Exact algorithms for a task assignment problem

Kamer Kaya, Bora Uçar

Parallel Processing Letters

EXACT ALGORITHMS FOR A TASK ASSIGNMENT PROBLEM

KAMER KAYA*
Department of Computer Engineering, Bilkent University, 06800, Ankara, Turkey.

and

BORA UÇAR†

Centre National de la Recherche Scientifique,
Laboratoire de l'Informatique du Parallélisme,
(UMR CNRS-ENS Lyon-INRIA-UCBL),
Université de Lyon,
46, Allée d'Italie, ENS-Lyon, F-69364, Lyon France.

Received December 2008
Revised March 2009
Communicated by J. Dongarra and B. Tourancheau

ABSTRACT

We consider the following task assignment problem. Communicating tasks are to be assigned to heterogeneous processors interconnected with a heterogeneous network. The objective is to minimize the total sum of the execution and communication costs. The problem is NP-hard. We present an exact algorithm based on the well-known A* search. We report simulation results over a wide range of parameters where the largest solved instance contains about three hundred tasks to be assigned to eight processors.

Keywords: task assignment; heterogeneous computing systems; task interaction graph; A* search

1. Introduction

Given a model of tasks and a model of computing environment, the task assignment problem asks for a proper assignment of tasks to available processors in order to optimize some performance metric. In our target problem, the tasks are modeled using a task interaction graph (TIG). In this model, the vertices of the graph correspond to the tasks and the edges correspond to the intertask communications. There is no

*Supported by the Turkish Scientific and Technological Research Agency (TÜBİTAK) Ph.D. scholarship.
†The work of this author is partially supported by "Agence Nationale de la Recherche", through the SOLSTICE project ANR-06-CIS6-010.

precedence relation among the tasks—the edges represent the communication that takes place at any time or intermittently throughout the execution of the respective tasks. We assume a heterogeneous computing system in which the execution cost of a task depends on which processor it is executed. We further assume that the network is heterogeneous, i.e., the cost of communication between two interacting tasks depends on which processors they are mapped. The objective is to minimize the sum of the total execution and communication costs. The problem is known to be NP-hard [4]. We propose an exact algorithm based on A* search. Exact algorithms can be useful in there different contexts. First, when the optimal utilization of resources is utmost importance, such an approach would be indispensable, of course if the solutions are delivered in an acceptable time frame. Second, in case a set of tasks is run multiple times, the optimal mapping would be reused, and hence the time to compute such a solution can be amortized. Third, probably of more general adoption, the exact solutions can be used to evaluate the heuristic approaches designed for the same problem.

The TIG model was first introduced by Stone [13]. In this original work, the model is used to represent sequentially executing, persistent tasks. In this setting, at any time exactly one task is being executed on one of the processors. The edges of the model represent two-way interactions between the tasks where a task passes control to another one and waits the control to be returned back again. The same interpretation is used, for example, in mapping parallel pipelines [17] and phased messages-passing programs [8]. For more on the use of TIG model on task assignment in distributed computing systems, we refer the reader to two recent papers [2, 16] and the references therein.

Formally, the task assignment problem we consider is as follows. Let $\mathcal{P}$ be the set of P processors in the heterogeneous computing system, $\mathcal{T}$ be the set of T tasks to be assigned to the processors, $ETC = \{x_{ip}\}_{T\times P}$ be the expected time to compute matrix where x_{ip} denotes the execution cost of task i on processor p, and $\mathcal{G} = (\mathcal{T}, \mathcal{E})$ be the task interaction graph. The processors are heterogeneous in the sense that there is no special structure on the ETC matrix. In other words, processor p being faster than processor q on task i, e.g., $x_{ip} \leq x_{iq}$, does not imply anything about their speeds for another task. Each edge $(i, j) \in \mathcal{E}$ is associated with a weight c_{ij} representing the amount of data transfers needed between tasks i and j. The network is heterogeneous in the sense that the links between different pairs of processors are not uniform. A "distance" d_{pq} is associated with the link of processors p and q, e.g., the inverse of the bandwidth between p and q, such that if tasks i and j are mapped to the processors p and q, then a communication cost of $c_{ij}d_{pq}$ is incurred. The distance metric is symmetric, i.e., $d_{pq} = d_{qp}$. Furthermore, we assume that $d_{pp} = 0$ for any processor p. Hence, no communication cost is incurred if two interacting tasks are assigned to the same processor. The objective is to find an assignment $A : \mathcal{T} \to \mathcal{P}$ that minimizes the sum of execution and communication costs. The formulation is as follows:

$$Minimize \left(\sum_{i=1}^{T} \sum_{p=1}^{P} a_{ip} x_{ip} + \sum_{(i,j) \in E} \sum_{p=1}^{P} \sum_{q=1}^{P} a_{ip} a_{jq} c_{ij} d_{pq} \right) \ subject\ to$$

$$\sum_{p=1}^{P} a_{ip} = 1, \quad i \in \mathcal{T}$$

$$a_{ip} \in \{0,1\}, \quad p \in \mathcal{P}; \ \ i \in \mathcal{T} \ .$$

The variables are a_{ip}, where if task i is assigned to processor p, then $a_{ip} = 1$, otherwise $a_{ip} = 0$. The constraint $\sum_{p=1}^{P} a_{ip} = 1$ ensures that task i is assigned to exactly one processor. The first part of the objective function corresponds to the total cost of task executions, and the second part corresponds to the total communication cost. Although the problem is NP-hard [4], some special instances are polynomial time solvable. We note especially the instances whose TIGs are in tree structure. Those instances are solvable in $O(TP^2)$ time [4] in heterogeneous networks. If, furthermore, the network is homogeneous, then an $O(TP)$ time solution exists [3].

