Available online at www.sciencedirect.com

ScienceDirect

Journal of Discrete Algorithms 5 (2007) 162–175

JOURNAL OF DISCRETE ALGORITHMS

www.elsevier.com/locate/jda

On-line load balancing made simple: Greedy strikes back [☆]

Pilu Crescenzi [a,*], Giorgio Gambosi [b], Gaia Nicosia [c], Paolo Penna [d,1], Walter Unger [e]

[a] *Dipartimento di Sistemi e Informatica, Università di Firenze, Via Lombroso 6/17, 50134 Firenze, Italy*
[b] *Dipartimento di Matematica, Università di Roma "Tor Vergata", Via della Ricerca Scientifica, 00133 Roma, Italy*
[c] *Dipartimento di Informatica e Automazione, Università di Roma "Roma Tre", Via della Vasca Navale 79, 00146 Roma, Italy*
[d] *Dipartimento di Informatica ed Applicazioni "R.M. Capocelli", Università di Salerno, Via S. Allende 2, 84081 Baronissi (SA), Italy*
[e] *RWTH Aachen, Ahornstrasse 55, 52056 Aachen, Germany*

Received 31 May 2005; accepted 16 February 2006

Available online 3 May 2006

Abstract

We provide a new approach to the on-line load balancing problem in the case of restricted assignment of temporary weighted tasks. The approach is very general and allows us to derive on-line algorithms whose competitive ratio is characterized by some combinatorial properties of the underlying graph G representing the problem: in particular, the approach consists in applying the greedy algorithm to a suitably constructed subgraph of G. In the paper, we prove the NP-hardness of the problem of computing an optimal or even a c-approximate subgraph, for some constant $c > 1$. Nevertheless, we show that, for several interesting problems, we can easily compute a subgraph yielding an optimal on-line algorithm. As an example, the effectiveness of this approach is shown by the hierarchical server model introduced by Bar-Noy et al. (2001). In this case, our method yields simpler algorithms whose competitive ratio is at least as good as the existing ones. Moreover, the algorithm analysis turns out to be simpler. Finally, we give a sufficient condition for obtaining, in the general case, $O(\sqrt{n})$-competitive algorithms with our technique: this condition holds in the case of several problems for which a $\Omega(\sqrt{n})$ lower bound is known.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Approximation algorithm; Computational complexity; Load balancing; On-line algorithm

1. Introduction

Load balancing is a fundamental problem which has been extensively studied in the literature because of its many applications in resource allocation, processor scheduling, routing, network communication, and many others. The

[☆] A preliminary version of this paper has been presented at the *Thirtieth International Colloquium on Automata, Languages and Programming*, 2003. A similar title is used in [S. Guha, S. Khuller, Greedy strikes back: Improved facility location algorithms, in: ACM-SIAM Symposium on Discrete Algorithms, 1998] for a facility location problem.

[*] Corresponding author.
E-mail addresses: piluc@dsi.unifi.it (P. Crescenzi), gambosi@mat.uniroma2.it (G. Gambosi), nicosia@dia.uniroma3.it (G. Nicosia), penna@dia.unisa.it (P. Penna), quax@cs.rwth-aachen.de (W. Unger).

[1] Supported by the European Project IST-2001-33135, Critical Resource Sharing for Cooperation in Complex Systems (CRESCCO). Work partially done while at the Dipartimento di Matematica, Università di Roma "Tor Vergata" and while at the Institut für Theoretische Informatik, ETH Zentrum.

problem is to assign tasks to a set of n processors, where each task has an associated *load vector* and duration. Tasks must be assigned to exactly one processor, thereby increasing the load of that processor by the amount specified by the corresponding coordinate of the load vector for the duration of the task. Usually, the goal is to minimize the maximum load over all processors.

The *on-line* load balancing problem, in which tasks must be assigned upon their arrival, has several natural applications. For instance, consider the case in which processors represent channels and tasks are communication requests which arrive one by one. When a request is assigned to a channel, a certain amount of its bandwidth is reserved for the duration of the communication. Since channels have limited bandwidth, the maximum load is an important measure here.

Several variants of the on-line load balancing problem have been proposed depending on the structure of the load vectors, whether we allow *preemption* (i.e., to reassign tasks), whether tasks remain in the system "forever" or not (i.e., permanent vs. temporary tasks), whether the (maximum) duration of the tasks is known, and so on [1–5,12,14] (see also [6] for a survey).

In this paper, we study the on-line load balancing problem in the case of *temporary tasks* with *restricted assignment* and *no preemption*, that is:

- Tasks arrive one by one and their duration is unknown.
- Each task can be assigned to one processor among a subset depending on the type of the task.
- Once a task has been assigned to a processor, it cannot be reassigned to another one.
- Each task has a weight and assigning a task to a processor increases the processor load by an amount equal to the weight of the task.

The problem is to find an assignment of the tasks to the processors which minimizes the maximum load over all processors and over time.

Among others, this variant has a very important application in the context of wireless networks. In particular, consider the case in which we are given a set of base stations and each mobile user can only connect to a subset of them, that is, those that are "close enough". A user may unexpectedly appear in some spot and ask for a connection at a certain transmission rate (i.e., bandwidth). Also, the duration of this transmission is not specified (this is the typical case of a telephone call). Because of the application, it is desirable to avoid the reassignment of users to other base stations (i.e., to avoid the handover), unless this becomes unavoidable because a user moves away from the transmission range of its current base (in this latter case, we can model this as a termination of the current request and a new request appearing in the new position).

As usual, we compare the cost of a solution computed by an on-line algorithm with the best off-line algorithm which minimizes the maximum load *knowing the entire sequence* of task arrivals and departures. Informally, an on-line algorithm is r-competitive if, at any instant, its maximum processor load is at most r times the optimal maximum processor load.

It is convenient to formulate our problem by means of a bipartite graph with vertices corresponding to processors and possible "task types". More formally, let $P = \{p_1, \dots, p_n\}$ be a set of processors and let $\mathcal{T} \subseteq 2^P$ be a set of *task types*. The *associated bipartite graph* is $G_{P,\mathcal{T}}(X_{\mathcal{T}} \cup P, E_{\mathcal{T}})$, where

$$X_{\mathcal{T}} = \{x_1, \dots, x_{|\mathcal{T}|}\}$$

and

$$E_{\mathcal{T}} = \big\{(x_i, p_j) \mid p_j \text{ belongs to the } i\text{th element of } \mathcal{T}\big\}.$$

A *task* t is a pair (x, w), where $x \in X_{\mathcal{T}}$ and w is the positive integer weight of t. The set of processors to which t can be *assigned* is $P_t = \{p \mid (x, p) \in E_{\mathcal{T}}\}$, that is, the set of nodes of $G_{P,\mathcal{T}}$ that are adjacent to x. In the previous example of mobile users, the type of a task corresponds to the user's position.[2] In general, we consider two tasks which can be assigned to the same set of processors as belonging to the same type.

We follow the intuition that "nice" graphs may yield better competitive ratios, as one can see with the following examples:

[2] Clearly, this is a simplification of the reality where other constraints must also be taken into account.

