Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 695–714
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

A revisit of fast greedy heuristics for mapping a class of independent tasks onto heterogeneous computing systems

Ping Luo[a,*], Kevin Lü[b], Zhongzhi Shi[a]

[a] *Key Laboratary of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, No. 6 Kexueyuan Nanlu, Beijing 100080, China*
[b] *Brunel University, Uxbridge UB8 3PH, UK*

Received 3 March 2006; received in revised form 27 October 2006; accepted 7 March 2007
Available online 15 March 2007

Abstract

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links, to perform groups of computing-intensive applications that have diverse computational requirements and constraints. The problem of optimally mapping a class of independent tasks onto the machines of an HC environment has been proved, in general, to be NP-complete, thus requiring the development of heuristic techniques for practical usage. If the mapping has real-time requirements such that the mapping process is performed during task execution, *fast greedy* heuristics must be adopted. This paper investigates *fast greedy* heuristics for this problem and identifies the importance of the concept of *task consistency* in designing this mapping heuristic. We further propose *task priority graph* based *fast greedy* heuristics, which consider the factors of both *task consistency* and *machine consistency* (the same concept of *consistency* as in previous studies). A collection of 20 greedy heuristics, including 17 newly proposed ones, are implemented, analyzed, and systematically compared within a uniform model of task execution time. This model is implemented by the coefficient-of-variation based method. The experimental results illuminate the circumstances when a specific greedy heuristic would outperform the other 19 greedy heuristics.

Keywords: Heterogeneous computing; Independent tasks; Greedy heuristic; Task consistency; Task priority graph

1. Introduction

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links, to perform groups of computing-intensive applications that have diverse computational requirements and constraints [2]. The mapping heuristic, responsible for optimally mapping these tasks onto the machines of a distributed HC environment, is closely related to the performance of this computing system and has drawn a great deal of attention [2,14–17,7,11]. Mapping heuristics are also important in fields, such as *computational grids* [5] and *parallel program scheduling* [9,10].

According to the type of tasks being mapped, the heuristics can be classified into two types: mapping a class of independent (non-communicating) jobs [2,14,15,17] and mapping a directed acyclic graph (DAG) composed of communicating jobs [7,16,11]. The former type of mapping heuristics can be used in the latter mapping problems [16,12], and are the research focus of this paper. The goal of mapping a set of independent jobs onto HC systems is to minimize the time until the last job finishes (i.e. the makespan). The formal definition of this problem is given in the following subsection.

1.1. Problem definition

Provided the independent tasks vector $T = (t_1, \ldots, t_p)$, the heterogeneous machines vector $M = (m_1, \ldots, m_q)$, and the *ETC* (expected time to compute) matrix $E = [e_{ij}]_{p\times q}$, in which

* Corresponding author.
E-mail address: luop@ics.ict.ac.cn (P. Luo).

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.03.003

e_{ij} represents the expected execution time of task t_i on machine m_j.[1] It is also assumed that each machine executes a single task at a time. The problem is to find a task mapping scheme which minimizes the makespan of all the tasks in T.

More formally, let $\pi = \{T_1, \ldots, T_q\}$ be a q-element partition (a set of ordered objects at this time) of T, representing a mapping scheme where T_j is the set of tasks assigned to host j, for $1 \leqslant j \leqslant q$. The makespan of π is

$$ct(\pi) = \max_{1 \leqslant j \leqslant q} l_j$$

where $l_j = \sum_{i \in T_j} e_{ij}$ is the completion time of all the tasks on host j. The objective is to find the best mapping scheme $\pi_{\min}$ with the minimal makespan, such that

$$ct(\pi_{\min}) = \min_{\pi \in q\text{-}PART(T)} ct(\pi),$$

where q-$PART(T)$ is the set of all q-element partitions of T.

1.2. The state-of-the-art of this problem

This is a problem of combinatorial optimization. It is clear that $|q\text{-}PART(T)| = q^p$ when $|T| = p$ and $|M| = q$. To obtain the optimal solution, all the elements in q-$PART(T)$ must be checked. The general problem of optimally mapping independent tasks to machines in a HC suite has been shown to be (weakly) NP-complete [4,6]. To address this problem, a number of heuristics have been proposed and can be categorized into *fast* and *slow* algorithms according to the time it takes to obtain the sub-optimal solution. *Slow* heuristics, such as by *ant optimization* [15] and by *genetic algorithm* [19], take a significantly longer time than *fast* heuristics [2], however, they aim to find better solutions.

In [2] 11 heuristics are compared and it is concluded that the *fast greedy* heuristic *min–min* performs well in comparison to the other techniques. Paper [15] reports that the technique of *ant optimization* outperforms *min–min* and *genetic algorithm* at the expense of a much longer mapping process. Some generic post-optimization techniques have also been proposed [14,16] to perform a local search around a mapping scheme obtained by existing heuristics. However, only *fast greedy* heuristics can be adopted in the following situations, where the mapping process is performed during the execution of the mapped tasks.

- When an HC system is constructed as an application service provider to respond to online computational requests, the waiting tasks can be scheduled in a batch mode in order to increase the system *utilization ratio*. They are not mapped onto the machines once they arrive; instead they are collected into a meta-task [13] that is examined for mapping at pre-specified time intervals. In this situation, although a time-consuming heuristic can achieve a shorter mapping makespan, the much longer mapping process would postpone the eventual task completion-time, which must be forbidden if these tasks have real-time constraints. Thus, this problem can only be addressed by effective *fast greedy* mapping heuristics.
- The DAG mapping problem in HC systems can be based upon mapping multiple groups of independent tasks [16,12]. The mapping process is performed during the execution of the early mapped tasks. Thus, *fast greedy* heuristics are also the *only* choice in this situation.

In this paper we investigate *fast greedy* heuristics for mapping a class of independent tasks onto HC systems. We introduce the concept of *task consistency* and find that this trait is significant in distinguishing the full functions of different mapping heuristics. A collection of 20 *fast greedy* heuristics, most of which are proposed in this paper, are analyzed and compared systematically within a uniform model of task execution time. This model is implemented by the *coefficient-of-variation* (COV) based method [1]. In this model of task execution time, *task consistency* in addition to the other three previous parameters of *machine consistency* (the same concept of *consistency* in [2]), *task heterogeneity* and *machine heterogeneity* are used to characterize different HC environments and computational tasks. The main experimental result is that one of the proposed *fast greedy* heuristics TPD↓–minCT–minCT with the support of a *task priority graph* outperforms the other heuristics, including *min–min*, in most situations.

The remainder of this paper is organized as follows. Section 2 provides some basic definitions, including the new concept of *task consistency*, and the motivation for the new greedy heuristic proposed in this paper. Section 3 describes 20 greedy heuristics, which can be categorized into two classes: with and without the support of a *task priority graph*. The task execution time model with a new parameter of *task consistency* is presented in Section 4. Section 5 lists and analyzes the experimental results of all the 20 greedy heuristics in all cases of different parameters for the task execution time model. The conclusions are given in Section 6.

2. Basic definitions and motivation

2.1. Task consistency in execution time model for HC

To the best of our knowledge, all previous studies in this area have only considered *machine consistency* of an ETC. The factor of *task consistency*, as we define it in the following, has not been considered, and neither has the impact of how this factor would affect the performance of *fast greedy* heuristics been investigated.

Definition 2.1 (*Machine consistency*). Assume the independent tasks vector be $T = (t_1, \ldots, t_p)$, the heterogeneous machines vector be $M = (m_1, \ldots, m_q)$, the *ETC* matrix be $E = [e_{ij}]_{p \times q}$. The machine consistency relationship between two machines is described as a partial order $\langle M, \preccurlyeq' \rangle$ such that

$$m_k \preccurlyeq' m_l \quad \text{iff } e_{ik} \geqslant e_{il} \quad \text{for } 1 \leqslant i \leqslant p. \tag{1}$$

Definition 2.2 (*Task consistency*). Assume the independent tasks vector be $T = (t_1, \ldots, t_p)$, the heterogeneous machines

[1] How to get this *ETC* matrix is within another research field, i.e. execution time estimation model for tasks [8].