The A* search algorithm is a best-first graph search algorithm. Starting from a given initial node, it finds the least cost path to a goal node. A heuristic function lies at the core of an A* search-based algorithm. This heuristic function is used to estimate the minimum cost from a given node to a goal node, and it effects the way along which the search proceeds.

In [9], an A* search-based algorithm is proposed for mapping TIGs to heterogeneous processors interconnected with a homogeneous network where the objective is to minimize the turnaround time, i.e., the maximum load of a processor in terms of the total execution and communication costs. In [15], two A* search-based algorithms are proposed, one for mapping TIGs to heterogeneous processors in a heterogeneous network, and another one for mapping communicating tasks with precedence constraints again with heterogeneous processors and network. The objective function is to minimize the turnaround time. In [14], A* search-based algorithm is proposed for the same problem considered in this paper, i.e., a TIG is to be mapped to heterogeneous processors interconnected with a heterogeneous network in order to minimize the total execution and communication cost. In [10], algorithms based on A* search are proposed for optimizing the execution of communicating tasks with precedence constraints on homogeneous processors. These four works explore the search space of the task assignments following a tree structure. The search space tree has $T + 1$ levels. In the first level there is only one node which corresponds to an empty assignment. That node is the initial node for the search. At level ℓ of the the tree there are P^ℓ nodes, where all the assignment for the first ℓ nodes are represented. The common heuristic function in these three works is based on computing the objective function without taking the communication costs between the yet to be assigned tasks into account, i.e., at a level ℓ node, the tasks $\ell + 1$ to T

are pared from their communication requirements, and the task assignment problem confined to those communication-free tasks are solved to obtain an estimate of the cost. In [15], it has been observed that if the tasks yet to be assigned form an independent set, i.e., a set of tasks among which there is no communication, then assigning those tasks after the others will reduce the search space drastically. This was possible, because the task assignment problem for a set of independent tasks is easily solvable. In other words, the heuristic function becomes an exact algorithm, if the tasks on which it is called form an independent set. Another study that aims to obtain exact solutions to some three task assignment problems is given in [11]. In this work, one of three problems corresponds to the problem formally defined above, but in a homogeneous network. The author proposes different formulations and uses a state-of-the-art commercial integer linear program solver to evaluate the effectiveness of the proposed formulations.

In this work, we propose the use of the exact algorithm for tree structured TIGs as the heuristic function to be used in the A* search. As it is an essential part of our contribution, we summarize that algorithm in Section 2.1. We then give a brief summary of the A* search in Section 2.2. In Section 3, we discuss the use of Bokhari's algorithm in the context of A* search, and elaborate on how we take advantage of that algorithm to develop the proposed exact algorithm. Section 4 contains a summary of a large set of experiments among which solutions to a problem instance with 307 tasks and 8 processors (yielding a search space of 8^{307} nodes) are reported. Section 5 concludes the paper with a brief summary.

2. Background

2.1. *Bokhari's shortest tree algorithm*

Here, we reproduce Bokhari's exact algorithm [4] for tree structured TIGs. In doing so, our aim is two-folds: completeness, and a clearer exposition of the algorithm described in [4].

Bokhari's algorithm uses a dynamic programming formulation. Consider a TIG in the form of a tree. Let $A(i,p)$ denote the cost of the optimal solution of the subtree rooted at the node i under the condition that the task associated with the node i is assigned to processor p. Now consider an internal node i such that for each child j of i, we have computed $A(j,k)$ for $k = 1,\ldots,P$. Then

$$A(i,p) = x_{ip} + \sum_{j \in \text{child}(i)} \min_k \{A(j,k) + c_{ij} d_{pk}\} \ .$$

That is, for each child j of i, we consider P optimal solutions $A(j,k)$ and their extensions by assigning the task i to the processor p. Clearly for a leaf node ℓ, we have $A(\ell,p) = x_{\ell p}$. Processing the nodes of tree in a topological order, that is in an order where all the children nodes are processed before their father, will yield $A(r,p)$ for the root node r and for each processor $p = 1,\ldots,P$. The value $\min_p A(r,p)$ is thus the value of an optimal solution.

Consider the computation of an $A(i,p)$. As seen from the above formula, it requires $O(1 + P|\text{child}(i)|)$ time. Therefore, for a node i, the time complexity of operations is $O(P + P^2|\text{child}(i)|)$. The sum of these quantities over all tasks thus gives a time complexity of $O(TP^2)$, as the sum of the number of children is $O(T)$.