General case We do not assume anything regarding the possible task types. Hence, the graph must contain all possible task types corresponding to any subset of processors, that is, $\mathcal{T} = 2^P$. Under this assumption, the best achievable ratio is $\Theta(\sqrt{n})$ [1,3] (see also [15]), while the greedy algorithm is exactly $\frac{3n^{2/3}}{2}(1 + o(1))$-competitive [1], and thus not optimal.

Identical machines There is only one task type since a task can be assigned to any of the processors. Therefore, the graph is the complete bipartite graph $K_{1,n}$ and the competitive ratio of the problem is $\Theta(2 - 1/n)$. This ratio is achieved by the greedy algorithm [10,11], which is optimal [2].

Hierarchical servers Processors are totally ordered and the type of a task corresponds to the "rightmost" processor that can execute that task. The set $\mathcal{T}$ contains one node per processor and the ith node of $\mathcal{T}$ is adjacent to all processors j, with $1 \leqslant j \leqslant i$. In this case, there exists a 5-competitive algorithm [7] and the greedy algorithm is at least $\Omega(\log n)$-competitive.

Noticeably, one can consider the first two cases as the two extremes of our problem, because of both the (non-) optimality of the greedy algorithm and the (non-) constant competitive ratio of the optimal on-line algorithm. From this point of view, the latter problem is somewhat in between.

A related question is whether the greedy approach performs badly because of the fact that it must decide where to assign a task only based on *local* information (i.e., the current load of those processors that can execute that task). Indeed, the optimal algorithm for the general case (see [3, Robin-Hood Algorithm]) requires the computation of (an estimation of) the off-line optimum, which seems hard to compute in this local fashion. Also the algorithms for the hierarchical server case [7] require the computation of a quantity related to the optimum which depends on the current assignment of tasks of several types (see [7, Algorithm Continuous and Optimal Lemma]).

The idea of exploiting combinatorial properties of the graph $G_{P,\mathcal{T}}$ has been first used in [9]. In particular, that approach is based on the construction of a suitable subgraph which is used by the greedy algorithm (in place of the original one). This subgraph is the union of a set of complete bipartite subgraphs (called clusters). This method can hence be seen as a modification of the greedy algorithm where the topology of the network is taken into account in order to limit the possible choices of the algorithm.[3] Therefore, the resulting algorithms use only "local information" as the greedy one does.

Several topologies have been considered in [9] for which the method improves over the greedy algorithm and matches the lower bounds for the problems. In all such cases, however, the improvement is only by a constant factor, since the greedy algorithm is already $O(1)$-competitive.

The main contribution of this paper is a new approach to the problem based on the construction of a suitable subgraph to be used by the greedy algorithm. In this sense, our work is similar to [9]. However, our results improve over the method presented in that paper, since:

- In some cases, such as the hierarchical server case, we obtain the same competitive ratio, but with a simpler construction and analysis.
- In other cases, such as the hierarchical server version in which processors are ordered as in a rooted tree (see below), we obtain optimal results that we are currently not able to prove with the method of [9] (even though we do not know whether this is actually impossible to do).
- In general, our approach subsumes the one in [9] since the latter can be seen as a special case of the one presented here: hence, our method can be made to enjoy the same competitive ratios obtained with the method in [9].

Our method, similarly to the method in [9], yields examples in which, with a simple modification of the greedy algorithm, there is a significant improvement of the competitive ratio. This arises from the relevant case of hierarchical topologies, for which we attain a competitive ratio of 5 (4 for unweighted tasks[4]), while the greedy algorithm is at least $\Omega(\log n)$-competitive in both cases. Table 1 summarizes the results obtained for these topologies: Even though we achieve the same asymptotic competitive ratio of [7], our algorithms and their analysis turn out to be much simpler.

[3] This approach is somewhat counterintuitive since the algorithm improves when adding further restrictions to it (not to the adversary). This is reminiscent of the well-known Braess' paradox [8,13], where the removal of some edges from a graph unexpectedly improves the latency of the flow at Nash equilibrium.

[4] We denote the version of these problems in which all tasks have weight 1 by *unweighted*.

Table 1

	Our method	Previous (asymptotic) best*	Greedy
Weighted	$5n/(n+2)$ [Theorem 15]	5 [7]	$\Omega(\log n)$ [folkl.]
Unweighted	$4n/(n+2)$ [Theorem 15]	4 [7]	$\Omega(\log n)$ [folkl.]

* Although Bar-Noy et al. [7] only demonstrated asymptotic bounds, their proof technique can be extended to show that their algorithm actually achieves a ratio slightly better than $4/(1+\pi^2/\log^2 n)$ (for the weighted case, $1+4/(1+\pi^2/\log^2 n)$).

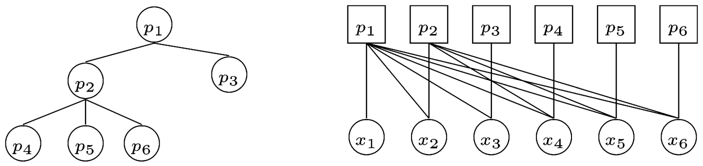

Fig. 1. An example of tree hierarchy and the corresponding bipartite graph.

We then turn our attention to the general case. In general, it might be desirable to automatically compute the best subgraph to be used by the greedy algorithm, as this would also yield a simple way to test the goodness of our method with respect to a given graph. Unfortunately, one of our results is the NP-hardness of the problem of computing an optimal or even a c-approximate subgraph, for some constant $c > 1$.

In spite of this negative result, we show that a "sufficiently good" subgraph can be obtained very easily in many cases. We first provide a *sufficient condition* (that is, the existence of a b-matching in a suitable subgraph of the graph $G_{\mathcal{P},\mathcal{T}}$, for some constant b independent of n) for obtaining $O(\sqrt{n})$-competitive algorithms with our technique. Notice that the lower bound for the general case is $\Omega(\sqrt{n})$, which also applies to randomized algorithms [1] and to sequences of tasks of length polynomial in n [15].

By using this result, we obtain a $(2\sqrt{n}+2)$-competitive algorithm for the hierarchical server version in which processors are ordered as in a rooted tree (see the example in Fig. 1). A $\Omega(\sqrt{n})$ lower bound for on-line algorithms also applies to this restriction [7], thus implying the optimality of our result. Additionally, we can achieve the same upper bound also when the ordering of the processors is given by *any* directed graph. This bound is only slightly worse than the $2\sqrt{n}+1$ bound given by the Robin-Hood algorithm presented in [3].

All algorithms obtained with our technique require only local information in the sense that they compute a task assignment only based on the current load of those processors that can be used for that task. This is a rather appealing feature since, in applications like the mobile networks above, introducing a global communication among bases for every new request to be processed may turn out to be unfeasible. Additionally, in several cases considered here (linear and tree hierarchical topologies), the construction of the subgraph is used solely for the analysis, while the actual on-line algorithm does not require any pre-computation (although the algorithm is different from the greedy one and it implements the subgraph used in the analysis).

Finally, we believe that our analysis is per se interesting since it translates the notion of adversary into a combinatorial property of the subgraph that we are able to construct during an off-line preprocessing phase. As a by-product, the analysis of our algorithms is simpler and more intuitive.

Paper road map. We introduce some notation and definitions in Section 1.1. The technique and its analysis are presented in Section 2. The hardness results are given in Section 2.4. We give a first application to the hierarchical topologies in Section 3. The application to the general case is described in Section 4 where we provide sufficient conditions for $O(\sqrt{n})$-competitiveness. These results are used in Section 4.1 where we obtain the results on generalized server hierarchies. Finally, in Section 5 we discuss some further features of our algorithms and present some open problems.

