge heuristic follows:

MAXIMUM EDGE ALGORITHM.

repeat
— on the transformed graph $G'(V', E')$ find
edge (i, j) with:
$c'_{ij} = \max\{c'_{uv}, (v, u) \in E'\}$
— delete edge (i, j)
if i, j tasks, then group tasks in a new task k
and set $c'_{kl} = \max\{c'_{il}, c'_{jl}\}, \forall l \in V'$
if i task and j processor, allocate i to j, delete
edges $(i, p), \forall p \in P, p \neq j$ and set
$c'_{lj} = \max\{c'_{li}, c'_{lj}\}, \forall l \in T, l \neq i$
until all edges are removed (all tasks are allocated)

Notice that weight c'_{kl} for task l and added node k replacing i and j is set to $\max\{c'_{il}, c'_{jl}\}$ and not to $c'_{il} + c'_{jl}$, as in the corresponding case in the Matching based heuristic of the previous section. The reason is that the above sum would automatically create a large value weight and the algorithm would proceed in the next step by contracting an edge neighbor to the one just contracted. Choosing the max weight prevents solution polarization towards a certain region of the graph and allows progressive algorithm function.

The complexity of the Maximum Edge Algorithm is $O(m(m+n)^2)$, as in the Matching based algorithm. The repeat loop is executed at most $O(m+n)$ times and edge weight updates in the repeat loop require $O(m(m+n))$ time.

6. COMPARATIVE STUDY

The performance of the proposed algorithms was evaluated through an experimental study on randomly generated problem instances. In our study, the algorithm VML proposed by Virginia Mary Lo [11] was chosen among existing methods as one of the most general and effective. The method consists essentially of three stages. In the first stage, a partial task assignment is determined by considering repeatedly two processor networks formed by one processor and then a superprocessor representing the $n-1$ processors. In the second stage, the possibility of assigning all remaining tasks to the same processor is investigated. Finally, in the third stage, strongly communicating tasks form groups for which lower cost allocations are determined. The complexity of the proposed algorithm is $O(nm^4 \log m)$.

During test data generation, execution costs were randomly generated integers following a uniform distribution within the interval [1, 100]. Similarly, communication costs were uniformly distributed integers in an interval which varied in order to estimate the impact of the relative size of execution and communication costs. During instance generation, the following parameters were considered:

- m: the number of tasks
- n: the number of processors
- the density of the task graph, defined as the probability of existence of an edge between any two nodes of the task graph (task graph generation guaranteed the connectivity of the graph independently from graph density)
- r_{com}: the communication ratio defined as the ratio of the mean communication on the mean execution cost (r_{com} = 0.5 implies communication and execution costs uniformly distributed in [1, 50] and [1, 100] respectively)

For each choice of the above parameters, 20 problem instances were generated and solved by the three algorithms considered: the Matching heuristic, the Max Edge heuristic, and the reference algorithm VML heuristic. In order to evaluate the performance of the methods, the percentage of cases where each algorithm provided better solutions than the

others was measured. As a measure of solution quality, the relative distance from the best solution was calculated for each of the three algorithms. For algorithm A, the relative distance from the best solution is defined as

$$\mathrm{rd}_A = \frac{S_A - S^*}{S^*},$$

where S_A is the solution provided by algorithm A and S^* is the best solution provided by the three methods. Clearly, $\mathrm{rd}_A = 0$, if A has provided the best solution for the specific problem instance. The mean relative distance of each algorithm over the different problem instances is considered through the present section.

Experimental studies proved that algorithm Max Edge outperforms both VML and Matching heuristics. For equal mean execution and communication costs ($r_{\mathrm{com}} = 1$) and for different graph densities, Max Edge provided better solutions than VML in 60–80% of cases on small size instances ($m \times n = 5 \times 3,\ 10 \times 7$) and in 100% of cases on large size instances ($m \times n = 20 \times 10,\ 30 \times 15,\ 50 \times 20$). In contrast, Matching provided better solutions than VML only in 10% of cases for small instances. For large instances its performance was always inferior. Percentage on solution comparisons between couples of algorithms for $r_{\mathrm{com}} = 1$ are provided in Table I.