2.2. *A* Search*

A* search is a best-first, graph search algorithm mostly used in artificial intelligence literature [12]. Its aim is to find a least cost path from a given initial node to a goal node. It starts from the initial node and expands it, that is, generates all of its successors—all its neighboring nodes. Each such node v is evaluated with a function

$$f(v) = g(v) + h(v) \ , \tag{1}$$

where $g(v)$ is the actual cost to reach the node v from the initial node, and $h(v)$ is a heuristic function that estimates the cost of the cheapest path from v to a goal node. In other words, the evaluation function is an estimate of the cheapest cost of a path that passes through v. After evaluating each generated node, those nodes are placed in a list of active nodes. Among the active nodes, the one with the minimum f-value is selected to be expanded next, whereupon it becomes inactive. The search terminates with the least cost path when an active node with the minimum f-value is a goal node. The most important component of the A* search is the heuristic function h. The search is guaranteed to terminate by finding an optimal goal node if h satisfies certain conditions. When run on tree's, the necessary condition on h for the search to be as such is that h should be an admissible heuristic, i.e., $h(v)$ should never overestimate the cost of the path from v to a goal node. Other conditions are required for general graphs [12, p.99]. Two notable characteristics of the A* search are in order. First, it is optimally efficient for any heuristic function h in the sense that no other algorithm, for example branch-and-bound, employing h can expand fewer nodes than A*. Second, A* never expands a node whose f-value is larger than an optimal solution.

In the task assignment problem, the graph is the search space of assignments which can be perceived as a tree. The initial node is the empty assignment. The nodes are partial assignments. In particular, all nodes at a distance ℓ from the initial node, pertain to assigning task t_ℓ to a particular processor given a particular assignment of all tasks t_1 to $t_{\ell-1}$. The goal nodes are those at distance T from the initial node, i.e., the complete assignments. Each active node v of the tree thus contains a node identifier and the values $g(v)$ and $h(v)$. The assignments of tasks t_1 through $t_{\ell-1}$ that yielded the node v can also be stored in order to be able to return an optimal task-to-processor assignment. With these definitions, the A* search for the task assignment problem assumes the following form (somehow simplified).

1: Let $\mathcal{L}$ be the list of active nodes, initially containing P assignments $t_1 \to p$ for $p = 1, \ldots, P$, keyed with the g+h values of the assignments
2: **while** $\mathcal{L} \neq \emptyset$ **do**
3: $\langle t_\ell, q\rangle \leftarrow$extractMin($\mathcal{L}$)
4: **if** $t_\ell = t_T$ **then**
5: return
6: **for** $p = 1, \ldots, P$ **do**
7: compute g and h for the assignment $t_{\ell+1} \to p$
8: add $\langle t_{\ell+1}, q\rangle$ to $\mathcal{L}$ with key g+h

The design of an efficient and effective heuristic function h for the task assignment problem and making best use of that heuristic are our contributions to be explained in the following section.

3. An exact algorithm

In order to be able to expose the designed algorithm, we resume the discussion of A*. Recall that we explore the search space of the task assignment problem which can be represented as a tree. The initial node is empty assignment, and an intermediate node at level ℓ of the search space tree represents the assignment of tasks t_1 to t_ℓ. For example, at level 1, we have P nodes, each one corresponding to the assignment of task t_1 to a particular processor. Each of these P nodes has P children, each corresponding to the assignment of task t_2 to a certain processor. As it is implicitly mentioned, there is an order of task assignments—which we will return to in the next subsection. For now, assume that we are given an order $t_1, \ldots, t_T$ of tasks. We represent a node of the search space with a triplet $v = \langle \ell, p, \pi(v)\rangle$, where ℓ is the level of the node v, hence v is associated with the task t_ℓ; the second component p is the processor number to which t_ℓ is assigned; $\pi(v)$ is the father of the node v in the search space tree. Two values g and h are associated with each node. The value g is the actual cost of the assignments of tasks t_1 to t_ℓ to the processors that led to the node v, and the value h is the heuristic estimate of the cost of the best assignment of the tasks $t_{\ell+1}$ to t_T.

As discussed before, A* search keeps a list of active nodes and chooses the node with the least evaluation function f to expand next. The children of that node are generated, and those nodes are added to the list of active nodes after computing the g and h values for each. Suppose the node $v = \langle \ell - 1, p, \pi(v)\rangle$ is an active node with the least f-value. Then the nodes

$$n_q = \langle \ell, q, v\rangle \ , \text{ for } 1 \leq q \leq P \ ,$$

are generated. Each of these nodes corresponds to appending the assignment of the task t_ℓ to a processor, q, to the partial assignment represented by v. Clearly,

$$g(n_q) = g(v) + x_{t_\ell,q} + \sum_{\substack{u=1,\\ (u,t_\ell)\in\mathcal{E}}}^{\ell-1} \sum_{p=1}^{P} a_{up} c_{t_\ell,u} d_{pq} \ . \tag{2}$$

In other words, the actual cost of a child node $\langle \ell, q, v \rangle$ is computed from its parent cost by adding the execution cost $x_{t_\ell,q}$ and the communication cost with the already assigned tasks (those tasks t_i where $i < \ell$).