1.1. Preliminaries and notation

An *instance* σ of the on-line load balancing problem with processors P and task types $\mathcal{T}$ is defined as a sequence of $\texttt{new}(\cdot,\cdot)$ and $\texttt{del}(\cdot)$ commands. In particular: (i) $\texttt{new}(x, w)$ means that a new task of weight w and type $x \in \mathcal{T}$ is created; (ii) $\texttt{del}(i)$ means that the task created by the ith $\texttt{new}(\cdot,\cdot)$ command of the instance is deleted.

As already mentioned, we model the problem by means of a bipartite graph $G_{P,\mathcal{T}}(X_{\mathcal{T}} \cup P, E_{\mathcal{T}})$, where $\mathcal{T}$ depends on the problem version we are considering. For the sake of brevity, in the following we will always omit the subscripts '$P, \mathcal{T}$' and '$\mathcal{T}$' since the set of processors and the set of task types will be clear from the context. Given a graph $G(V, E)$, $\Gamma_G(v)$ denotes the neighborhood of the node $v \in V$. So, a task of type x can be assigned to any processor in $\Gamma_G(x)$. Furthermore, we denote by $E(v)$ the set of edges incident in v.

We will distinguish between the *unweighted* case, in which all tasks have weight 1, and the *weighted* case, in which the weights may vary from task to task. We also refer to (un-)weighted tasks to denote these variants.

Given an instance σ, a *configuration* is an assignment of the "alive" tasks of σ to the processors in P, such that each task $t = (x, w)$ is assigned to a processor in $P_t = \{p \mid (x, p) \in E_{\mathcal{T}}\}$. Given a configuration C, we denote with $l_C(i)$ the *load* of processor p_i, that is, the sum of the weights of all tasks assigned to p_i. In the sequel, we will usually omit the configuration when it will be clear from the context. The load of C is defined as the maximum of all the processor loads and is denoted with $l(C)$. Given an instance $\sigma = \sigma_1 \cdots \sigma_n$ and given an on-line algorithm $\mathcal{A}$, let $C_h^{\mathcal{A}}$ (with $1 \leqslant h \leqslant n$) be the configuration reached by $\mathcal{A}$ after processing the first h commands. Moreover, let C_h^{off} be the configuration reached by the optimal off-line algorithm after processing the first h commands. Let also $\mathsf{opt}(\sigma) = \max_{1 \leqslant h \leqslant n} l(C_h^{\text{off}})$ and $l_{\mathcal{A}}(\sigma) = \max_{1 \leqslant h \leqslant n} l(C_h^{\mathcal{A}})$.

An on-line algorithm $\mathcal{A}$ is said to be *r-competitive* if there exists a constant b such that, for any instance σ, it holds that $l_{\mathcal{A}}(\sigma) \leqslant r \cdot \mathsf{opt}(\sigma) + b$. An on-line algorithm $\mathcal{A}$ is said to be *strictly r-competitive* if, for any instance σ, it holds that $l_{\mathcal{A}}(\sigma) \leqslant r \cdot \mathsf{opt}(\sigma)$.

A simple on-line algorithm for the load-balancing problem described above is the *greedy algorithm* that assigns a new task to the least loaded processor among those processors that can serve the task. That is, whenever a $\texttt{new}(x, w)$ command is encountered and the current configuration is C, the greedy algorithm looks for the processor p_i in $P_{t=(x,w)}$ such that $l_C(i)$ is minimal and assigns the new task t to p_i (ties are broken arbitrarily).

2. (Sub-)graphs and (sub-)greedy algorithms

In the sequel we will describe an on-line algorithm whose competitive ratio depends on some combinatorial properties of $G(X \cup P, E)$. The two main ideas used in our approach are the following:

1. We remove some edges on G and then we apply the greedy algorithm to the resulting bipartite graph.
2. While removing edges we try to balance the number of processors that can be used by tasks of type $x \in X$ and the number of processors that the adversary can use to assign the same set of tasks in the original graph G.

Let us observe that our method aims to obtain a good competitive ratio by *adding further constraints* to the original problem: This indeed corresponds to removing a suitable set of edges from G.

Choosing which edges have to be removed depends on some combinatorial properties that must be satisfied by the resulting bipartite graph. Before giving a formal description of such properties, we will describe the basic idea behind our approach. To this aim, let us consider a generic iteration of an algorithm that has to assign a task of type $x \in X$, and let us assume that our algorithm takes into account a set $U(x)$ of processors and assigns the task to the least loaded one. In order to evaluate the competitive ratio of this approach we need to know which set $A(x)$ of processors an adversary can use to assign the overall load currently in $U(x)$ (see Fig. 2). As we will see in the sequel the competitive ratio of our algorithm is roughly $|A(x)|/|U(x)|$. In the following, we will show how the set $A(x)$ is determined by the choices of our algorithm in the previous steps.

2.1. The subgreedy algorithm...

We now formalize the idea above and we provide the performance analysis of the resulting algorithm. In the following, for any $U \subseteq E$, we set $G_U = G(X \cup P, U)$.

Definition 1 *(Used processors).* For any $x \in X$ and for any $U \subseteq E$, we define the set $U(x) = \Gamma_{G_U}(x)$ of *used processors*. Moreover, given a processor $p \in P$, we denote by $U^{-1}(p)$ the set of vertices in X that include p among their used processors, i.e., $U^{-1}(p) = \{x \mid p \in U(x)\}$.

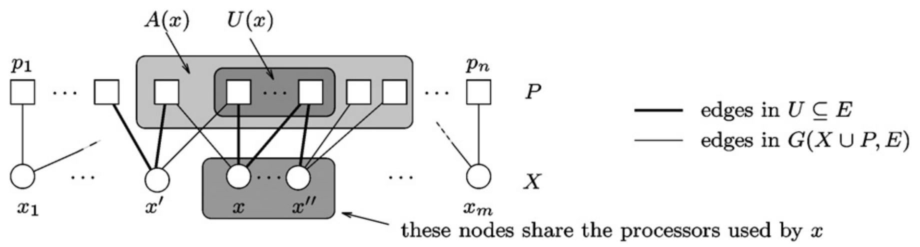

Fig. 2. The main idea of the sub-greedy algorithm is to balance the ratio between the number of used processors (that is, $|U(x)|$) and the number of adversary processors (that is, $|A(x)|$).

Definition 2 *(Adversary processors).* For any $x \in X$ and for any $U \subseteq E$, $A_U(x)$ denotes the set of processors that an off-line adversary can use to balance the load assigned to $U(x)$, i.e., $A_U(x) = \bigcup_{p \in U(x)} \bigcup_{x' \in U^{-1}(p)} \Gamma_G(x')$.

In the following, we will always omit the subscript 'U', since the subset of edges will be clear from the context. Moreover, it is easy to see that $A(x) = \Gamma_G(\Gamma_{G_U}(\Gamma_{G_U}(x)))$ (see also Fig. 2).

Definition 3 *(Sub-greedy algorithm).* For any $U \subseteq E$ such that $U(x) \neq \emptyset$ for any $x \in X$, the *sub-greedy algorithm* (relative to U) is defined as the greedy on-line algorithm applied to $G_U = G(X \cup P, U)$.