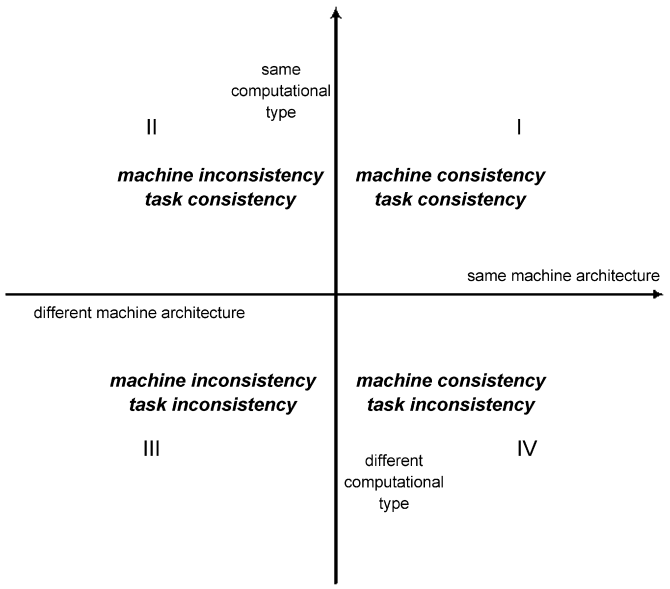

Fig. 1. Different types of consistency in an ETC.

vector be $M = (m_1, \ldots, m_q)$, the *ETC* matrix be $E = [e_{ij}]_{p\times q}$. The task consistency relationship between two tasks is described as the following partial order $\langle T, \preccurlyeq\rangle$ such that

$$t_k \preccurlyeq t_l \quad \text{iff } e_{kj} \leqslant e_{lj} \quad \text{for } 1 \leqslant j \leqslant q. \tag{2}$$

The concepts behind *machine consistency* and *task consistency* are *machine speed* and *task effort*, respectively. $m_k \preccurlyeq' m_l$ means that the *speed* of machine l is faster than that of machine k when executing the tasks in the tasks vector, and $t_k \preccurlyeq t_l$ means that the *task effort* of t_l is always bigger than that of t_k on each machine in the HC suite. Ideally, the concept of *machine speed* is constant regardless of which task it executes while the concept of *task effort* is constant regardless of which machine the task is executed on.

Task consistency occurs widely in practice when the computational types of two tasks are identical. Task computational types include readily parallelizable tasks, difficult to parallelize tasks, tasks that are floating point intensive, simple text formatting tasks, etc. [1]. If the runtime of a task is shorter than that of another task of the same computational type, the *task effort* of the former task is less than that of the latter. Thus the former task has a shorter execution time on any machine in the HC suite. In this situation, *task consistency* occurs. On the other hand, *machine consistency* occurs when two hosts in the HC suite have the same machine architecture [1]. Fig. 1 describes the different types of consistency in an ETC. When the machine architectures are the same (quadrant I and IV), *machine consistency* exists. When the computational type of tasks is the same (quadrant I and II), *task consistency* exists.

Paper [2] concludes that the relatively simple *min–min* heuristic performs well in comparison to the other techniques. However, this conclusion was drawn without considering the factor of *task consistency*. The ETCs used in the experiments of [2] are actually from quadrant III and IV in Fig. 1. In this paper, we propose the fast greedy heuristic, which is expected to perform better than *min–min* for ETCs from all quadrants.

2.2. Motivation for HC

For mapping a class of independent tasks onto heterogeneous systems, it is clear that a solution is optimal if the completion time is the same on each machine. This means that by trying to obtain a solution that is very balanced, we end up with a solution with a rather good makespan.

Only considering the factor of *machine consistency*, *min–min* firstly maps the task with the minimal completion time from among all unmapped tasks. Thus, it prefers to map the tasks with a smaller *task effort* earlier than the tasks with a bigger *task effort*. However, if the tasks with a bigger *task effort* are mapped at the end of the scheduling process, it is more likely to enlarge the imbalance of loads among hosts and deviate from the load balance motivation, and thus to generate a worse solution. Therefore, the *fast greedy* mapping heuristic must consider the factors of both *machine consistency* and *task consistency*. For two tasks **without** a *task consistency* relationship, the one with the earlier completion time should be mapped first. For two tasks **with** a *task consistency* relationship, the one with the bigger *task effort* must be mapped in advance. The combination of the above two mapping criteria forms the proposed mapping heuristic, which outperforms *min–min*.

2.3. The other basic definitions

The following definitions will also be used in this paper.

Definition 2.3 (*Cover relation*). Let x and y be two different elements of a partial order $\langle P, \preccurlyeq\rangle$. Then x covers y, or y is covered by x, provided that $x \succcurlyeq y$ and for any $z \succcurlyeq y$, $x \not\succcurlyeq z$. In this situation x is called an upper cover of y while y is called a lower cover of x.

Definition 2.4 (*Hasse diagram*). If $\langle P, \preccurlyeq\rangle$ is a finite partial order, then it can be represented by a Hasse diagram, which is a DAG whose vertices are elements of P and whose edges correspond to the corresponding cover relation.

Definition 2.5 (*Minimal/maximal element*). Let $\langle P, \preccurlyeq\rangle$ be a partial order, then an element $b \in P$ is a minimal element of P if there is no element $a \in P$ that satisfies $a \preccurlyeq b$. Similarly, an element $b \in P$ is a maximal element of P if there is no element $a \in P$ that satisfies $b \preccurlyeq a$.

Definition 2.6 (*Degree of task consistency*). Let the independent tasks vector be $T = (t_1, \ldots, t_p)$, the heterogeneous machines vector be $M = (m_1, \ldots, m_q)$, the *ETC* matrix be $E = [e_{ij}]_{p\times q}$. $\langle T, \preccurlyeq\rangle$ is the partial order of task consistency in E. The degree of task consistency in E is defined as

$$d_{\text{task}}(E) = \frac{|\{(t_i, t_j) | t_i \preccurlyeq t_j \text{ or } t_j \preccurlyeq t_i, t_i \in T, t_j \in T\}|}{|T|(|T| - 1)}.$$

Table 1
ETC for the running example

Task no.	m_0	m_1	m_2	m_3
t_0	2000.0	2200.0	2400.0	2600.0
t_1	900.0	1100.0	1300.0	1500.0
t_2	950.0	1050.0	1350.0	1550.0
t_3	980.0	1080.0	1330.0	1480.0
t_4	600.0	700.0	800.0	1490.0
t_5	550.0	1070.0	750.0	1000.0
t_6	920.0	600.0	600.0	1000.0
t_7	100.0	200.0	300.0	400.0

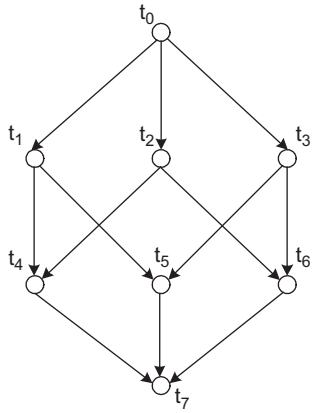

Fig. 2. Task priority diagram for the ETC in Table 1.

If $d_{\text{task}}(E) = 1$, $\langle T, \preccurlyeq \rangle$ is a linear order. If $d_{\text{task}}(E) = 0$, the Hasse diagram of $\langle T, \preccurlyeq \rangle$ is $|T|$ unconnected vertexes.

Definition 2.7 (*Degree of machine consistency*). Let the independent tasks vector be $T = (t_1, \ldots, t_p)$, the heterogeneous machines vector be $M = (m_1, \ldots, m_q)$, the *ETC* matrix be $E = [e_{ij}]_{p\times q}$. $\langle M, \preccurlyeq' \rangle$ is the partial order of machine consistency in E. The degree of machine consistency in E is defined as

$$d_{\text{machine}}(E) = \frac{|\{(m_i, m_j) | m_i \preccurlyeq' m_j \text{ or } m_j \preccurlyeq' m_i, m_i \in M, m_j \in M\}|}{|M|(|M|-1)}.$$

The ETC shown in Table 1 is used as a running example in this paper. In this example, the independent tasks vector is $T = \{t_0, \ldots, t_7\}$, the heterogeneous machines vector is $M = (m_0, \ldots, m_3)$, and the ETC matrix is $E = [e_{ij}]_{8\times 4}$. Based on the definition of *task consistency* in Section 2.1, it is straightforward to construct the Hasse diagram of $\langle T, \preccurlyeq \rangle$ as depicted in Fig. 2. This graph, in fact, is a TPD, which contains the information about the mapping sequence of tasks. For the *min–min* heuristic, the element in the lower level of the diagram always has higher priority for mapping, and an element cannot be mapped until all its predecessors have been mapped.