We next describe the heuristic function h that we have designed. In short, we create a task assignment problem with a tree structure defined on the tasks yet to be assigned. We then solve this task assignment problem using Bokhari's shortest tree algorithm [4] (see summary in the previous section), and use the cost of the resulting assignment as the h-value. Given a node $v = \langle \ell, p, \pi(v) \rangle$ of the search space, we take a spanning forest $\mathcal{F}_\ell = (\mathcal{T}_\ell, \mathcal{E}_\ell)$ of the nodes corresponding to the tasks $t_{\ell+1}, \ldots, t_T$. That is, $\mathcal{T}_\ell = \mathcal{T} \setminus \{t_1, \ldots, t_\ell\}$ and $\mathcal{E}_\ell \subseteq \mathcal{E} \cap \mathcal{T}_\ell \times \mathcal{T}_\ell$. Note that any node of the search space at level ℓ from the root is associated with the same graph, and therefore $\mathcal{F}_\ell = (\mathcal{T}_\ell, \mathcal{E}_\ell)$ is a common spanning forest to all those nodes. Any edge $(i, j) \in \mathcal{E}_\ell$ inherits the same communication amount c_{t_i,t_j} from the original problem. We could have done the same for the execution costs of the tasks in $\mathcal{T}_\ell$, but one can do better by utilizing the partial assignment associated with node v. Consider an edge $(t_i, t_k) \in \mathcal{E}$ where $i < \ell \leq k$. As the task t_i has been assigned to a processor, say q, a cost of $x_{t_k,p} + c_{t_i,t_k} d_{pq}$ will be incurred if the task t_k is executed on processor p. Therefore, we define the expected time to compute matrix $ETC_\ell = \{x'_{t_k,p}\}_{(T-\ell)\times P}$ as follows

$$x'_{t_k,p} = x_{t_k,p} + \sum_{\substack{i<\ell, \\ (t_i,t_k)\in\mathcal{E}}} a_{t_i,q} c_{t_i,t_k} d_{pq} \quad \text{for } k \geq \ell \text{ and } p = 1, \ldots, P\,. \tag{3}$$

Note that $a_{t_i,q}$ is defined according to the partial assignment associated with node v of the search space, and hence the subproblems associated with nodes at level ℓ from the root node of the search space can have different ETC matrices.

As the spanning forest is a part of the original problem, using the exact solution for subproblem associated with a node forms an admissible heuristic function. The proposed heuristic dominates the heuristic used in [9, 10, 15] in the sense that the assignment cost found by the proposed heuristic is never less than those that can be found by that alternative (the latter one ignores the communication of the remaining tasks). There can be many spanning forests associated with the tasks in $\mathcal{T}_\ell$, and the heuristic function would be admissible when run on any of those forests. As we would like to include as much cost as possible into the subproblem associated with the node v, we choose the maximum edge weighted spanning forest which can be found using Kruskal's or Prim's algorithms [6, Chapter 24] in $O(|\mathcal{E}_\ell| \log_2 |\mathcal{T}_\ell|)$ time. We compute the maximum edge weighted spanning forest $\mathcal{F}_\ell$ for each task t_ℓ before commencing on the search procedure. During A*, at a node v containing task t_ℓ, we use the spanning forest $\mathcal{F}_\ell$.

3.1. *Order of the task assignments*

As the proposed heuristic function becomes exact for a tree structured subproblem, we can stop the search procedure as soon as the subgraph $\mathcal{G}_\ell = (\mathcal{T}_\ell, E \cap \mathcal{T}_\ell \times \mathcal{T}_\ell)$ of

the active node with the minimum f-value is a tree. In other words, we can stop when $\mathcal{G}_\ell$ does not contain a cycle, or, when $\mathcal{T} \setminus \mathcal{T}_\ell$ contains at least one vertex from every cycle of $\mathcal{G}$. In such a case, the search space that needs to be explored becomes of size $P^{T-|\mathcal{T}_\ell|}$. To minimize the size of the search space we would like to have the largest subgraph without cycles (so as to maximize $|\mathcal{T}_\ell|$), or have the smallest set of vertices that contains at least one vertex from every cycle (so as to minimize $T - |\mathcal{T}_\ell|$). However, finding such a set is known to be NP-hard (see [7], problem GT7, the minimum feedback vertex set).

As the problem of minimizing the size of the search space is NP-hard, we resort to some heuristics. We proceed in two stages. First, we find an independent set in $\mathcal{G} = (\mathcal{T}, \mathcal{E})$, and then augment the independent set with vertices as long as the vertex set thus grown is acyclic.