Remark 4. The sub-greedy algorithm is a generalization of the cluster algorithm presented in [9], since the latter imposes each connected component of G_U to be a complete bipartite graph [9, Definition of Cluster].

2.2. *... and its analysis*

It is clear that the performance of the sub-greedy algorithm will depend on the choice of the set $U \subseteq E$. In particular, we can characterize its competitive ratio in terms of the ratio between the set $A(x)$ of adversary processors and the set $U(x)$ of used processors. More formally, let us consider the following quantities:

$$\rho_{\mathsf{w}}(U) = \max_{x \in X}\left\{\frac{|A(x)| - 1}{|U(x)|}\right\} \quad \text{and} \quad \rho_{\mathsf{u}}(U) = \max_{x \in X}\left\{\frac{|A(x)|}{|U(x)|}\right\}.$$

Then, the following two results hold.

Theorem 5. *The sub-greedy algorithm is strictly* $(1 + \rho_{\mathsf{w}}(U))$*-competitive in the case of weighted tasks.*

Proof. Let p_i be the processor with the highest load and let $t = (w, x)$ be the last task assigned to p_i by the sub-greedy algorithm. Since t has been assigned to p_i whose load, before the arrival of t, was $l(i) - w$, we have that each processor in $U(x)$ had load at least $l(i) - w$. So, the overall load of $U(x)$ is at least $|U(x)|(l(i) - w) + w$. The number of processors that any off-line strategy can use to spread such load is equal to $|A(x)|$: Hence, the optimal off-line solution has measure

$$l^* \geqslant \max\left\{\frac{|U(x)|l(i) - w(|U(x)| - 1)}{|A(x)|}, w\right\} \geqslant \frac{|U(x)|l(i)}{|A(x)| + |U(x)| - 1},$$

where the second inequality is due to the fact that the minimum value of the max function is reached when $w = \frac{|U(x)|l(i) - w(|U(x)|-1)}{|A(x)|}$. Hence, the following bound on the competitive ratio holds:

$$\frac{l(i)}{l^*} \leqslant \frac{|A(x)| + |U(x)| - 1}{|U(x)|} \leqslant 1 + \rho_{\mathsf{w}}(U).$$

Thus, the theorem follows. □

Theorem 6. *The sub-greedy algorithm is strictly* $\lceil \rho_{\mathsf{u}}(U) \rceil$*-competitive in the case of unweighted tasks. Moreover, in this case the algorithms is* $\rho_{\mathsf{u}}(U)$*-competitive.*

Proof. Let us consider a generic iteration of the sub-greedy algorithm in which a task t arising in $x \in X$ has been assigned to $p_i \in U(x)$. Since t has been assigned to p_i whose load, before the arrival of t, was $l(i) - 1$, we have that each processor in $U(x)$ had load at least $l(i) - 1$. This implies that the overall number of tasks in $U(x)$, after the arrival of t, was at least $|U(x)|(l(i) - 1) + 1$. On the other hand, the number of processors to which the off-line optimal solution can assign these tasks is at most $|A(x)|$. Thus, the optimal off-line solution has measure at least

$$l^* \geqslant \frac{|U(x)|(l(i)-1)+1}{|A(x)|} \geqslant \frac{l(i)-1}{\rho_{\mathsf{u}}(U)} + \frac{1}{|A(x)|}. \tag{1}$$

By contradiction, let us suppose that $l(i) > l^* \lceil \rho_{\mathsf{u}}(U) \rceil$. Then, since both $l(i)$ and l^* are integers, we have that $l(i) - 1 \geqslant l^* \lceil \rho_{\mathsf{u}}(U) \rceil \geqslant l^* \rho_{\mathsf{u}}(U)$. This leads to the following contradiction:

$$l^* \geqslant \frac{l(i)-1}{\rho_{\mathsf{u}}(U)} + \frac{1}{|A(x)|} \geqslant \frac{l^* \rho_{\mathsf{u}}(U)}{\rho_{\mathsf{u}}(U)} + \frac{1}{|A(x)|} > l^*.$$

We have thus proved that the sub-greedy algorithm is strictly $\lceil \rho_{\mathsf{u}}(U) \rceil$-competitive. Finally, Eq. (1) implies that

$$l(i) < l^* \rho_{\mathsf{u}}(U) + 1.$$

Hence, the sub-greedy algorithm is also $\rho_{\mathsf{u}}(U)$-competitive, and the theorem follows. □

2.3. *The sub-greedy* algorithm*

Consider the bipartite graph $G(X \cup P, E)$ with $X = \{x_1, \ldots, x_n\} \cup \{y\}$, $P = \{p_1, \ldots, p_n\}$ and $E = \{(x_i, p_i) \mid 1 \leqslant i \leqslant n\} \cup \{(y, p_i) \mid 1 \leqslant i \leqslant n\}$. It is easy to see that any subset U of edges yields $\rho_{\mathsf{w}}(U) = n - 1$. However, a rather simple idea might be to separate the high-degree vertex y from the low-degree vertices $x_1, \ldots, x_n$ and to process tasks of type y *independently* from tasks of type $x_1, \ldots, x_n$. More generally, we define the following variation of the sub-greedy algorithm.

Definition 7 *(Sub-greedy* algorithm).* Let $X_1, \ldots, X_k$ be any partition of the set X of the task type vertices of $G(X \cup P, E)$. Also let $G_i = G(X_i \cup P, E_i)$ be the corresponding induced subgraph, and let $U_i \subseteq E_i$ for $1 \leqslant i \leqslant k$. The *sub-greedy* algorithm* (relative to $X_1, \ldots, X_k$ and to $U_1, \ldots, U_k$) is defined as the sub-greedy algorithm applied separately to each of the subgraphs G_i (relative to U_i) with only tasks of type X_i as input.

In the sequel we denote by $\rho_{\mathsf{w}}(U_i, G_i)$ the quantity $\rho_{\mathsf{w}}(U)$ computed with respect to the graph G_i and to the subset of edges $U_i \subseteq E_i$.

Theorem 8. *The sub-greedy* algorithm is strictly $(k + \rho^*_{\mathsf{w}}(U))$-competitive in the case of weighted tasks, where k is the number of subgraphs and $\rho^*_{\mathsf{w}}(U) = \sum_{i=1}^{k} \rho_{\mathsf{w}}(U_i, G_i)$.*

Proof. Given a sequence of tasks σ, let $\sigma(i)$ denote the subsequence containing tasks whose type is in X_i and in which the `del(·)` commands have been appropriately reindexed. Also let $l(j)$ denote the load of processor p_j at some time step and $l^i(j)$ the load at the same time step restricted to tasks whose type is in X_i. Then, the definition of the sub-greedy* algorithm and Theorem 5 imply that $\max_{1 \leqslant j \leqslant n} l^i(j) \leqslant \mathsf{opt}(\sigma(i))(1 + \rho_{\mathsf{w}}(U_i, G_i))$, for $1 \leqslant i \leqslant k$. It then holds that

$$\max_{1 \leqslant j \leqslant n} l(j) \leqslant \sum_{i=1}^{k} \max_{1 \leqslant j \leqslant n} l^i(j) \leqslant \sum_{i=1}^{k} \mathsf{opt}\big(\sigma(i)\big)\big(1 + \rho_{\mathsf{w}}(U_i, G_i)\big) \tag{2}$$

$$\leqslant k \cdot \mathsf{opt}(\sigma) + \mathsf{opt}(\sigma) \sum_{i=1}^{k} \rho_{\mathsf{w}}(U_i, G_i), \tag{3}$$

where the last inequality follows from the fact that $\mathsf{opt}(\sigma(i)) \leqslant \mathsf{opt}(\sigma)$. □

Note that, if we apply the above theorem to the previous example, we have that the sub-greedy* algorithm is $(2 + \frac{n-1}{n})$-competitive. More importantly, the above theorem will be a key-ingredient in deriving algorithms for the general case (see Section 4).