TABLE I
Percentage (%) of Cases Where Solutions Were Better or Worse for Each Pair of Algorithms for $r_{\mathrm{com}} = 1$ and for Varying Graph Density p (Cases of Equality Correspond to Percentages Which Do Not Sum to 100%)

		Max Edge		Max Edge		Matching	
		better	worse	better	worse	better	worse
$m \times n$	p	(than VML)		(than Matching)		(than VML)	
(5,3)	0.3	60	20	95	5	10	90
	0.5	75	15	100	0	20	75
	0.8	50	25	100	0	15	85
(10,7)	0.3	95	5	100	0	0	100
	0.5	100	0	100	0	0	100
	0.8	95	5	100	0	20	80
(20,10)	0.3	100	0	100	0	0	100
	0.5	100	0	100	0	0	100
	0.8	100	0	100	0	10	90
(30,15)	0.3	100	0	100	0	0	100
	0.5	100	0	100	0	0	100
	0.8	100	0	100	0	0	100
(50,20)	0.3	100	0	100	0	0	100
	0.5	100	0	100	0	0	100
	0.8	100	0	100	0	0	100

The mean relative distance for each algorithm and for the different values of graph density is presented in Fig. 3 when $r_{\mathrm{com}} = 1$. The exact measures corresponding to the curves can be found in Table II. The quality of the solutions provided by the Max Edge heuristic is obviously and constantly superior. The Matching heuristic is outperformed by both the Max Edge and the VML heuristic. As the gap between relative distances grows with the problem size, Max Edge provides impressively better solutions for large size instances (30–50 tasks on 15–20 processors). Note that the performance of all three algorithms does not depend on the graph density p. Results on solution comparison (Table I) and fluctuations of relative distance (Fig. 3 and Table II) for each algorithm are quite similar for different density values. Since the same behavior was constantly observed for different r_{com} values, in what follows

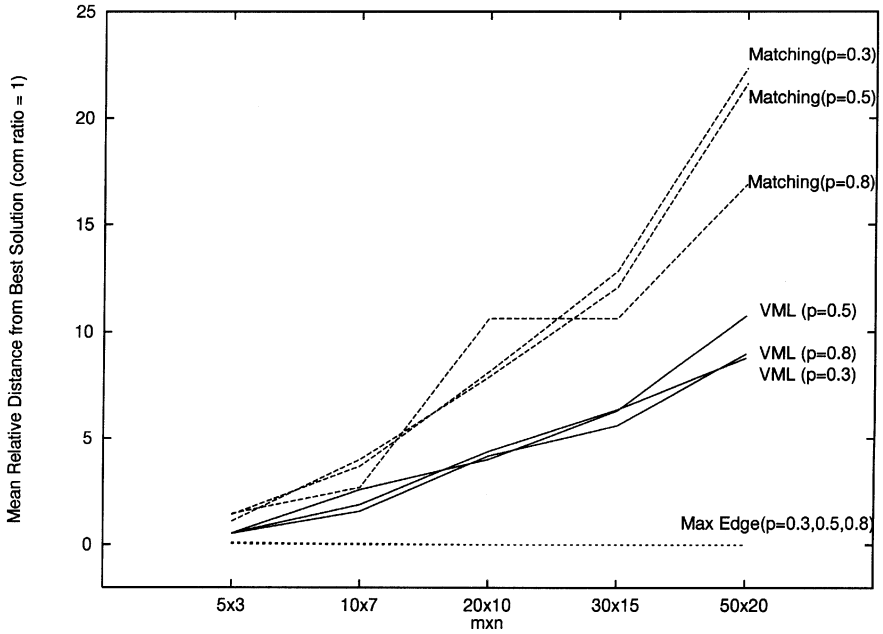

FIG. 3. Mean relative distance for com ratio = 1 and for varying density p.

TABLE II
Mean Relative Distance from Best Solution for $r_{com} = 1$ and for Varying Graph Density p

$m \times n$	p	VML	Matching	MaxEdge
(5,3)	0.3	0.53	1.41	0.11
	0.5	0.54	1.09	0.04
	0.8	0.52	1.45	0.07
(10,7)	0.3	1.87	3.68	0.02
	0.5	2.57	4.00	0.00
	0.8	1.56	2.68	0.05
(20,10)	0.3	4.38	8.10	0.00
	0.5	3.99	7.68	0.00
	0.8	4.16	10.63	0.00
(30,15)	0.3	6.39	12.83	0.00
	0.5	6.29	12.08	0.00
	0.8	5.59	10.62	0.00
(50,20)	0.3	8.75	22.35	0.00
	0.5	10.74	21.64	0.00
	0.8	8.95	16.88	0.00