In order to find an independent set, we sort the vertices of the graph $\mathcal{G} = (\mathcal{T}, \mathcal{E})$ in the increasing order of degrees and then visit the vertices in that order. Each vertex is added to the set $\mathcal{S}$, initially an empty set, if it has no neighbor in $\mathcal{S}$. At the end of this step $\mathcal{S}$ is a maximal independent set, i.e., no other vertex can be added to $\mathcal{S}$ without violating this property. Then, we visit the vertices in $\mathcal{T} \setminus \mathcal{S}$ one more time, in the same order and add each visited vertex to $\mathcal{S}$, if its neighbors in $\mathcal{S}$ are not connected through paths. For this purpose, we use disjoint set operations [6, Chapter 22]. We start the second stage by making each vertex of the independent set found at the end of first stage a set. During visiting a vertex t_i in the second stage, we test whether its neighbors in $\mathcal{S}$ are in disjoint sets. If that is the case, those sets are unified by set union operation, including the vertex t_i, and t_i becomes a member of $\mathcal{S}$. If not, t_i is skipped. We note also that $\mathcal{S}$ is a maximal acyclic set.

We now define the order of task assignments, i.e., define the task t_ℓ to be assigned at a node at the level ℓ from the initial node of the search space. We build a tentative assignment of tasks from scratch, where the order in which task assignments took place defines the task orders to be used during the A* search. We define the order of task assignments only for the tasks in $\mathcal{T} \setminus \mathcal{S}$, as the assignments for those in $\mathcal{S}$ would not be represented in the search space—the A* search terminates when a node at level $|\mathcal{T} \setminus \mathcal{S}| + 1$ is reached. Note that this ordering of tasks is done only once before starting the search procedure.

For a task t_ℓ, we compute the g- and h-values as we have done in the A* search for each possible assignment of t_ℓ, i.e., we compute f-values for the assignments $t_\ell \rightarrow p$, for each p. Then, we add up these f-values and obtain a single value for the task t_ℓ. This procedure is performed for each task yet to be assigned, and the task with the minimum sum is tentatively assigned to the processor p where the assignment $t_\ell \rightarrow p$ yielded the minimum f-value among the P alternatives. Then, the process is repeated with a reduced number of tasks yet to be assigned. The total cost of this preprocessing step is thus $O\left(\sum_{r=1}^{T-1} r^2 P^2\right) = O(P^2 T^3)$.

3.2. *Other details*

In order to find the active node with the minimum f-value, we use a priority queue implemented as a binary heap. We maintain the heap in such a way that among the nodes with the same f-value, the one which is at he highest level has the highest priority. This way, we aim to go deeper in the search tree whenever possible.

We also utilize task assignment heuristics from the literature to find upper bounds on the optimal assignment cost. For this purpose, we use variants of the algorithms given in [16] for homogeneous networks and that of the sufferage heuristic given in [5] for independent tasks to compute upper bounds. The variants keep the main ideas of the original works while adapting them for communicating tasks in heterogeneous networks. Before starting the A* search, we run each of these variants once and get the lowest of the upper bounds to be used during the search procedure. For a generated node, we check if its f-value is larger than the upper bound (at line 8 of the generic algorithm of Section2.2). If so, the node is not added to the list of active nodes. We note that this pruning operation reduces the run-time of the algorithm only due to heap operations, and saves memory only for the pruned nodes themselves, as the A* search never expands a node with an f-value larger than the optimal cost.

4. Experiments

We tested the performance of the proposed exact algorithm in comparison with some other heuristic. All the algorithms were implemented in C language on a Linux platform. All experiments were performed on a PC equipped with a Dual 250 Opteron AMD processors and 4GB of RAM.

We constructed the TIGs from five small sparse matrices, members of the DWT matrix set (available at `http://math.nist.gov/MatrixMarket`). We conducted our experiments for $T \in \{59, 72, 87, 209, 307\}$ and $P \in \{2, 3, 4, 8\}$. Table 1 gives the number of edges, the size of a maximal independent set (IS), and the size of a maximal acyclic component (AS) for the resulting task assignment problem instances.

Table 1: Some properties of the problems.

	$\mathcal{T}$				
	59	72	87	209	307
$\lvert\mathcal{E}\rvert$	104	75	227	767	1,108
IS	22	33	18	53	79
AS	38	63	38	103	147

The task and processor heterogeneity were modeled by incorporating different execution times for each task on different processors. The estimated execution-time values of the tasks were stored in a $T \times P$ expected time to compute (ETC) matrix. The ETC matrix can be consistent or inconsistent in terms of the relation between execution times of different tasks [1]. In a consistent ETC matrix, if a processor executes a task faster than another processor, then it executes all other tasks faster

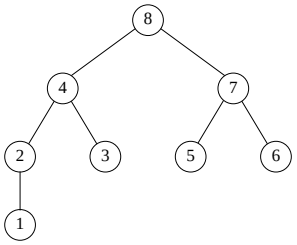
than that processor. If there is no such relation between execution times, then the ETC matrix is said to be inconsistent. We believe that an inconsistent ETC matrix is a better model for the task assignment problem since today's computing environments contains very heterogeneous computing resources with different task execution characteristics [1].

We used the methods in [1] to obtain four different ETC matrices to better evaluate the performance of the proposed task assignment algorithm. These ETC matrices differ in terms of the task and processor heterogeneity where the variations along a column and a row of an ETC matrix are referred to as the task and processor heterogeneity, respectively. The ETC matrix types are given in Table 2.

Table 2: ETC matrix types with different task and processor heterogeneity.