3. Mapping a class of independent tasks with and without TPD

In this section, we present all 20 greedy heuristics with and without TPD for mapping a class of independent tasks and identify the relationship amongst them.

3.1. Comparing optional mapping steps

Let the unmapped tasks set be $T' = (t_1, \ldots, t_l)$, the current machine free-time vector be $F = (f_1, \ldots, f_q)$, which means that host i is free after f_i time-units for $1 \leqslant i \leqslant q$. The standard deviation of F is $sd(F) = \sqrt{\frac{\sum_{i=1}^{q}(A-f_i)^2}{q-1}}$, where $A = \frac{\sum_{i=1}^{q} f_i}{q}$. The machine free-time vector after update is $F_{ij} = (f_1, \ldots, f_j + e_{ij}, \ldots, f_q)$, representing a mapping step where task i is mapped to host j. Given $F_{i_1 j_1}$ and $F_{i_2 j_2}$, the problem is to specify which mapping step will be mapped first. Different algorithms give different answers as shown in Sub-algorithms 3.1–3.3.

minCT in Sub-algorithm 3.1 prefers the mapping step with the minimal completion time for the task, while minSD in Sub-algorithm 3.3 prefers the mapping step which minimizes the standard deviation of the system loads after update. maxCT in Sub-algorithm 3.2 is opposite to minCT. All these sub-algorithms are easily extended to the functions of a set of optional mapping steps, each of which outputs the mapping step with the best corresponding heuristic measure. They are described as functions of two mapping steps only for conciseness in writing.

Sub-algorithm 3.1 minCT($F_{i_1 j_1}$, $F_{i_2 j_2}$)

Input: $T' = (t_1, \ldots, t_l)$, $F = (f_1, \ldots, f_q)$, $F_{i_1 j_1} = (f_1, \ldots, f_{j_1} + e_{i_1 j_1}, \ldots, f_q)$, $F_{i_2 j_2} = (f_1, \ldots, f_{j_2} + e_{i_2 j_2}, \ldots, f_q)$

1: $ct_1 = f_{j_1} + e_{i_1 j_1}$
2: $ct_2 = f_{j_2} + e_{i_2 j_2}$
3: **if** $ct_1 > ct_2$ **then**
4: **return** $F_{i_2 j_2}$
5: **else**
6: **return** $F_{i_1 j_1}$
7: **end if**

Sub-algorithm 3.2 maxCT($F_{i_1 j_1}$, $F_{i_2 j_2}$)

Input: $T' = (t_1, \ldots, t_l)$, $F = (f_1, \ldots, f_q)$, $F_{i_1 j_1} = (f_1, \ldots, f_{j_1} + e_{i_1 j_1}, \ldots, f_q)$, $F_{i_2 j_2} = (f_1, \ldots, f_{j_2} + e_{i_2 j_2}, \ldots, f_q)$

1: $ct_1 = f_{j_1} + e_{i_1 j_1}$
2: $ct_2 = f_{j_2} + e_{i_2 j_2}$
3: **if** $ct_1 < ct_2$ **then**
4: **return** $F_{i_2 j_2}$
5: **else**
6: **return** $F_{i_1 j_1}$
7: **end if**

Sub-algorithm 3.3 minSD($F_{i_1j_1}$, $F_{i_2j_2}$)

Input: $T' = (t_1, \ldots, t_l)$, $F = (f_1, \ldots, f_q)$, $F_{i_1j_1} = (f_1, \ldots, f_{j_1} + e_{i_1j_1}, \ldots, f_q)$, $F_{i_2j_2} = (f_1, \ldots, f_{j_2} + e_{i_2j_2}, \ldots, f_q)$

1: **if** $sd(F_{i_1j_1}) > sd(F_{i_2j_2})$ **then**
2: **return** $F_{i_2j_2}$
3: **else**
4: **return** $F_{i_1j_1}$
5: **end if**

The proposed Sub-algorithm *minSD* in 3.3 only considers the relative values between sd_1 and sd_2, where sd_1=$sd(F_{i_1j_1})$ and sd_2=$sd(F_{i_2j_2})$. Thus, many of the calculations in Sub-algorithm 3.3 can be erased by the following analytical analysis.

When $j_1 \neq j_2$, let $A = \sum_{i=1}^{q} f_i$

$$(q-1)sd_1^2 = \left[\sum_{i\neq j_1,j_2}\left(f_i - \frac{A+e_{i_1j_1}}{q}\right)^2\right] + \left(f_{j_1} + e_{i_1j_1} - \frac{A+e_{i_1j_1}}{q}\right)^2 + \left(f_{j_2} - \frac{A+e_{i_1j_1}}{q}\right)^2,$$

$$(q-1)sd_2^2 = \left[\sum_{i\neq j_1,j_2}\left(f_i - \frac{A+e_{i_2j_2}}{q}\right)^2\right] + \left(f_{j_1} - \frac{A+e_{i_2j_2}}{q}\right)^2 + \left(f_{j_2} + e_{i_2j_2} - \frac{A+e_{i_2j_2}}{q}\right)^2,$$

$$\begin{aligned}(q-1)sd_1^2 - (q-1)sd_2^2 &= 2q^2e_{i_1j_1}f_{j_1} - 2q^2e_{i_2j_2}f_{j_2} + 2qe_{i_2j_2}A \\ &\quad -2qe_{i_1j_1}A + q(q-1)e_{i_1j_1}^2 - q(q-1)e_{i_2j_2}^2.\end{aligned}$$

Then,

$$sd_1 - sd_2 = \begin{cases} > 0 & \text{if } 2qe_{i_1j_1}f_{j_1} + 2e_{i_2j_2}A + (q-1)e_{i_1j_1}^2 \\ & \quad > 2qe_{i_2j_2}f_{j_2} + 2e_{i_1j_1}A + (q-1)e_{i_2j_2}^2, \\ \leqslant 0 & \text{otherwise.}\end{cases}$$

When $j_1 = j_2 = j$, let $A = \sum_{i=1}^{q} f_i$

$$\begin{aligned}(q-1)sd_1^2 &= \sum_{i\neq j}\left(f_i - \frac{A+e_{i_1j}}{q}\right)^2 + \left(f_j + e_{i_1j} - \frac{A+e_{i_1j}}{q}\right)^2 \\ &= \frac{(q-1)}{q}e_{i_1j}^2 + \frac{2(qf_j - A)}{q}e_{i_1j} + \left(\sum_{i=1}^{q} f_i^2 - \frac{A^2}{q}\right).\end{aligned}$$

It is clear that at this time $(q-1)sd_1^2$ is a quadratic function of e_{i_1j} with the axis of symmetry $e_{i_1j} = \frac{A-qf_j}{q-1}$, where this function reaches its minimum. Thus, given sd_1 and sd_2,

if $A - qf_j \leqslant 0$

$$sd_1 - sd_2 = \begin{cases} > 0 & \text{if } e_{i_1j} > e_{i_2j}, \\ \leqslant 0 & \text{otherwise.}\end{cases}$$

if $A - qf_j > 0$

$$sd_1 - sd_2 = \begin{cases} > 0 & \text{if } \left|\frac{A-qf_j}{q-1} - e_{i_1j}\right| > \left|\frac{A-qf_j}{q-1} - e_{i_2j}\right|, \\ \leqslant 0 & \text{otherwise.}\end{cases}$$

3.2. Mapping algorithms without TPD

Given the independent tasks vector $T = (t_1, \ldots, t_p)$, the heterogeneous machines vector $M = (m_1, \ldots, m_q)$, and the *ETC* matrix $E = [e_{ij}]_{p\times q}$, the general form of the greedy scheduling algorithm without TPD is described in Algorithm 3.12. In Algorithm 3.12 the mapping iteration repeats until all the tasks have been mapped. In each round of this iteration only one task is selected to be mapped onto a suitable machine.