2.4. Computing good subgraphs

From Theorem 5 it follows that, in order to attain a good competitive ratio, it is necessary to select a subset of edges $U \subseteq E$ such that $\rho_{\mathsf{w}}(U)$ is as small as possible. Similarly, Theorem 8 implies that U should minimize $k + \rho^*_{\mathsf{w}}(U)$, when considering the sub-greedy* algorithm. This leads to the following two minimization problems.

MIN WEIGHTED ADVERSARY SUBGRAPH (MWAS)
Instance. A bipartite graph $G(X \cup P, E)$.
Solution. A subset $U \subseteq E$ such that, for every $x \in X$, $|\Gamma_{G_U}(x)| \geqslant 1$.
Measure. $\rho_{\mathsf{w}}(U, G) = \max_{x \in X} \frac{|\Gamma_G(\Gamma_{G_U}(\Gamma_{G_U}(x)))|-1}{|\Gamma_{G_U}(x)|}$.

MIN WEIGHTED ADVERSARY MULTI-SUBGRAPH (MWAMS)
Instance. A bipartite graph $G(X \cup P, E)$.
Solution. A partition $X_1, \ldots, X_k$ of X and a collection $U = \{U_1, \ldots, U_k\}$ of subsets of edges $U_i \subseteq E_i$, where E_i denotes the set of edges of the subgraph G^i of G induced by $X_i \cup P$, such that, for every $1 \leqslant i \leqslant k$ and for any $x \in X_i$, $|\Gamma_{G^i_{U_i}}(x)| \geqslant 1$.
Measure. $k + \rho^*_{\mathsf{w}}(U, G) = k + \sum_{i=1}^{k} \rho_{\mathsf{w}}(U_i, G^i)$.

According to Theorem 6, we define the MIN UNWEIGHTED ADVERSARY SUBGRAPH (MUAS) problem by replacing ρ_{w} with ρ_{u} in the definition of MWAS. By using the same technique of Theorem 8, it is possible to show that, for unweighted tasks, the sub-greedy* algorithm is $\rho^*_{\mathsf{u}}(U, G)$-competitive, where

$$\rho^*_{\mathsf{u}}(U, G) = \sum_{i=1}^{k} \rho_{\mathsf{u}}\bigl(U_i, G^i\bigr).$$

We thus define the MIN UNWEIGHTED ADVERSARY MULTI-SUBGRAPH (MUAMS) problem as the problem obtained by modifying, in the definition above, the measure of MWAMS by $\rho^*_{\mathsf{u}}(U, G) = \sum_{i=1}^{k} \rho_{\mathsf{u}}(U_i, G^i)$.

It is possible to construct a reduction showing the NP-hardness of all these problems. The same reduction is a gap-creating reduction, thus implying the non existence of a PTAS for any such problem. In particular, we have the following result.

Theorem 9. *MUAS cannot be approximated within a factor smaller than* $\frac{21}{20}$, *unless* $\mathsf{P} = \mathsf{NP}$. *The same result also applies to the MUAMS problem.*

Proof. We first consider the MUAS problem. Let $F = \{c_1, \ldots, c_m\}$ be a 3CNF Boolean formula with variable set $\{x_1, \ldots, x_n\}$. We construct a bipartite graph $G^F(X^F \cup P^F, E^F)$ such that

B1 if F is satisfiable then there exists $U \subseteq E^F$ for which $\rho_{\mathsf{u}}(U, G^F) \leqslant 5/3$;
B2 if F is not satisfiable, then, for any $U \subseteq E^F$, $\rho_{\mathsf{u}}(U, G^F) \geqslant 7/4$.

The set of edges E^F is partitioned into three sets E^F_{T}, E^F_{F} and $E^F_{\star}$, also called T-edges, F-edges and $\star$-edges, respectively. For each variable x_i and for each clause c_j, we construct a corresponding gadget as shown in Fig. 3. In particular, all nodes are newly added nodes and edges are labeled according to the set, among E^F_{T},E^F_{F} and $E^F_{\star}$, they belong to.

We then connect the variable gadget of x_i to the gadgets of those clauses c_j containing x_i as follows. Let $c_j = (l^1_j, l^2_j, l^3_j)$ and let $l = l^k_j$, $1 \leqslant k \leqslant 3$ (i.e., $l \in c_j$).[5] In this case, we connect node l with node L^k_j by means of a (suitable) path. The set of all connections from l to all gadgets of the clauses containing l is obtained by means of the "tree-like" graph T_l shown in the left part of Fig. 4 (observe that T_l could be unbalanced and could have dangling leaves not connected to any clause gadget, but this does not affect our proof in any way): T_l contains at the rightmost level

[5] Notice that we consider each clause as an ordered set of three literals.

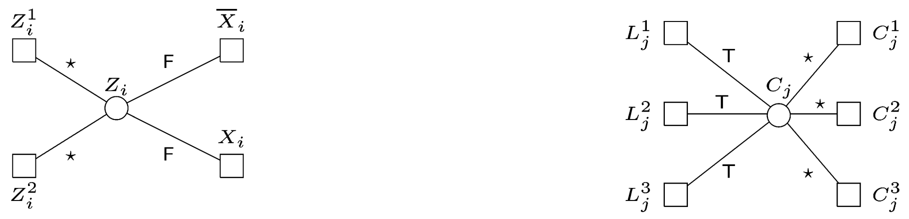

Fig. 3. The variable gadget corresponding to x_i (left part) and the clause gadget corresponding to c_j (right part).

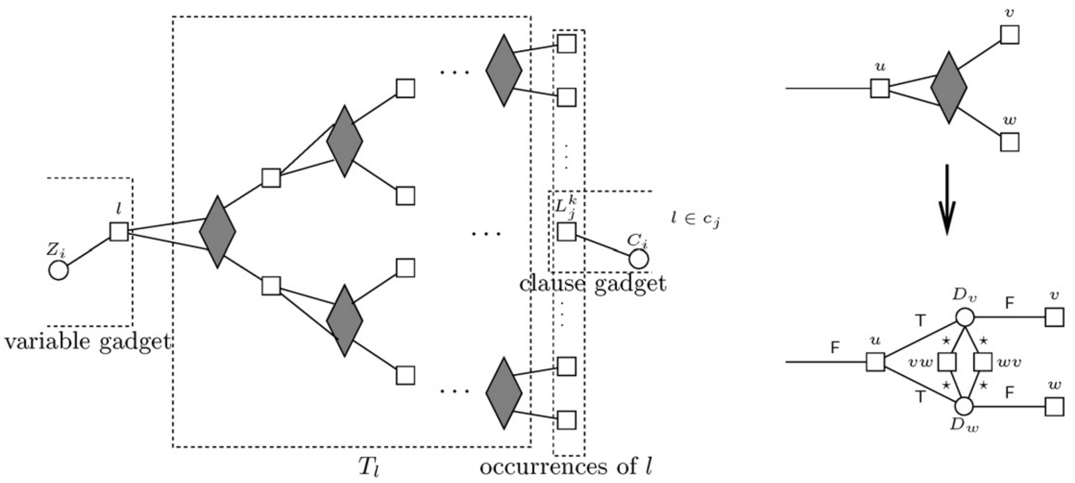

Fig. 4. The "tree-like" graph T_l (left part) and the gadget corresponding to a diamond (right part).