ETC	task heterogeneity	processor heterogeneity
0	low	low
1	low	high
2	high	low
3	high	high

To model network heterogeneity, we first constructed a processor tree as in Fig 1(a). From the processor tree, we obtained the matrix in Fig. 1(b) for distances. In the experiments, for each $P \in \{2, 3, 4, 8\}$, we use the $P \times P$ principal submatrix as the distance matrix.



(a)

$$\begin{pmatrix} 0 & 1 & 3 & 2 & 5 & 5 & 4 & 3 \\ 1 & 0 & 2 & 1 & 4 & 4 & 3 & 2 \\ 3 & 2 & 0 & 1 & 4 & 4 & 3 & 2 \\ 2 & 1 & 1 & 0 & 3 & 3 & 2 & 1 \\ 5 & 4 & 4 & 3 & 0 & 2 & 1 & 2 \\ 5 & 4 & 4 & 3 & 2 & 0 & 1 & 2 \\ 4 & 3 & 3 & 2 & 1 & 1 & 0 & 1 \\ 3 & 2 & 2 & 1 & 2 & 2 & 1 & 0 \end{pmatrix}$$

(b)

Fig. 1. (a) The processor tree. (b) Associated distance matrix.

Our experiments vary with the communication-to-computation ratio ρ defined as $\rho = \left(\sum_{t,u} c_{tu}\right) \Big/ \left(\left(\sum_{t,p} x_{tp}\right)/P\right)$. We used three different ratios $\rho = \{0.7, 1.0, 1.4\}$ to simulate the computation and communication intensive tasks along with the balanced task sets where the required amount of computation is closer to the required amount of communication.

We created 2 instances for each T, P, ρ, and ETC matrix type quadruplet and reported the average performance of the algorithms in the tables. Table 3 shows the number of examined nodes (i.e., for which g- and h-values are computed) for the proposed exact task assignment algorithm with the proposed heuristic function.

Table 3: The number of opened nodes for the exact task assignment algorithm.

P	ETC	ρ	T				
			59	72	87	209	307
2	0	0.7	18	8	64	109	250
		1.0	18	9	64	161	203
		1.4	25	9	64	171	247
	1	0.7	26	10	84	214	242
		1.0	34	10	64	214	322
		1.4	26	8	84	108	242
	2	0.7	18	10	64	163	458
		1.0	34	9	64	193	179
		1.4	36	8	84	415	1,535
	3	0.7	32	9	251	677,546	117,779,136
		1.0	33	9	220	10,209,628	NA
		1.4	63	9	2,633	12,196,477	NA
3	0	0.7	27	9	44	162	368
		1.0	27	11	65	182	545
		1.4	27	9	44	169	601
	1	0.7	27	11	85	215	243
		1.0	27	11	85	215	243
		1.4	35	9	65	162	323
	2	0.7	28	9	65	162	325
		1.0	27	11	44	178	342
		1.4	35	11	65	168	2539
	3	0.7	85	10	205	13,052,714	19,648,139
		1.0	157	12	2,181	85,338,182	NA
		1.4	119	10	1,093	NA	NA
4	0	0.7	28	10	86	163	331
		1.0	20	12	66	185	3,060
		1.4	28	18	100	1576	6,618
	1	0.7	28	12	66	216	244
		1.0	36	12	86	216	244
		1.4	32	10	66	163	244
	2	0.7	28	12	86	219	689
		1.0	28	11	137	242	889
		1.4	38	13	93	565	258,305
	3	0.7	81	19	1,126	20,167,622	NA
		1.0	262	13	9,571	NA	NA
		1.4	143	14	21,552	NA	NA
8	0	0.7	40	21	90	225	340
		1.0	40	33	90	347	956
		1.4	68	16	120	1,057	26,713
	1	0.7	40	16	90	220	248
		1.0	32	16	90	220	328
		1.4	40	16	90	220	328
	2	0.7	40	19	90	250	1,673
		1.0	40	26	97	357	807
		1.4	52	18	97	103,462	3,647,736
	3	0.7	79	49	3,210	NA	NA
		1.0	547	40	532,934	NA	NA
		1.4	9,953	37	1,457,131	NA	NA

In the experiments, we restricted the maximum number of active nodes to be 50 million. Note that this number is different than the examined nodes, as some become

inactive after expanding its children, and some other might have been pruned (see Section 3.2). In Table 3, the NAs represent the problem instances which were not solved within this limit. As expected, when the number of tasks and/or the number of tasks increase the number of examined (and also active) nodes increases. The number of examined nodes for dwt72 does not obey this rule, however, as we realized after looking its structure, the dwt72 matrix contains a big acyclic component hence the admissible heuristic runs very well for those instances.

Another important observation is that the task assignment algorithm becomes harder when the ETC matrices have both high task and processor heterogeneity, i.e., when the problem instances contain ETC matrices of type 3. Note that all of the problem instances which are not solvable with less than 50 million active nodes contain such an ETC matrix. The effects of other ETC matrix types on the memory requirement are not as significant.

Table 3 also shows that for communication intensive task sets, i.e., for $\rho = 1.4$, finding an optimal assignment is harder than finding one for the computation intensive tasks, i.e., for $\rho = 0.7$.