The difference among the various mapping algorithms is the different sub-algorithm First-Mapping-Step (in Sub-algorithms 3.4–3.9) used. In First-Mapping-Step two metrics are used. For each task the first metric is to select the best machine onto which the task can be mapped. (Thus, maxCT cannot be used as the first metric.) Based on this result, the second metric is used to select the best task with its corresponding destination machine. Hence, new heuristics are in fact designed by combining the metrics, i.e. minCT, minSD and maxCT, in the two steps of the First-Mapping-Step algorithm. The possible combinations include minCT–minCT, minCT–minSD, minCT–maxCT, minSD–minCT, minSD–minSD and minSD–maxCT, described in Sub-algorithms 3.4 through 3.9.

Algorithms 3.10 and 3.11 are two other more simple mapping algorithms without TPD, in which each task is assigned in arbitrary order to the machine according to minCT and minSD, respectively.

Sub-algorithm 3.4 First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, minCT–minCT)

1: $\vec{F} = \{F_{ij'}|F_{ij'} = \text{minCT}(\{F_{ij}|1 \leqslant j \leqslant q\}), t_i \in T\}$
2: $F_{ij} = \text{minCT}(\vec{F})$
3: **return** F_{ij}

Sub-algorithm 3.5 First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, minCT–minSD)

1: $\vec{F} = \{F_{ij'}|F_{ij'} = \text{minCT}(\{F_{ij}|1 \leqslant j \leqslant q\}), t_i \in T\}$
2: $F_{ij} = \text{minSD}(\vec{F})$
3: **return** F_{ij}

Sub-algorithm 3.6 First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, minCT–maxCT)

1: $\vec{F} = \{F_{ij'}|F_{ij'} = \text{minCT}(\{F_{ij}|1 \leqslant j \leqslant q\}), t_i \in T\}$
2: $F_{ij} = \text{maxCT}(\vec{F})$
3: **return** F_{ij}

Sub-algorithm 3.7 First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, minSD–minCT)

1: $\vec{F} = \{F_{ij'}|F_{ij'} = \text{minSD}(\{F_{ij}|1 \leqslant j \leqslant q\}), t_i \in T\}$
2: $F_{ij} = \text{minCT}(\vec{F})$
3: **return** F_{ij}

Sub-algorithm 3.8 First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, minSD–minSD)

1: $\vec{F} = \{F_{ij'}|F_{ij'} = \text{minSD}(\{F_{ij}|1 \leqslant j \leqslant q\}), t_i \in T\}$
2: $F_{ij} = \text{minSD}(\vec{F})$
3: **return** F_{ij}

Sub-algorithm 3.9 First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, minSD–maxCT)

1: $\vec{F} = \{F_{ij'}|F_{ij'} = \text{minSD}(\{F_{ij}|1 \leqslant j \leqslant q\}), t_i \in T\}$
2: $F_{ij} = \text{maxCT}(\vec{F})$
3: **return** F_{ij}

Algorithm 3.10 simple-minCT: Mapping Algorithm without TPD

1: **while** $T \neq \Phi$ **do**
2: t_i is any task in T
3: $F_{ij'} = \text{minCT}(\{F_{ij}|1 \leqslant j \leqslant q\})$
4: map task t_i to machine $m_{j'}$ and update the load on $m_{j'}$
5: $T = T - \{t_i\}$
6: **end while**

Algorithm 3.11 simple-minSD: Mapping Algorithm without TPD

1: **while** $T \neq \Phi$ **do**
2: t_i is any task in T
3: $F_{ij'} = \text{minSD}(\{F_{ij}|1 \leqslant j \leqslant q\})$
4: map task t_i to machine $m_{j'}$ and update the load on $m_{j'}$
5: $T = T - \{t_i\}$
6: **end while**

Algorithm 3.12 NTPD-*heuristic*: Mapping Algorithm without TPD

1: **while** $T \neq \Phi$ **do**
2: $F_{i'j'}$ =First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, *heuristic*) {output the first mapping step according to the selected heuristic}
3: map task $t_{i'}$ to machine $m_{j'}$ and update the load on $m_{j'}$
4: $T = T - \{t_{i'}\}$
5: **end while**

3.3. Mapping algorithms with TPD

Algorithm 3.13 is the generic form of the mapping algorithm with TPD in the decreasing order of *task effort*. In this algorithm the cycle repeats until all the tasks have been mapped. In each round of the iteration, T is the set of all the maximal elements of the unmapped tasks in the current TPD. The First-Mapping-Step algorithm can then be used to select a task $t_{i'}$ from T and map it onto $m_{j'}$. After $t_{i'}$ is mapped, vertex $t_{i'}$ and its edges are removed from the TPD G, and the set T, containing the maximal elements of the new G, is also updated.

Algorithm 3.13 TPD↓-*heuristic*: Mapping Algorithm with TPD

1: G =Hasse-Diagram-Generator($E_{p\times q}$) {E is the input ETC matrix}
2: $T = \{t|t$ is a maximal element of $G\}$
3: **while** $T \neq \emptyset$ **do**
4: $F_{i'j'}$ =First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, *heuristic*)
5: map $t_{i'}$ to $m_{j'}$ and update the load on $m_{j'}$
6: delete $t_{i'}$ and the edges of $t_{i'}$ from G
7: $T = \{t|t$ is a maximal element of the new $G'\}$
8: **end while**

Algorithm 3.14 TPD↑-*heuristic*: Mapping Algorithm with TPD

1: G =Hasse-Diagram-Generator($E_{p\times q}$) {E is the input ETC matrix}
2: $T = \{t|t$ is a minimal element of $G\}$
3: **while** $T \neq \emptyset$ **do**
4: $F_{i'j'}$ =First-Mapping-Step($\{F_{ij}|t_i \in T, 1 \leqslant j \leqslant q\}$, *heuristic*)
5: map $t_{i'}$ to $m_{j'}$ and update the load on $m_{j'}$
6: delete $t_{i'}$ and the edges of $t_{i'}$ from G
7: $T = \{t|t$ is a minimal element of the new $G\}$
8: **end while**

Property 3.1. *The operation in line* 7 *of algorithm* 3.13 *is equivalent to* (3)

$$T = (T - \{t_{i'}\}) \cup \{t|t \text{ is covered only by } t_{i'} \text{ in } G\}. \quad (3)$$

Proof. Let t be an element, which is *only* covered by $t_{i'}$ in G and there is not any other element covering t in G. It is clear that t turns into the new maximal element in the new G. The maximal elements, except $t_{i'}$ in G, remain maximal in the new G. It follows the conclusion. □

This property simplifies the process of finding all the maximal elements in the new G.

Fig. 3 depicts the execution procedure of TPD↓–minCT–minCT (Algorithm 3.13 with minCT–minCT) for the running example in Section 2.3. Each round of the iteration in this algorithm is described by a sub-figure, where the current TPD, the set T and the mapping step generated by minCT–minCT for