$\lceil m_l/2 \rceil$ diamonds, where m_l is the number of occurrences of l in F. Each diamond corresponds to a gadget as shown in the right part of Fig. 4.

The whole graph G^F is obtained by building the graph T_l for each literal and by connecting every such T_l as in Fig. 4. Observe that the clause gadget of $c_j = (l_j^1, l_j^2, l_j^3)$ is connected to the three graphs $T_{l_j^1}$, $T_{l_j^2}$ and $T_{l_j^3}$. Finally, the partition of the set of nodes is obtained by including in X^F (respectively, P^F) all the circle (respectively, square) nodes. Observe that the circle nodes are of three types: the Z-nodes which are included in a variable gadget, the C-nodes which are included in a clause gadget, and the D-nodes which are included in a diamond gadget.

Lemma 10. *If F is satisfiable, then there exists U such that, for any circle node x,*

$$\left|A(x)\right| \leqslant \begin{cases} 4 & \text{if } x \text{ is a } Z\text{-node,} \\ 5 & \text{if } x \text{ is a } D\text{-node,} \\ 6 & \text{if } x \text{ is a } C\text{-node,} \end{cases} \quad \text{and} \quad \left|U(x)\right| \geqslant \begin{cases} 3 & \text{if } x \text{ is a } Z\text{-node,} \\ 3 & \text{if } x \text{ is a } D\text{-node,} \\ 4 & \text{if } x \text{ is a } C\text{-node.} \end{cases}$$

Proof. If F is satisfiable, then there exists a truth-assignment such that, for each clause c_j, there exists a literal l_j in c_j which is assigned the value true. We then derive the subset U of E^F as follows:

- U includes all the $*$-edges.
- For each clause c_j, U includes all the T-edges on the (single) $*$-free path[6] connecting Z_{i_j} and C_j, where Z_{i_j} is the Z-node corresponding to the variable in l_j. In addition, U includes all the F-edges not on the above paths.

It is easy to verify that, for any circle node x, U satisfies the bounds stated in the lemma. □

[6] A $*$-free path is a path that does not include any $*$-edge.

From the previous lemma, the upper bound (B1) follows: in order to prove the lower bound (B2), we introduce the notion of a canonical solution.

Let U be a subset of E^F. We say that U is *canonical* if it satisfies the following properties.

P1 For any Z-node and for any D-node x, U includes at least three edges incident to x, while, for any C-node x, U includes at least four edges incident to x.

P2 If U includes an F-edge e, then U does not include any T-edge that intersects e in a square node.

Lemma 11. *If a canonical U exists, then F is satisfiable.*

Proof. From (P1) it follows that, for each clause c_j, U includes at least one T-edge e_j incident to C_j. From (P1) and (P2) it follows that the $*$-free path starting with e_j and ending at a Z-node Z_i terminates with an F-edge which is not included in U, while from (P1) it follows that the other F-edge incident to Z_i belongs to U. Hence, the assignment that assigns the value true to all literals corresponding to Z-nodes which are extremes of an F-edge not included in U is a truth-assignment that satisfies F. □

Lemma 12. *Let U be a subset of E^F such that $\rho_{\mathsf{u}}(U, G^F) < 7/4$. Then U is canonical.*

Proof. U satisfies (P1): Otherwise, there would exist a Z-node or a D-node (respectively, C-node) x such that $|U(x)| \leqslant 2$ (respectively, $|U(x)| \leqslant 3$) and $|A(x)| \geqslant 4$ (respectively, $|A(x)| \geqslant 6$), contradicting the fact that $\rho_{\mathsf{u}}(U, G^F) < 7/4$. Moreover, U satisfies (P2). On the contrary, let us assume that U includes both an F-edge (x, y) and a T-edge (y, w), where y is a square node. Then x must be a Z-node or a D-node since it is adjacent to an F-edge: Hence, $|U(x)| \leqslant 4$ and $|A(x)| \geqslant 3 + |U(x)|$, contradicting the fact that $\rho_{\mathsf{u}}(U, G^F) < 7/4$. □

From the previous two lemmata the lower bound (B2) follows. We have thus proved the theorem for the MUAS problem.

Let us now consider G^F as an instance of the MUAMS problem. Since any solution of MUAS is a solution of MUAMS with the same cost, the upper bound (B1) holds also in this case. In order to extend the lower bound (B2) to the MUAMS case, observe that any solution for this latter problem, that is, a partition $X_1, \ldots, X_k$ of X^F and a family of subsets $U = \{U_1, \ldots, U_k\}$, cannot achieve a measure $\rho^*_{\mathsf{u}}(U, G^F)$ smaller than 2, unless $k = 1$. Hence, this solution is also feasible for the MUAS problem and $\rho^*_{\mathsf{u}}(U, G^F) = \rho_{\mathsf{u}}(U, G^F)$. The same hardness results thus apply to MUAMS too. □

By using the same reduction of the proof of the previous theorem, we can also show the following result.

Theorem 13. *MWAS cannot be approximated within a factor smaller than* $9/8$, *unless* $\mathsf{P} = \mathsf{NP}$. *Moreover, MWAMS cannot be approximated within a factor smaller* $15/14$, *unless* $\mathsf{P} = \mathsf{NP}$.

Proof. The proof is similar to the one of Theorem 9 and it uses the same upper-bound/lower-bound outline. In particular, if F is satisfiable, then from Lemma 10 it follows that there exists $U \subseteq E^F$ for which $\rho_{\mathsf{w}}(U, G^F) \leqslant 4/3$ and for which $\rho^*_{\mathsf{w}}(U, G^F) \leqslant 7/3$.

Let U be a subset of E^F such that $\rho_{\mathsf{w}}(U, G^F) < 3/2$. Similarly to the proof of Lemma 12, we can show that U is canonical. From Lemma 11, we know that if there exits a canonical U, then F is satisfiable. Hence, if F is not satisfiable, then, for any $U \subseteq E^F$, $\rho_{\mathsf{w}}(U, G^F) \geqslant 3/2$.