We tested the effect of network heterogeneity on the performance of the exact algorithm by solving the same problem instances with unit processor distances. The comparison of the results is given in Table 4.

Table 4: The effect of network heterogeneity on the memory requirement.

P	T			
	59	87	209	307
3	35	65	168	2,539
	28	65	1,347	27,538
4	38	93	565	258,305
	36	104	1,998	11,931,202
8	52	97	103,462	3,647,736
	57	165	21,378,979	NA

The values in Table 4 are the number of examined nodes for problem instances with an ETC matrix of type 2 and $\rho = 1.4$. In each cell, the value above represents the number of examined nodes for the heterogeneous network, whereas the value below represents the same number for the homogeneous network. Table 4 shows that when we use the distance matrix given in Fig. 1(b), the task sets are easier to assign compared to the case when the network is homogeneous. It is expected since with heterogeneity, an assignment which assigns two communicating tasks to two distant processors can be pruned earlier in the A* search.

We also investigated the performance of the acyclic component approach with respect to that of the independent set (IS) approach. Essentially, we compare the proposed exact algorithm with the approach proposed in [14] with a slight difference in the algorithm that is used to find an IS. Table 5 shows the results of this comparison. Note that we find an acyclic component starting from an independent set (see Section 3.1), i.e., even if the IS heuristic of [14] is used, the acyclic component will be always larger than the IS found.

Table 5: The memory savings of the acyclic component approach compared to the IS approach.

P	ETC	ρ	T				
			59	72	87	209	307
2	0	0.7	45.5%	89.9%	31.0%	60.6%	35.7%
		1.0	45.5%	90.2%	32.1%	52.3%	84.4%
		1.4	63.4%	94.7%	32.4%	76.2%	94.0%
	1	0.7	46.4%	93.8%	31.1%	31.4%	29.3%
		1.0	46.9%	92.2%	31.0%	31.4%	29.4%
		1.4	46.4%	91.8%	31.1%	31.2%	59.8%
	2	0.7	58.6%	91.7%	31.0%	48.6%	68.8%
		1.0	46.9%	95.4%	31.4%	72.0%	88.9%
		1.4	66.7%	91.9%	48.0%	94.4%	99.4%
	3	0.7	83.6%	97.2%	98.1%	–	–
		1.0	85.9%	92.7%	99.5%	–	NA
		1.4	99.0%	91.8%	99.9%	–	NA
3	0	0.7	45.5%	82.3%	30.2%	36.1%	77.8%
		1.0	45.5%	91.2%	44.6%	94.1%	98.8%
		1.4	61.4%	88.1%	10.0%	98.6%	–
	1	0.7	45.5%	91.6%	30.9%	31.3%	29.3%
		1.0	45.5%	89.2%	30.9%	31.3%	29.3%
		1.4	46.2%	85.9%	30.6%	31.2%	37.2%
	2	0.7	70.7%	87.9%	30.6%	33.0%	89.5%
		1.0	60.9%	92.3%	63.2%	31.5%	99.7%
		1.4	73.5%	89.4%	32.5%	99.3%	99.9%
	3	0.7	98.1%	98.1%	99.6%	–	–
		1.0	99.5%	96.4%	99.7%	–	NA
		1.4	100.0%	91.3%	–	NA	NA
4	0	0.7	44.6%	88.0%	13.7%	32.2%	35.2%
		1.0	42.9%	96.9%	37.2%	94.6%	99.5%
		1.4	52.1%	94.7%	46.2%	99.9%	–
	1	0.7	44.6%	87.9%	30.3%	31.2%	29.2%
		1.0	45.5%	87.1%	30.6%	31.2%	29.2%
		1.4	57.2%	86.1%	30.3%	31.1%	–
	2	0.7	77.2%	94.3%	37.6%	67.4%	99.5%
		1.0	46.7%	96.6%	46.5%	90.9%	99.7%
		1.4	64.9%	93.5%	67.9%	100.0%	NA
	3	0.7	99.3%	98.1%	49.4%	–	NA
		1.0	99.9%	92.7%	99.6%	NA	NA
		1.4	99.9%	95.2%	NA	NA	NA

In Table 5, the values show the memory savings when the acyclic component approach is used in the admissible heuristic instead of the IS approach. These values are computed by dividing, whenever possible, the reduction in the number of examined nodes to the number of examined nodes with the IS approach. The table does not contain the instances with $P = 8$, because with the IS approach quite a number of the instances could not be solved with less than 50 million active nodes. In the table, the cells marked with "–" shows the cases where the IS-based heuristic could not obtain solution, whereas the proposed one did. Those marked with NA marks the cases where both of the approaches could not deliver a solution.

Table 5 shows that the memory savings depend on the structure of the TIG. For example, there is a big acyclic component in dwt72. With this acyclic component,

the proposed approach save more than 85% memory for all of the cases. This is not the case for other TIGs. However, even for these TIGs, significant memory savings are also obtained, especially when the task and processor heterogeneity are high. Note that these instances are harder to solve and the effect of the TIG structure is expected to be less effective on the hardness of the optimal assignment problem.