TABLE III
Mean Relative Distance from Best Solution for Varying r_{com} (Mean over 20 Problem Instances for Each Value of Graph Density p)

r_{com}	$m \times n$	VML	Matching	MaxEdge
= 1	(5,3)	0.53	1.32	0.07
	(10,7)	2.00	3.45	0.02
	(20,10)	4.19	8.90	0.00
	(30,15)	6.09	11.9	0.00
	(50,20)	9.49	20.29	0.00
= 0.2	(5,3)	0.08	0.33	0.11
	(10,7)	0.25	0.46	0.05
	(20,10)	0.73	1.13	0.01
	(30,15)	1.16	1.26	0.01
	(50,20)	1.98	3.70	0.00
= 0.05	(5,3)	0.03	0.21	0.49
	(10,7)	0.08	0.11	0.69
	(20,10)	0.16	0.08	0.28
	(30,15)	0.19	0.11	0.10
	(50,20)	0.50	0.60	0.00
	(80,30)	0.75	1.30	0.00

we use average measures for the three different density values and present them as a single measure.

The percentage of cases where algorithm Max Edge provided the best solution was constantly between 90% and 100% for different values of communication ratio r_{com}. In Fig. 4, the mean relative distance comparison is presented for $r_{com} = 0.2$, as an indication of the stability of Max Edge algorithm (corresponding detailed values in Table III). The relative behavior of the curves is quite similar and Max Edge converges fast to zero, indicating that it provided the best solution among the three methods in most of the cases. However, the absolute values of the relative distances for VML and Matching heuristics is clearly reduced.

In order to investigate the performance of the algorithms, experiments were affected with several different r_{com} values, superior to 1, as well as inferior to 1. It was observed that only in the extreme case where communication cost was much weaker than execution cost ($r_{com} = 0.05$), relative performance

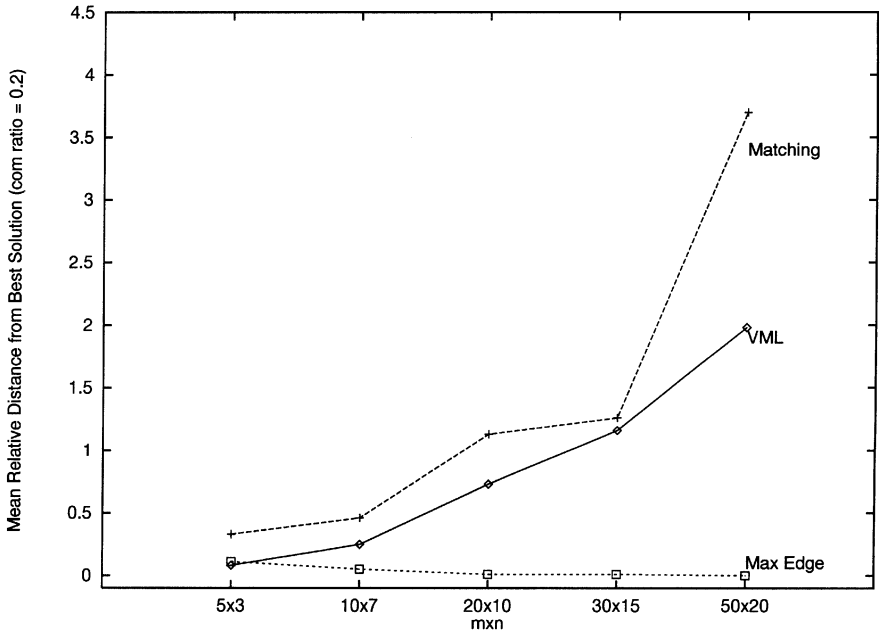

FIG. 4. Mean relative distance com ratio = 0.2 (mean over different densities).