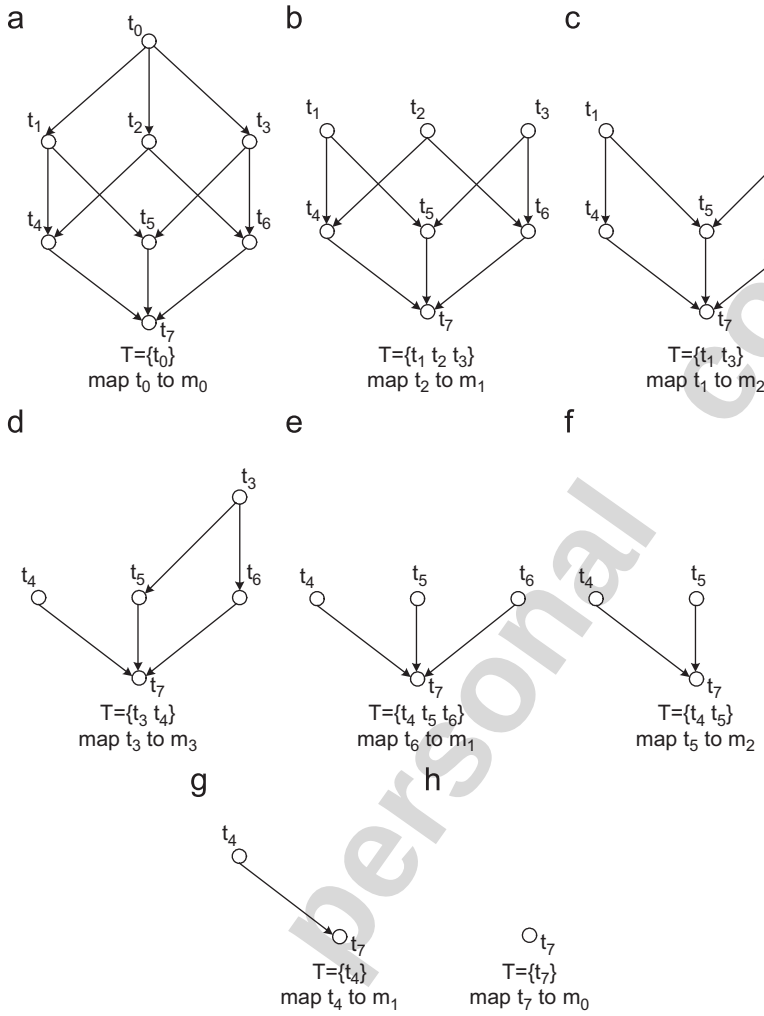

Fig. 3. Execution process of Algorithm 3.13 for the running example. (a) The first iteration, (b) the second iteration, (c) the third iteration, (d) the fourth iteration, (e) the fifth iteration, (f) the sixth iteration, (g) the seventh iteration, (h) the eighth iteration.

this round are listed. The two mapping schemes generated by NTPD–minCT–minCT (Algorithm 3.12 with minCT–minCT) and TPD↓–minCT–minCT are listed in Fig. 4. The makespans of these two mapping schemes generated by NTPD–minCT–minCT and TPD↓–minCT–minCT are 3200 and 2350, respectively. This gives a typical example to show that TPD↓–minCT–minCT performs better than NTPD–minCT–minCT when *task consistency* exists in the input ETC.

Algorithm 3.14 is the mapping algorithm with TPD in the increasing order of *task effort*, which is different from Algorithm 3.13 in that the mapping set T contains all the minimal elements of the diagram in each cycle.

3.4. Summary of greedy heuristics

Table 2 gives a summary of 20 greedy mapping algorithms with and without the support of TPD. Different heuristics used in Algorithms 3.12–3.14 form different mapping algorithms. H_1, H_3 and H_5 are actually the algorithms of *mct*, *min–min* and *max–min* in [2], respectively. All the other algorithms are newly proposed in this paper.

The following properties identify the relationships amongst these greedy algorithms. Any two algorithms are referred to as equivalent to each other if they output the same result for any given input.

Property 3.2. *NTPD-heuristic*, *TPD↓-heuristic*, *TPD↑-heuristic are equivalent to each other for any ETC E, such that* $d_{\text{task}}(E) = 0$.

Proof. $d_{\text{task}}(E) = 0$ means that there does not exist *task consistency* relationship between any two tasks in E. Thus, each task is either a maximal or a minimal element in the TPD. The set T in Algorithms 3.13 and 3.14 always contains all unmapped tasks. It follows the conclusion. □

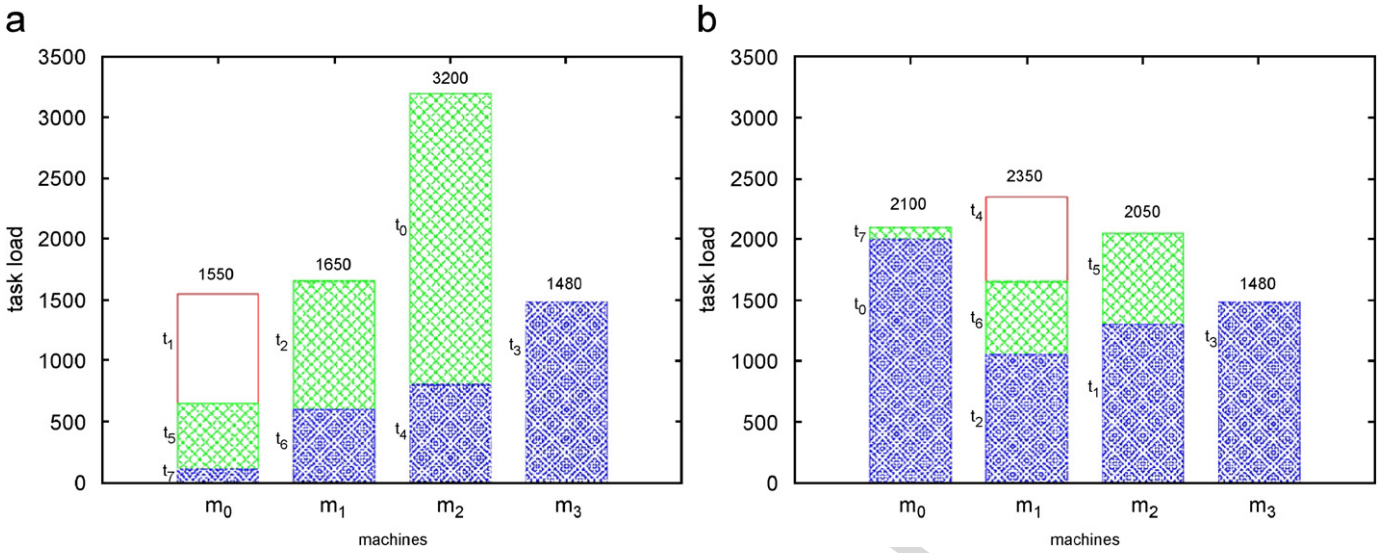

Fig. 4. The mapping schemes for the running example. (a) The mapping scheme generated from NTPD–minCT–minCT. (b) The mapping scheme generated from TPD↓–minCT–minCT.

Table 2
Summary of greedy heuristics for mapping a class of independent tasks

No.	Name	Components	Reference
H_1	simple-minCT	Algorithm 3.10	mct in [2]
H_2	simple-minSD	Algorithm 3.11	New
H_3	NTPD–minCT–minCT	Algorithm 3.12, Sub-algorithm 3.4	min–min in [2]
H_4	NTPD–minCT–minSD	Algorithm 3.12, Sub-algorithm 3.5	New
H_5	NTPD–minCT–maxCT	Algorithm 3.12, Sub-algorithm 3.6	max–min in [2]
H_6	NTPD–minSD–minCT	Algorithm 3.12, Sub-algorithm 3.7	New
H_7	NTPD–minSD–minSD	Algorithm 3.12, Sub-algorithm 3.8	New
H_8	NTPD–minSD–maxCT	Algorithm 3.12, Sub-algorithm 3.9	New
H_9	TPD↑–minCT–minCT	Algorithm 3.14, Sub-algorithm 3.4	New
H_{10}	TPD↑–minCT–minSD	Algorithm 3.14, Sub-algorithm 3.5	New
H_{11}	TPD↑–minCT–maxCT	Algorithm 3.14, Sub-algorithm 3.6	New
H_{12}	TPD↑–minSD–minCT	Algorithm 3.14, Sub-algorithm 3.7	New
H_{13}	TPD↑–minSD–minSD	Algorithm 3.14, Sub-algorithm 3.8	New
H_{14}	TPD↑–minSD–maxCT	Algorithm 3.14, Sub-algorithm 3.9	New
H_{15}	TPD↓–minCT–minCT	Algorithm 3.13, Sub-algorithm 3.4	New
H_{16}	TPD↓–minCT–minSD	Algorithm 3.13, Sub-algorithm 3.5	New
H_{17}	TPD↓–minCT–maxCT	Algorithm 3.13, Sub-algorithm 3.6	New
H_{18}	TPD↓–minSD–minCT	Algorithm 3.13, Sub-algorithm 3.7	New
H_{19}	TPD↓–minSD–minSD	Algorithm 3.13, Sub-algorithm 3.8	New
H_{20}	TPD↓–minSD–maxCT	Algorithm 3.13, Sub-algorithm 3.9	New

Property 3.2 shows that TPD↓-*heuristic* and TPD↑-*heuristic* have the same performance as NTPD-*heuristic* when no *task consistency* exists between any two of the given tasks.