Table 6 shows the average execution times of the proposed exact task assignment algorithm for all problem instances averaged over ρ. For most of the cases, the algorithm seems to be very efficient and assigns the tasks in less than a second. For comparison with the IS-based approach, we give its results on two instances: with $T = 59$, $P = 8$, $\rho = 1.0$ and an ETC matrix of type 3, the IS-based heuristic obtained the optimal result in about 6,988 seconds, whereas the proposed approach obtained the same result in less than 2 seconds; with $T = 307$, $P = 3$, $\rho = 1.4$ and an ETC matrix of type 2, the timings were 4,863 versus 4.5 seconds.

Table 6: The execution times of the proposed algorithm in seconds.

P	ETC	T				
		59	72	87	209	307
2	0	0.44	0.25	0.38	0.48	0.3
	1	0.25	0.32	0.20	0.42	0.29
	2	0.32	0.31	0.38	0.42	0.34
	3	0.32	0.25	0.34	318.20	8,871.86
3	0	0.45	0.32	0.45	0.27	0.38
	1	0.26	0.37	0.45	0.26	0.34
	2	0.38	0.25	0.25	0.27	0.49
	3	0.26	0.31	0.29	4,528.07	2,873.64
4	0	0.32	0.32	0.26	0.36	1.02
	1	0.45	0.25	0.26	0.29	0.38
	2	0.44	0.38	0.32	0.32	17.43
	3	0.26	0.19	0.93	2,682.29	NA
8	0	0.26	0.32	0.35	0.68	9.62
	1	0.33	0.19	0.35	0.45	0.74
	2	0.33	0.38	0.48	22.49	1,139.59
	3	0.59	0.32	57.77	NA	NA

5. Conclusion

We have presented an exact algorithm for assigning communicating tasks into heterogeneous processors interconnected with a heterogeneous network. Our algorithm is based on A* search which needs a heuristic function to estimate the cost of a subproblem. For this purpose, we have adapted an algorithm which solves the tree structured problems exactly in polynomial time. To better make use of that polynomial time algorithm and to reduce the search space drastically, we described a preprocessing step. We have reported a summary of our extensive experiments in which task assignment problems with about 300 tasks and 8 processors are solved in a reasonable amount of time.

Acknowledgements and availability

We thank Dr. Jack Dongarra and Dr. Bernard Tourancheau, the organizers of the Clusters and Computational Grids for Scientific Computing (CCGSC 2008) meeting, for encouraging us to write the paper. The authors keep a web page of the data set and the optimal solutions. As of time of writing, the page is at `http://graal.ens-lyon.fr/~bucar/tig/`.

References

[1] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Task execution time modeling for heterogeneous computing systems. In Cauligi Raghavendra, editor, *Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000)*, pages 185–199, Cancun, Mexico, May 2000. IEEE.

[2] B. Arafeh, K. Day, and A. Touzene. A multilevel partitioning approach for efficient tasks allocation in heterogeneous distributed systems. *J. Syst. Architect.*, 54:530–548, 2008.

[3] A. Billionnet. Allocating tree structured programs in a distributed system with uniform communication costs. *IEEE T. Parall. Distr.*, 5(4):445–448, 1994.

[4] S. H. Bokhari. A shortest tree algorithm for optimal assignments across space and time in distributed processor system. *IEEE T. Software Eng.*, 7:583–589, 1981.

[5] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. Heuristics for parameter sweep applications in Grid environments. In *Proc. Ninth Heterogeneous Computing Workshop*, pages 349–363. IEEE Computer Society Press, 2000.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, first edition, 1990.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[8] J. Gehring and A. Reinefeld. MARS – A framework for minimizing the job execution time in a metacomputing environment. *Future Gener. Comp Sy.*, 12:97–99, 1996.

[9] M. Kafil and I. Ahmad. Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurr.*, 6:42–51, 1998.

[10] Y.-K. Kwok and I. Ahmad. On multiprocessor task scheduling using efficient state space search approaches. *J. Parallel Distr. Com.*, 65:1515–1532, 2005.

[11] S. Menon. Effective reformulations for task allocation in distributed systems with a large number of communicating tasks. *IEEE T. Knowl. Data En.*, 16:1497–1508, 2004.

[12] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education Inc., New Jersey, USA, second edition, 2003.

[13] H. S. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE T. Software Eng.*, SE-3:85–93, 1977.

[14] A. Tom P. and C. S. R. Murthy. An improved algorithm for module allocation in distributed computing systems. *J. Parallel Distr. Com.*, 42:82–90, 1997.

[15] A. Tom P. and C. S. R. Murthy. Optimal task allocation in distributed systems by graph matching and state space search. *J Syst. Software*, 46:59–75, 1999.

[16] B. Uçar, C. Aykanat, K. Kaya, and M. Ikinci. Task assignment in heterogeneous computing systems. *J. Parallel Distr. Com.*, 66:32–46, 2006.

[17] J. B. Weissman and X. Zhao. Run-time support for scheduling parallel applications in heterogeneous nows. In *HPDC '97: Proc. of the 6th International Symposium on High Performance Distributed Computing*, pages 347–355, Portland, USA, 1997. IEEE.