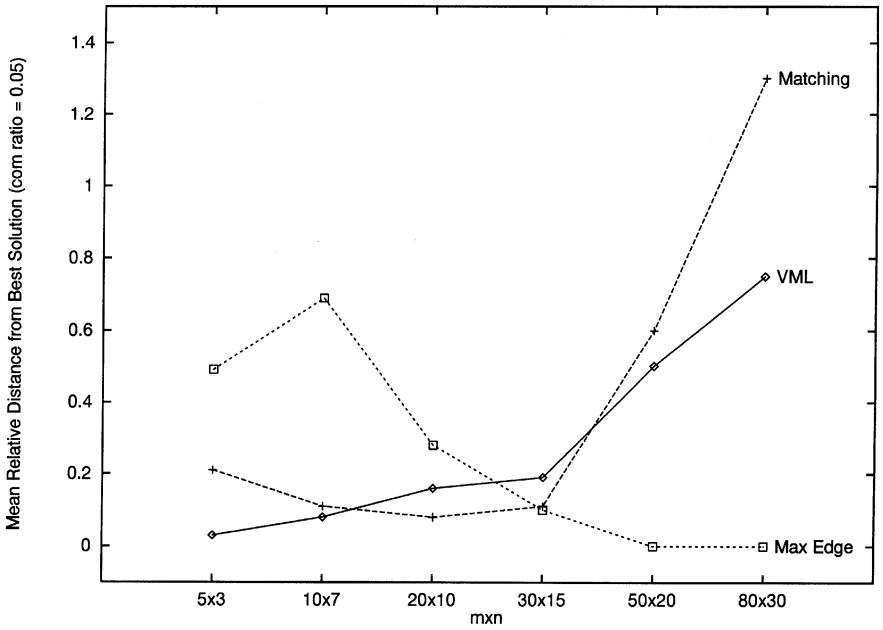

FIG. 5. Mean relative distance com ratio 0.05. Critical point is 30×15.

changed for small size instances. In Fig. 5 (corresponding detailed values in Table III) it is clear that Matching and VML heuristics provide better solutions for $m \times n = 5 \times 3$, 10×7, and 20,10. At the point $m \times n = 30 \times 15$, the methods seem almost equivalent and for $m \times n = 50 \times 20$ and 80×30 the superiority of Max Edge is obvious and its relative distance from best solution converges to zero. In order to confirm the above observation for $r_{\text{com}} = 0.05$, we fixed the number of processors to 10 and we varied the number of tasks. Results are presented in Fig. 6.

Matching and VML heuristics outperform Max Edge for small sizes (10×10, 15×10, 20×10). The point of

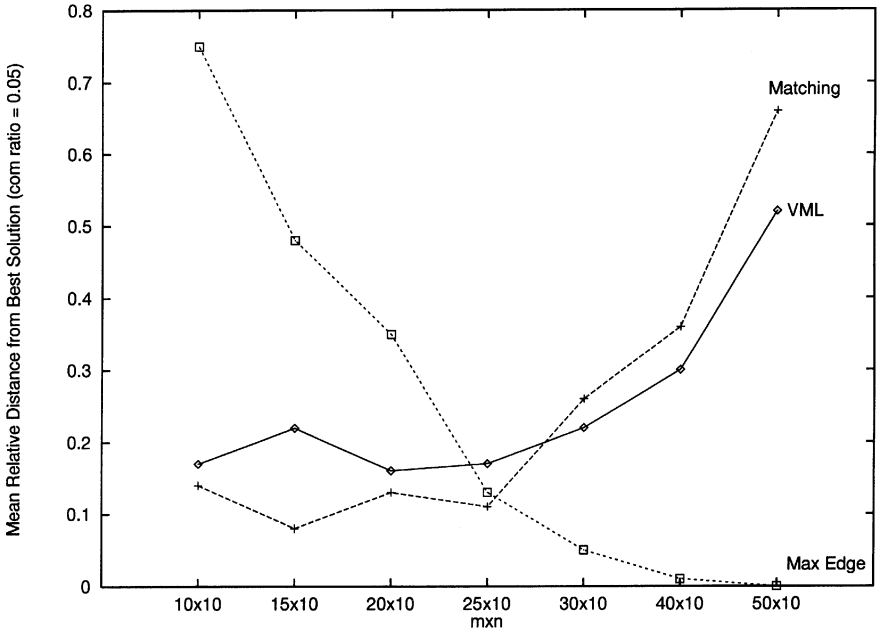

FIG. 6. Mean relative distance for com ratio 0.05 and for 10 processors.