For what concerns MWAMS, observe that if a solution U satisfies $\rho^*_{\mathsf{w}}(U, G) < 5/2$, then it must be the case $k \leqslant 2$. If $k = 2$, then let $G^{F,1}$ and $G^{F,2}$ denote the two subgraphs of G^F corresponding to U_1 and U_2, respectively: Note that every circle node retains its degree in the subgraph it belongs to. Let x be a circle node of degree four in one of the two subgraphs (say, $G^{F,1}$): Then, $(|A_{U_1}(x)| - 1)/|U_1(x)| \geqslant 3/4$. By definition, we obtain $2 + \rho_{\mathsf{w}}(U_1, G^{F,1}) + \rho_{\mathsf{w}}(U_2, G^{F,2}) \geqslant 2 + 3/4 > 5/2$. We have thus proven that, if a solution U satisfies $\rho^*_{\mathsf{w}}(U, G) < 5/2$, then it must be the case $k = 1$, that is, U is also a solution for MWAS problem. Hence, the hypothesis $\rho^*_{\mathsf{w}}(U, G^F) < 5/2$ implies that $\rho_{\mathsf{w}}(U, G^F) < 3/2$. In conclusion, if F is not satisfiable, then, for any $U \subseteq E^F$, $\rho^*_{\mathsf{w}}(U, G^F) \geqslant 5/2$. □

3. Application to hierarchical server topologies

In this section we apply our method to the hierarchical server topologies introduced in [7]. In particular, we consider the *linear* hierarchical topology: Processors are ordered from p_1 (the most capable processor) to p_n (the least capable processor) in decreasing order with respect to their capabilities. So, if a task can be assigned to processor p_i, for some i, then it can also be assigned to any p_j with $1 \leqslant j < i$. We can therefore consider the task types as corresponding to the sets $\{p_1, \ldots, p_i\}$, for each $1 \leqslant i \leqslant n$.

The resulting bipartite graph $G(X \cup P, E)$ is given by $X = \{x_1, \ldots, x_n\}$, $P = \{p_1, \ldots, p_n\}$ and $E = \{(x_i, p_j) \mid x_i \in X \wedge p_j \in P \wedge 1 \leqslant j \leqslant i \leqslant n\}$. We denote this graph as K_n^{lht}.

We next provide an efficient construction of subgraphs of K_n^{lht}.

Lemma 14. *For any positive integer n, there exists a U such that*

$$\rho_{\mathsf{w}}\big(U, K_n^{lht}\big) \leqslant (4n-2)/(n+2)$$

and

$$\rho_{\mathsf{u}}\big(U, K_n^{lht}\big) \leqslant 4n/(n+2).$$

Moreover, the set U can be computed in linear time.

Proof. For each $1 \leqslant i \leqslant n$, we define the set $U(x_i)$ as

$$U(x_i) = \{p_{\lceil i/2 \rceil}, \ldots, p_i\} = \begin{cases} \{p_{i/2}, \ldots, p_i\} & \text{if } i \text{ is even,} \\ \{p_{(i+1)/2}, \ldots, p_i\} & \text{otherwise.} \end{cases}$$

Clearly, $|U(x_i)| = i/2 + 1$ if i is even, and $|U(x_i)| = (i+1)/2$ otherwise. Moreover, $|A(x_i)| = \max_{i \leqslant j \leqslant n}\{j \mid U(x_i) \cap U(x_j) \neq \emptyset\}$. It is easy to see that $|A(x_i)| \leqslant 2i$, thus implying

$$\rho_{\mathsf{w}}(U) = \max_{1 \leqslant i \leqslant n} \frac{|A(x_i)| - 1}{|U(x_i)|} \leqslant \max_{1 \leqslant i \leqslant n} \frac{4i-2}{i+1} \leqslant \frac{4n-2}{n+1}.$$

A better bound can be obtained by distinguishing two cases: (1) $i \leqslant n/2$ and (2) $i \geqslant n/2 + 1$. In case (1) we apply the bound above, while for (2) we simply use $|A(x_i)| \leqslant n$; in both cases we obtain $\rho_{\mathsf{w}}(U) \leqslant (4n-2)/(n+2)$.

With a similar proof we can show that the same construction yields $\rho_{\mathsf{u}}(U) \leqslant 4n/(n+2)$. □

An immediate application of Lemma 14 combined with Theorems 5 and 6 is the following result.

Corollary 15. *In the case of linear hierarchy topologies, the sub-greedy algorithm is strictly $5n/(n+2)$-competitive in the case of weighted tasks, and strictly $4n/(n+2)$-competitive for unweighted tasks.*

Notice that our approach matches the 5-competitive (respectively, 4-competitive) algorithm for weighted (respectively, unweighted) tasks given in [7]: However, our construction and proof are simpler (at the cost of being less tight than the implicit bounds of [7]).

4. The general case

We now provide a sufficient condition for obtaining $O(\sqrt{n})$-competitive algorithms. This condition applies to the hierarchical server model when the ordering of the servers forms a tree. Thus, in this case our result is optimal because of the $\Omega(\sqrt{n})$ lower bound [7].

We first define an overall strategy to select the set $U(x)$ depending on the degree $\delta(x)$ of x.

High degree (easy case). $\delta(x) \geqslant \sqrt{n}$. In this case we use *all of the adjacent vertices in P*. Since $|U(x)| = \delta(x) \geqslant \sqrt{n}$, we have $|A(x)|/|U(x)| \leqslant \sqrt{n}$.

Low degree (hard case). $\delta(x) < \sqrt{n}$. For low degree vertices our strategy will be to choose a *single* processor p_x^* in $\Gamma_G(x) \subseteq P$. The choice of this element must be carried out carefully in order to guarantee that $|A(x)| \leqslant \sqrt{n}$. For instance, it would suffice that p_x^* does not appear in any other set $\Gamma_G(x')$.

Then, our next idea will be to partition the graph $G(X \cup P, E)$ into two subgraphs $G_l(X_l \cup P, E_l)$ and $G_h(X_h \cup P, E_h)$ containing low and high degree vertices, respectively. Notice that, if we are able to have a $f(n)$-competitive algorithm for the low degree graph, then we have a $O(\sqrt{n} + f(n))$-competitive algorithm for our problem (see Theorem 8).

We next focus on low degree graphs and we provide sufficient conditions for obtaining $O(\sqrt{n})$-competitive algorithms.

Theorem 16. *If $G_l(X_l \cup P, E_l)$ admits a b-matching, then the sub-greedy* algorithm is at most $((b+1)\sqrt{n}+2)$-competitive.*

Proof. Let U be a b-matching for G_l. It is easy to see that, in G_l, $|A(x)| \leqslant b\sqrt{n}$, for all $x \in X_l$. Thus, $\rho_{\mathrm{w}}(U, G_l) \leqslant b\sqrt{n}$. By definition of G_h, $\rho_{\mathrm{w}}(E_h, G_h) \leqslant \sqrt{n}$. We can thus apply Theorem 8 with $k = 2$, $U_1 = U$ and $U_2 = E_h$. Hence the theorem follows. □

Theorem 17. *If $G(X \cup P, E)$ admits a b-matching, then the sub-greedy* algorithm is at most $(2\sqrt{bn}+2)$-competitive.*

Proof. Define low-degree vertices as those $x \in X$ such that $\Gamma_G(x) \leqslant \sqrt{n/b}$. Subgraphs G_l and G_h are defined accordingly. The existence of a b-matching U yields $\rho_{\mathrm{w}}(U, G_l) \leqslant b\sqrt{n/b}$. By definition of G_h, $\rho_{\mathrm{w}}(E_h, G_h) \leqslant \sqrt{bn}$. We can thus apply Theorem 8 with $k = 2$, $U_1 = U$ and $U_2 = E_h$. Hence the theorem follows. □

Theorem 18. *If $G(X \cup P, E)$ admits a matching, then the sub-greedy algorithm is at most $\delta_{\max}$-competitive, where $\delta_{\max} = \max_{x \in X} |\Gamma_G(x)|$.*

Proof. Let U be a matching for G. Then, $|A(x)| \leqslant |\Gamma_G(x)|$ and $|U(x)| = 1$, for any $x \in X$. □

4.1. Generalized hierarchical server topologies

We now apply these results to the hierarchical model in the case in which the ordering of the servers forms a tree. Fig. 1 shows an example of this problem version: Processors are arranged on a rooted tree and there is a task type x_i for each node p_i of the tree; a task of type x_i can be assigned to processor p_i or to any of its ancestors.