Property 3.3. *H_5 and H_{17} are equivalent to each other.*

Proof. Let T be the set of all maximal elements in a TPD, and T' be the set of all unmapped tasks. It is clear that $T \subseteq T'$. Suppose any task $t' \in T'$ and $t' \notin T$, there must exist a task $t \in T$ such that $t' \preccurlyeq t$. This shows that the completion time of t' on any machine is earlier than that of t. It is impossible that t' is picked out by line 2 in Algorithm 3.12 when the heuristic is minCT–maxCT. Thus, if task t'' is selected by line 2 Algorithm 3.12, $t'' \in T$. It follows the conclusion. □

Property 3.3 shows that H_5 is only equivalent to H_{17}. It is not generically equivalent to the other TPD↓-*heuristic* algorithms. However, under certain conditions H_5 can be equivalent to H_{15}, as shown in Property 3.4.

Property 3.4. *H_5 and H_{15} are equivalent to each other for any ETC E, such that $d_{\text{task}}(E) = 1$.*

Table 3
Computational complexity of greedy mapping heuristics

Algorithm name	Computational complexity
simple-minCT	$O(pq)$
simple-minSD	$O(pq)$
NTPD-*heuristic*	$O(p^2q)$
TPD↓-*heuristic*	$O(p^2q)$
TPD↑-*heuristic*	$O(p^2q)$

Proof. Trivial. □

Properties 3.2 and 3.4 show that with task full-consistency ($d_{\text{task}}(E) = 1$) H_{15} is equivalent to H_5, while with task non-consistency ($d_{\text{task}}(E) = 0$) H_{15} is equivalent to H_3. These formal properties coincide with the intuitive view that H_{15} is a tradeoff between H_5 and H_3.

Property 3.5. *H_3 and H_9 are equivalent.*

Proof. The proof of this property is similar to the proof of Property 3.3, and thus it is omitted due to space limitations. □

Given an ETC matrix $E = [e_{ij}]_{p\times q}$, the computational complexities for all types of greedy heuristics are listed in Table 3. NTPD-*heuristic*, TPD↓-*heuristic* and TPD↑-*heuristic* have the same computational complexity because the complexity of constructing a TPD is also $O(p^2q)$.

4. Task execution time modelling for HC systems

In order to allow the study of the relative performance of different mapping heuristics under different circumstances, the ETC model is needed to simulate different HC environments by changing the parameters of this model. The ETC model used in this study has four parameters: *machine heterogeneity*, *task heterogeneity*, *machine consistency* and *task consistency*. The variation along a row is referred to as the *machine heterogeneity*; this is the degree to which the machine execution times vary for a given task. A system's machine heterogeneity is based on a combination of the machine heterogeneities for all tasks (rows). Similarly, the variation along a column of an ETC matrix is referred to as the *task heterogeneity*; this is the degree to which the task execution times vary for a given machine. A system's *task heterogeneity* is based on a combination of the task heterogeneities for all machines (columns). *Task consistency* and *machine consistency* are measured by d_{task} and d_{machine} defined in Section 2.3, respectively.

The COV-based ETC generation method [1] is adopted in our experiments. This method provides greater control over spread of the execution time values than the range-based method used widely previously [2,14,15,17]. Let σ and μ be the standard deviation and mean, respectively, of a set of execution times. $V = \frac{\sigma}{\mu}$, the COV, is adopted to measure the degree of heterogeneity.

In this method, μ_{task}, V_{task}, V_{machine}, representing the mean of task execution time, the COV of tasks, and the COV of machines, respectively, are used to generate an initial ETC, which satisfies the requirements of heterogeneity. In this process the *gamma* distribution is used to closely represent the distribution of task execution times. (In our experiments the algorithm in [3], implemented by Java package [18], is adopted to sample values from a given *gamma* distribution.) Secondly, by sorting the values of selected entries in each row and each column, the ETC will then satisfy the requirements of *machine consistency* and *task consistency*, respectively. If the sorting is performed on all values in each row or column, *full-consistency* will be achieved. If the sorting is performed on half of the values in each row or column, *partial-consistency* will be obtained. An ETC without sorting is referred to as *original-consistency*.

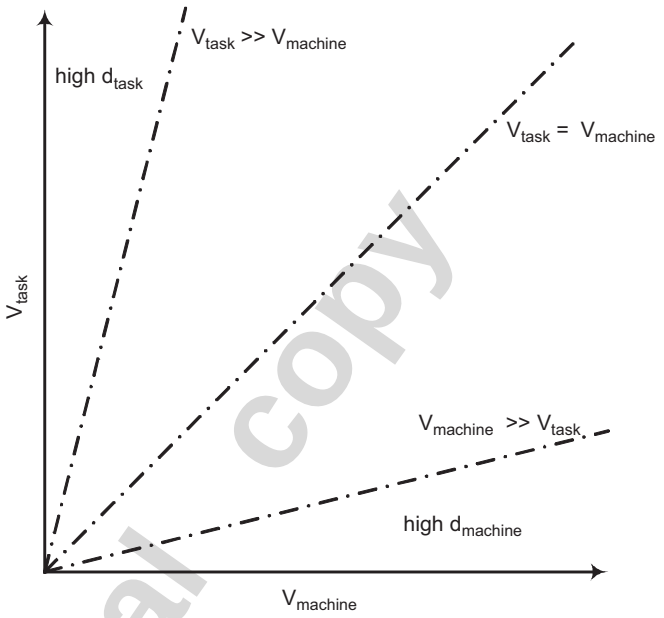

Fig. 5. Heterogeneity and consistency.

It should be noted that *heterogeneity* and *consistency* are two concepts closely related to each other. If $V_{\text{machine}} \gg V_{\text{task}}$, representing the *machine heterogeneity* is much bigger than the *task heterogeneity*, the machines may be much different from each other in terms of their computing speeds so that the differences in the computational requirements of the tasks would not affect the relative order of execution times for a given task on those machines. Thus, in this situation the generated ETC without sorting must have a high degree of *machine consistency*. Similarly, if $V_{\text{task}} \gg V_{\text{machine}}$ the generated ETC without sorting must have a high degree of *task consistency*. Additionally, given a fixed value of V_{machine}, d_{task} increases with the increase of V_{task}. Given a fixed value of V_{task}, d_{machine} increases with the increase of V_{machine}. This relationship is depicted in Fig. 5.

The ETCs generated from the COV-based method hold the aforementioned properties, however, the ETCs resulting from the range-based method do not hold them. This is the other reason why we use the COV-based generating method for ETCs in our experiments.

5. Experimental results

A purpose-built simulation software tool has been developed in this study to demonstrate and evaluate the greedy heuristics summarized in Table 2. These heuristics are applied to the ETCs described in Section 4 for performance assessment. The software allows users to specify the size and type of an ETC, and to choose which heuristics to execute. It then generates the specified ETCs, executes the desired heuristics, and displays the results. All the results discussed in this section were generated using this software.

The most important issue in performance measurement for comparing these mapping heuristics is *makespan*. Thus, the performance of different heuristics is measured by the average makespan of the ETCs with the specified type. For each heuristic and each type of ETC, the results are averaged over 1000 different ETCs of the same type. In all the experiments, the size of all ETCs is 512×16, the mean of task execution time μ_{task} is 1000, and the task COV V_{task} is in [0.1, 1.1] while the machine COV V_{machine} is in $[0.1, 0.6]$. The heterogeneous ranges were chosen to reflect the fact that in real situations there is more variability across execution times for different tasks on a given machine than the execution time for a single task across different machines.

The range bar for the average makespan of each heuristic shows a 95% confidence interval for the corresponding average makespan. This interval represents the likelihood that makespans of mappings for that type of heuristic fall within the specified range. That is, if another ETC matrix (of the same type) is generated, and the specified heuristic generates a mapping, then the makespan of the mapping would be within the given interval with 95% certainty.

The following two metrics are also recorded in the comparison of the heuristics.

- The *number of best solutions* (denoted by NB) is the number of times a particular method was the only one that produced the shortest makespan.