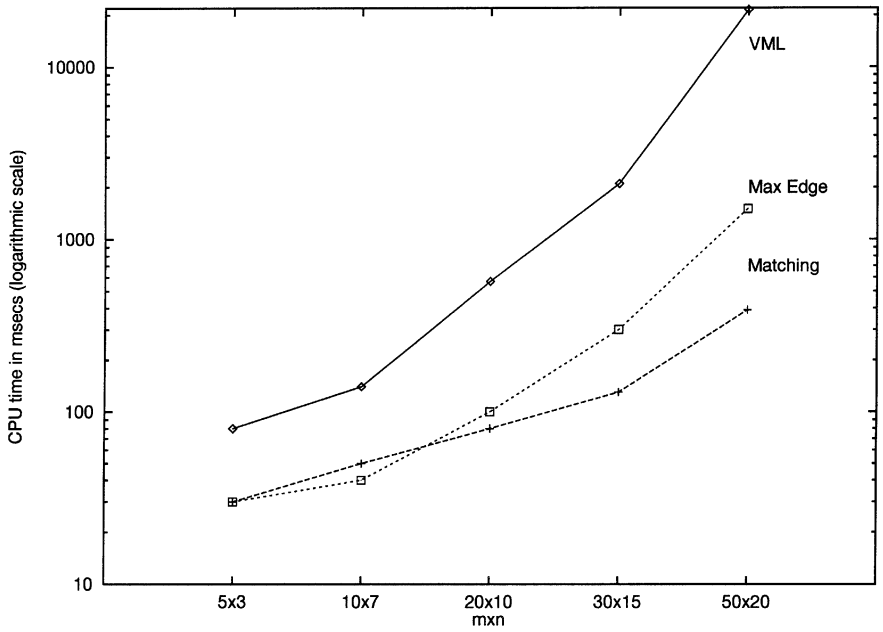

FIG. 7. CPU times in msecs for the three algorithms (in logarithmic scale).

equivalence of three methods is 25 × 10 and for large sizes (30 × 10, 40 × 10, 50 × 10) the relative distance of Max Edge converges again to zero.

The above behavior can be explained by the fact that the impact of communication costs in the objective function is negligible when $r_{\text{com}} = 0.05$ and the number of tasks is small. When the number of tasks is large enough to introduce important total communication costs (even though each communication cost is small), algorithm Max Edge regains in performance over Matching and VML algorithms.

In conclusion, experimental results have proved the effectiveness of Max Edge heuristic, especially when communications are important. For very small communication times and few tasks, Matching and VML heuristics provide better allocations. On the contrary, they fail in determining low cost allocations when there is an important tradeoff between execution and communication costs.

Finally, it is of great importance the fact that CPU execution time for both algorithms proposed in the present study is impressively small. Measures on CPU time for the three algorithms considered are presented in Fig. 7 in logarithmic scale. Matching and Max Edge algorithms are extremely fast even for large size instances (0.4 and 1.5 s respectively for 50 tasks on 20 processors compared to 21 s for VML). The Matching based heuristic is clearly faster than Max Edge for large size instances due to multiple clustering and allocation decisions at each step of the algorithm. In fact, this is the reason we have used a fast heuristic procedure for determining maximum weight matching, instead of the optimal algorithm (the complexity of the Matching based heuristic was calculated in Section 4). Even though there was a negligible improvement in solution quality when we used the optimal algorithm, the CPU time of the Matching based heuristic was significantly increased. Therefore, we preferred keeping the version of the algorithm with near optimal matchings as an alternative allocation strategy, clearly less efficient than the others, but faster for large problems.

7. CONCLUSION

We have presented two fast algorithms for task allocation, both based in a transformation of the initial minimization problem to a maximization one. In the present study, an experimental evaluation was affected, proving their efficiency.

Many questions arise from this work. Further performance analysis would possibly provide explanations on algorithm behavior. For example, it would be interesting to know if the break region for the three methods considered is due to algorithm function or to problem structural properties.

Another interesting perspective is the theoretical study of the heuristics proposed. The simplicity of problem transformation and of the structure of new algorithms can be exploited in order to provide approximation guaranties. Recent evolution on the approximability of the problem [9, 10] shows that the way to proceed is the analysis of special cases where execution or communication costs belong to a finite set of distinct values. In any case, the Max Edge algorithm avoids large penalties in the objective function due to communication and inefficient assignments at each step and this characteristic could be exploited in its worst-case performance analysis.