We first generalize this problem version to a more general setting.

Definition 19. Let $H(P, F)$ be a directed graph. The associated bipartite graph $G_H(X \cup P, E)$ is defined as $X = \{x_1, \ldots, x_n\}$, and $(x_i, p_j) \in E$ if and only if $i = j$ or there exists a directed path in H from p_i to p_j.

Lemma 20. *Let $H(P, F)$ be any directed graph. Then $G_H(X \cup P, E)$ admits a matching.*

Proof. It is easy to see that $M = \{(x_i, p_i) \mid 1 \leqslant i \leqslant n\}$ is a matching for G_H. □

We can model a tree hierarchy by considering a rooted tree T whose edges are directed *upward*. We then obtain the following result.

Theorem 21. *Let $T(P, E)$ be any rooted tree and let $G_T(X \cup P, E)$ be the corresponding bipartite graph. Then, the sub-greedy* algorithm is at most $(2\sqrt{n}+2)$-competitive. Moreover, the sub-greedy algorithm is at most h-competitive, where h is the height of T.*

Proof. The first part of the theorem follows from Lemma 20 and from Theorem 16 with $b = 1$. The second part follows from Theorem 18. □

Finally, the $(2\sqrt{n}+2)$ upper bound can be extended to any graph hierarchy, as stated by the following result, which is the last theorem of this paper.

Theorem 22. *Let $H(P,F)$ be any directed graph representing an ordering among processors, and let $G_H(X \cup P, E)$ be the corresponding bipartite subgraph. Then, the sub-greedy* algorithm is at most $(2\sqrt{n}+2)$-competitive.*

Proof. The theorem follows from Lemma 20 and from Theorem 16 with $b = 1$. □

5. Conclusions and open problems

We have presented a novel technique which allows to derive on-line algorithms with a simple modification of the greedy one. This modification preserves the good feature of deciding where to assign a task solely based on the current load of processors to which that task can be potentially assigned to. Indeed, the pre-computation of the subgraph required by our approach is performed off-line given the graph representing the problem constraints. Additionally, for several cases we have considered here, this subgraph is only used in the analysis, while the resulting algorithms are simple modifications of the greedy version *implementing* the subgraph: The construction of Lemma 14 yields an algorithm performing a greedy choice on the rightmost half of the available processors $\Gamma_G(x_i) = \{p_1, \ldots, p_i\}$. So, this algorithm can be implemented even without knowing n. A similar argument applies to the sub-greedy* algorithm with the subgraph of Theorem 21: In this case knowing n is enough to decide whether a vertex is "low-degree" or not; in the latter case the matching (x_i, p_i) yields a fixed assignment for tasks corresponding to type x_i.

A distinguishing feature of the sub-greedy algorithm is that it only requires knowledge of the current load, and does not require to keep track of history: this is not true for the sub-greedy* algorithm, since it needs to remember the task types. Indeed, the adopted strategy of the sub-greedy* algorithm depends on the type of the task and on the current load of the adjacent processors in the appropriate subgraph. Since the algorithm assigns tasks corresponding to different subgraphs *independently*, it must be able to compute the load of a processor with respect to a subset X_i; this can be easily done whenever tasks are specified as pairs (x, w).

For the generalized hierarchical topologies, the competitive ratio of our algorithm is only slightly worse than the $2\sqrt{n}+1$ upper bound provided by the Robin-Hood algorithm [3]. Also, for tree hierarchical topologies, our analysis yields a much better ratio whenever the height h of the tree is $o(\sqrt{n})$ (e.g., for balanced trees).

An interesting direction for future research might be that of characterizing the competitive ratio of "local" algorithms under several assumptions on the graph $G(X \cup P, E)$ such as: (1) G is *unknown*, (2) G is uniquely determined by n, but n is unknown, (3) G is known.

Finally, a related question is: Under which hypothesis does our technique yield optimal competitive ratios?

Acknowledgements

We are very grateful to the anonymous referee whose valuable comments allowed us to significantly improve the presentation of this paper both from a technical and from a stylistic point of view. Moreover, the fourth author wishes to thank Amotz Bar-Noy for a useful discussion and for bringing the work [7] to his attention.

References

[1] Y. Azar, A. Broder, A. Karlin, Online load balancing, Theoretical Computer Science 130 (1994) 73–84.
[2] Y. Azar, L. Epstein, On-line load balancing of temporary tasks on identical machines, in: Proc. 5th Israeli Symposium on Theory of Computing and Systems, 1997, pp. 119–125.
[3] Y. Azar, B. Kalyanasundaram, S. Plotkin, K. Pruhs, O. Waarts, Online load balancing of temporary tasks, Journal of Algorithms 22 (1997) 93–110.
[4] S. Albers, Better bounds for on-line scheduling, in: Proc. 29th ACM Symp. on Theory of Computing, 1997, pp. 130–139.
[5] Y. Azar, J. Naor, R. Rom, The competitiveness of online assignments, Journal of Algorithms 18 (1995) 221–237.
[6] Y. Azar, On-line load balancing, in: A. Fiat, G. Woeginger (Eds.), On-line Algorithms—The State of the Art, Springer-Verlag, Berlin, 1998.
[7] A. Bar-Noy, A. Freund, J. Naor, On-line load balancing in a hierarchical server topology, SIAM Journal on Computing 31 (2) (2001) 527–549.
[8] D. Braess, Ueber ein paradoxon der verkehrsplanung, Unternehmensforschung 12 (1968) 258–268.
[9] P. Crescenzi, G. Gambosi, P. Penna, On-line algorithms for the channel assignment problem in cellular networks, Discrete Applied Mathematics 137 (3) (2004) 237–266. Extended abstract in: Proc. of ACM DIALM'00, International Workshop on Discrete Algorithms and Methods for Mobile Computing.

[10] R. Graham, Bounds for certain multiprocessor anomalies, Bell System Technical Journal 45 (1966) 1563–1581.
[11] R. Graham, Bounds on multiprocessor timing anomalies, SIAM Journal on Applied Mathematics 17 (1969) 263–269.
[12] E. Tardòs, J.K. Lenstra, D.B. Shmoys, Approximation algorithms for scheduling unrelated parallel machines, Mathematical Programming 46 (1990) 259–271.
[13] J.D. Murchland, Braess's paradox of traffic flow, Transportation Research 4 (1970) 391–394.
[14] S. Phillips, J. Westbrook, Online load balancing and network flow, Algorithmica 21 (3) (1998) 245–261.
[15] S.Y. Ma, A. Plotkin, An improved lower bound for load balancing of tasks with unknown duration, Information Processing Letters 62 (6) (1997) 301–303.