Table 4
NB and NEB table for machine original-consistent ETCs with fixed machine COV 0.1

COV of tasks		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	26	0	0	0	0	0	1	0	0	0	0	35	33	9	5	0
	NEB	0	0	563	616	0	517	515	0	615	0	517	515	0	0	0	0	0	0
0.2	NB	0	0	30	2	0	0	0	0	2	0	0	0	0	190	187	35	32	0
	NEB	0	0	265	304	0	242	215	0	304	0	242	215	0	0	0	0	0	0
0.3	NB	0	0	8	0	0	0	0	0	1	0	0	0	0	451	352	64	38	0
	NEB	0	0	34	46	0	36	35	0	43	0	36	35	0	0	0	0	0	0
0.4	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	556	348	51	43	0
	NEB	0	0	1	1	0	1	1	0	1	0	1	1	0	0	0	0	0	0
0.5	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	569	351	57	23	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.6	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	563	370	41	26	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 5
NB and NEB table for machine partial-consistent ETCs with fixed machine COV 0.1

COV of tasks		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	165	25	0	105	0	0	25	0	99	0	0	1	1	0	0	0
	NEB	0	0	45	277	0	131	181	0	276	0	129	181	0	0	0	0	0	0
0.2	NB	0	0	176	43	0	86	0	0	39	0	105	0	0	74	88	14	17	0
	NEB	0	0	10	146	0	91	124	0	146	0	90	124	0	0	0	0	0	0
0.3	NB	0	0	29	10	0	33	0	0	4	0	22	0	0	388	335	65	55	0
	NEB	0	0	1	21	0	17	21	0	21	0	17	21	0	0	0	0	0	0
0.4	NB	0	0	0	0	0	3	0	0	0	0	0	0	0	476	412	51	57	0
	NEB	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
0.5	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	517	396	41	46	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.6	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	526	377	54	43	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 6
NB and NEB table for machine full-consistent ETCs with fixed machine COV 0.1

COV of tasks		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	55	255	0	83	0	0	128	0	117	0	0	72	127	25	48	0
	NEB	0	0	2	1	0	0	88	0	1	0	2	88	0	0	0	0	0	0
0.2	NB	0	0	0	4	0	0	0	0	2	0	0	0	0	300	420	120	153	0
	NEB	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0
0.3	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	341	428	119	112	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.4	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	338	477	93	92	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.5	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	359	473	80	88	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.6	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	423	429	77	71	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

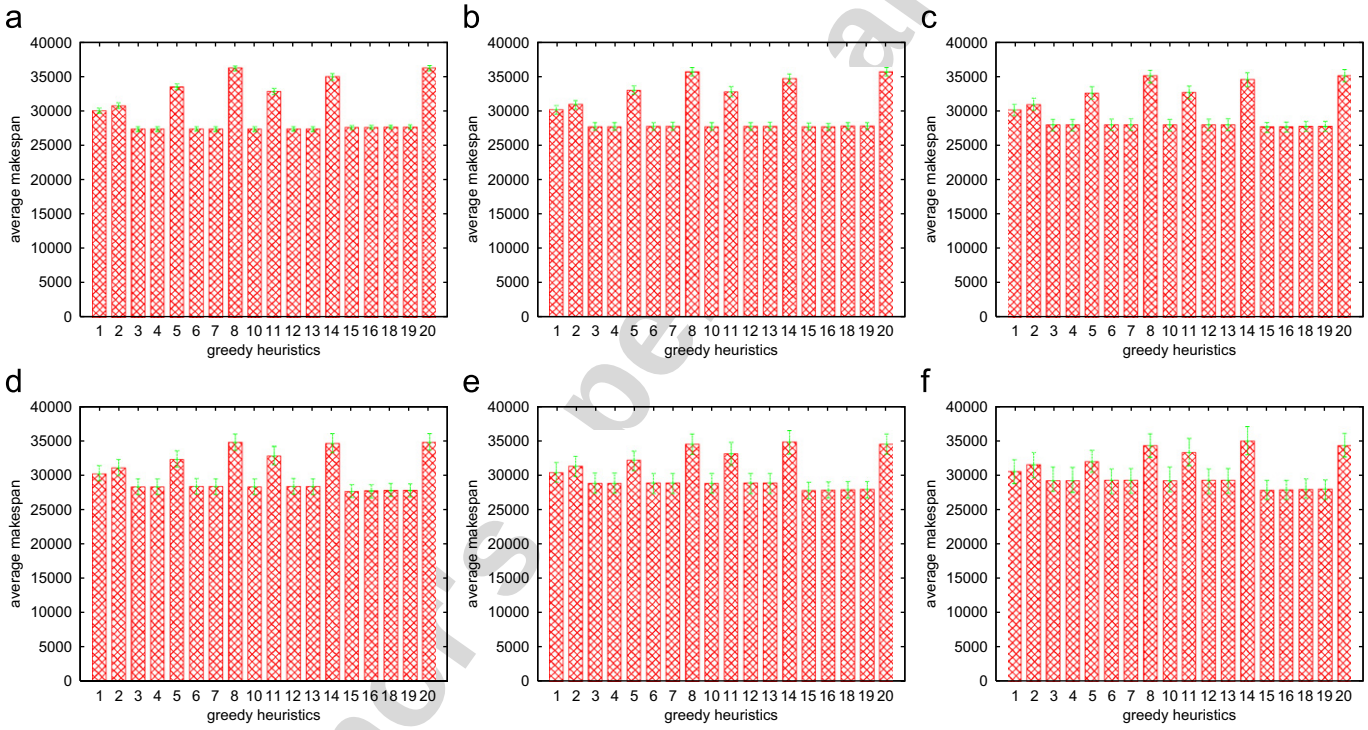

Fig. 6. Average makespan of the greedy heuristics when $V_{\text{machine}} = 0.1$ and ETCs are machine original-consistent. (a) $V_{\text{task}} = 0.1$, (b) $V_{\text{task}} = 0.2$, (c) $V_{\text{task}} = 0.3$, (d) $V_{\text{task}} = 0.4$, (e) $V_{\text{task}} = 0.5$, (f) $V_{\text{task}} = 0.6$.

- The *number of best solutions equal with another method* (denoted by NEB), which counts those cases where a particular method produced the shortest makespan but at least one other method also achieved the same makespan. NEB is the complement to NB.

All of the experiments are performed in two parts.

- In the first part, V_{task} increases with a step pace 0.1 while V_{machine} is fixed at a value smaller than V_{task}. In this process, the value of V_{task} minus V_{machine} becomes bigger and bigger, and thus the task consistency degree d_{task} of the generated ETCs increases gradually.
- In the second part, V_{machine} increases with a step pace 0.1 while the task COV v_{task} is fixed at a value bigger than V_{machine}. In this process, the value of V_{task} minus V_{machine} becomes smaller and smaller, and thus the task consistency degree d_{task} of the generated ETCs decreases gradually.

In each situation the values in *every row* of an ETC may or may not be sorted to form ETCs with different types of *machine consistency*: *original-consistency*, *partial-consistency* and *full-consistency*.