REFERENCES

1. Billionnet, A., Costa, M. C., Sutter, A. An efficient algorithm for task allocation problem. *J. Assoc. Comput. Mach.* **39,** 3 (1992), 502–518.
2. Bokhari, S. H. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Trans. Software Engrg.* **SE-7,** 6 (Nov. 1981).
3. Dahlhaus, E., Johnson, D. S., Papadimitriou, C. H., Seymour, P. D., and Yannakakis, M. The complexity of multiway cuts. *Proc. 24th ACM STOC.* 1992.
4. Efe, K. Heuristic models of task assignment scheduling in distributed systems. *IEEE Comput.* (June 1982).
5. Feo, T., Goldschmidt, O., and Khellaf, M. One-half approximation algorithms for the k-partition problem. *Oper. Res. Soc. Amer.* **40,** Suppl. 1 (1992), 170–173.
6. Fernandez-Baca, D. Allocating modules to processors in a distributed system. *IEEE Trans. Software Engrg.* **15,** 11 (Nov. 1989).
7. Gallo, G., and Simeone, B. Optimal grouping of researchers into departments. *Ric. Oper.,* 57 (1991), 45–69.
8. Goldschmidt, O., and Hochbaum, D. A polynomial algorithm for the k-cut problem for fixed k. *Math. Oper. Res.* **19,** 1 (1994), 24–37.
9. Lamari, M., and Fernandez de la Wega, W. The task allocation problem with complete communication. Research Report LRI, No. 1048, 1996.
10. Lamari, M., and Fernandez de la Wega, W. The module allocation problem: An average case analysis. *IRREGULAR'96* (G. Gaos, J. Hartmanis, and J. van Leeuwen, Eds.), Lecture Notes in Computer Science, Vol. 1117. Springer-Verlag, Berlin/New York, 1996.
11. Lo, V. M. Heuristic algorithms for task assignment in distributed systems. *IEEE Trans. Comput.* **C-37** (Nov. 1988), 1384–1397.
12. Lo, V. M. Algorithms for static task assignment and symmetric contraction in distributed computing systems. *Int. Conf. Parallel Processing.* 1988, pp. 239–244.
13. Magirou, V. F., and Milis, J. An algorithm for the multiprocessor assignment problem. *Oper. Res. Lett.* 8 (1989), 351–356.
14. Magirou, V. F. An improved partial solution to the task assignment problem. *Oper. Res. Lett.* **12** (1989).
15. Mainiezzo, V., Dorigo, M., and Colorni, A. Algodesk: An experimental comparison of eight evolutionary heuristics applied to the quadratic assignment problem. *Eur. J. Oper. Res.* **81** (1995), 188–204.
16. Milis, I. Task assignment in distributed systems using network flow methods. *CCS'95,* (M. Deza *et al.,* Eds.). LNCS 1120, pp. 396–405, 1996.
17. Price, C. C., and Krishnaprasad, S. Software allocation models for distributed computing systems. *Proc. 4th Int. Conf. on Distributed Computing Systems.* 1987.
18. Stone, Harold S. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Engrg.* **SE-3,** 1 (Jan. 1977).
19. Stone, Harold S. Critical load factors in two-processor distributed systems. *IEEE Trans. Software Engrg.* **SE-4,** 3 (May 1978).
20. Towsley, D. F. Allocating programs containing branches and loops within a processor system. *IEEE Trans. Software Engrg.* **SE-12** (1987).

YANNIS KOPIDAKIS received a M.Sc. in computer science from the University of Crete, in 1993. He is currently a Ph.D. student at the University of Paris—Sud. His research interests include approximation algorithms and scheduling in parallel and distributed systems.

MERIEM LAMARI received a Diplôme d'Etudes Approfondies from the University of Paris—Dauphine, in 1992. She is currently a Ph.D. student at the University of Paris—Sud. She is working on neural networks and tasks allocation in distributed systems.

VASSILIS ZISSIMOPOULOS is a graduate from Athens University, Greece, and in 1984, received his Ph.D. in computer science at the University of Paris—Sud. He is currently a Maître de Conférences of computer science at the University of Paris—Sud. His research interests include approximation algorithms, local search, neural networks in combinatorial optimization, task allocation, static and dynamic scheduling, load balancing, and adaptive algorithms.

Received March 20, 1996; revised January 8, 1997; accepted February 12, 1997