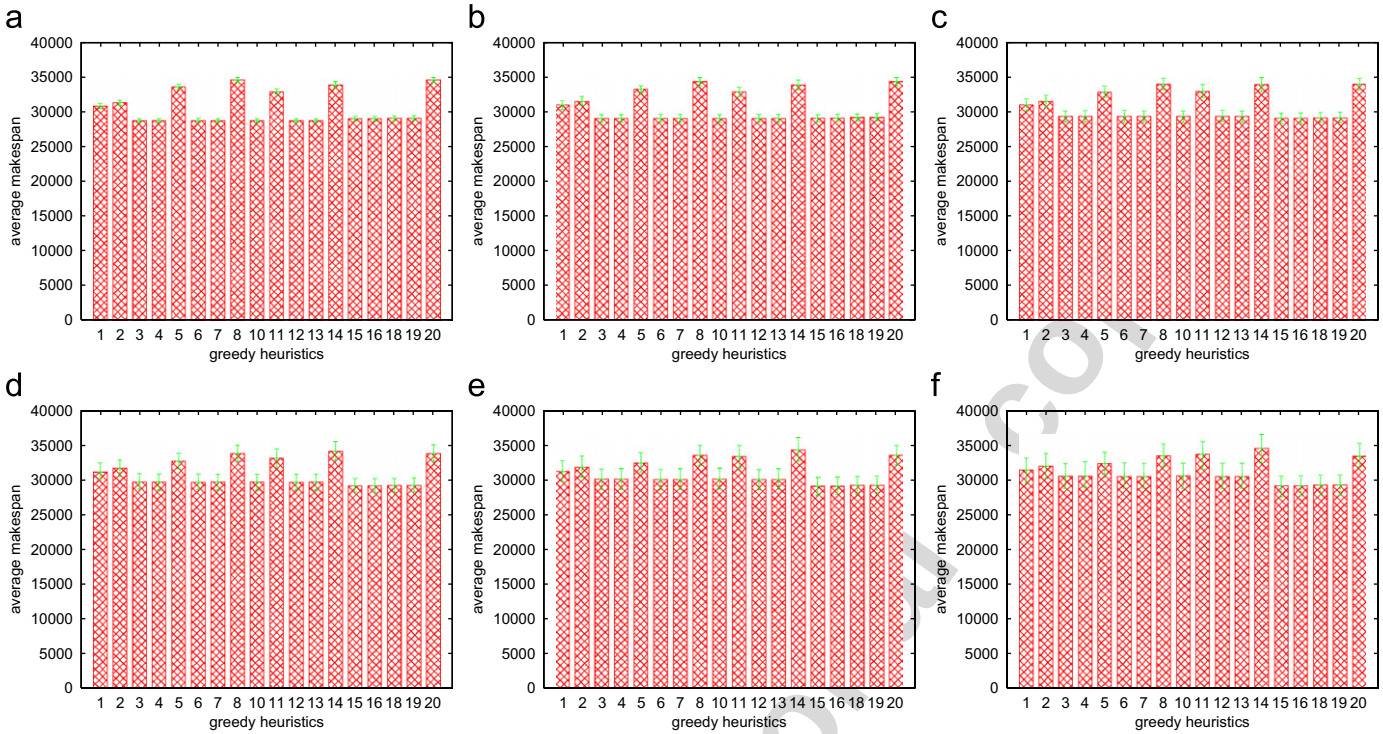

Fig. 7. Average makespan of the greedy heuristics when $V_{\text{machine}} = 0.1$ and ETCs are machine partial-consistent. (a) $V_{\text{task}} = 0.1$, (b) $V_{\text{task}} = 0.2$, (c) $V_{\text{task}} = 0.3$, (d) $V_{\text{task}} = 0.4$, (e) $V_{\text{task}} = 0.5$, (f) $V_{\text{task}} = 0.6$.

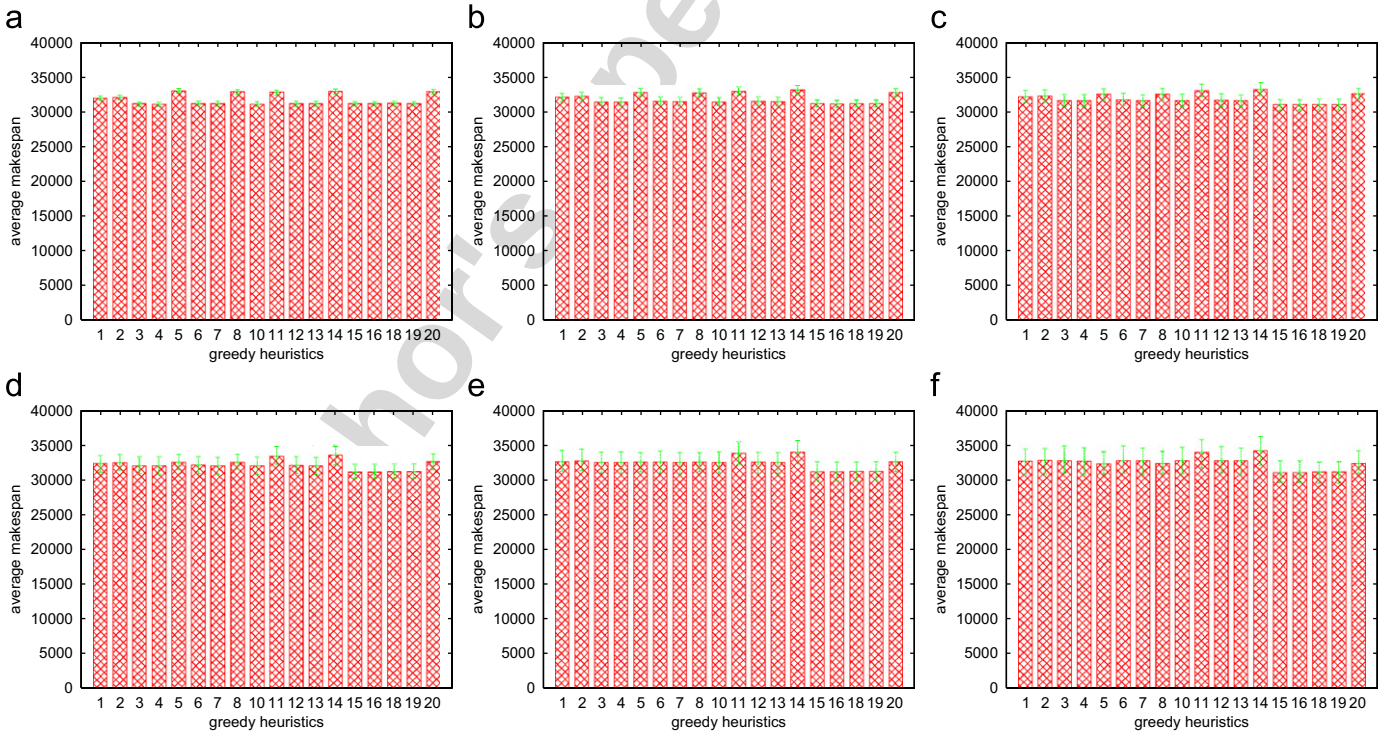

Fig. 8. Average makespan of the greedy heuristics when $V_{\text{machine}} = 0.1$ and ETCs are machine full-consistent. (a) $V_{\text{task}} = 0.1$, (b) $V_{\text{task}} = 0.2$, (c) $V_{\text{task}} = 0.3$, (d) $V_{\text{task}} = 0.4$, (e) $V_{\text{task}} = 0.5$, (f) $V_{\text{task}} = 0.6$.

Table 7
NB and NEB table for machine original-consistent ETCs with fixed machine COV 0.6

COV of tasks		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	12	0	0	1	0	0	1	0	0	0	0	846	24	9	0	0
	NEB	0	0	44	60	0	35	36	0	63	0	35	36	0	0	0	0	0	0
0.2	NB	0	0	7	0	0	2	0	0	0	0	0	0	0	917	11	8	0	0
	NEB	0	0	18	28	0	15	20	0	28	0	16	20	0	0	0	0	0	0
0.3	NB	0	0	5	1	0	0	0	0	0	0	0	0	0	953	14	6	0	0
	NEB	0	0	4	8	0	8	6	0	8	0	8	6	0	0	0	0	0	0
0.4	NB	0	0	2	1	0	1	0	0	0	0	0	0	0	977	6	2	0	0
	NEB	0	0	5	7	0	3	3	0	7	0	3	3	0	0	0	0	0	0
0.5	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	991	1	2	0	0
	NEB	0	0	2	3	0	1	2	0	3	0	1	2	0	0	0	0	0	0
0.6	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	993	1	3	0	0
	NEB	0	0	2	2	0	1	1	0	2	0	1	1	0	0	0	0	0	0

Table 8
NB and NEB table for machine partial-consistent ETCs with fixed machine COV 0.6

COV of tasks		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	100	84	0	79	1	0	65	0	66	6	0	287	130	1	0	0
	NEB	0	0	3	55	0	29	98	0	54	0	28	99	0	0	0	0	0	0
0.2	NB	0	0	66	52	0	44	1	0	58	0	56	9	0	446	155	4	0	0
	NEB	0	0	1	33	0	16	60	0	33	0	16	60	0	0	0	0	0	0
0.3	NB	0	0	44	21	0	28	4	0	25	0	34	0	0	556	200	5	0	0
	NEB	0	0	2	34	0	5	43	0	35	0	5	43	0	0	0	0	0	0
0.4	NB	0	0	31	24	0	20	2	0	17	0	20	3	0	685	149	6	0	0
	NEB	0	0	0	14	0	1	28	0	14	0	1	28	0	0	0	0	0	0
0.5	NB	0	0	19	11	0	13	0	0	8	0	14	3	0	731	181	5	0	0
	NEB	0	0	0	5	0	0	10	0	5	0	0	10	0	0	0	0	0	0
0.6	NB	0	0	7	6	0	3	1	0	8	0	3	1	0	804	144	13	0	0
	NEB	0	0	0	4	0	0	6	0	4	0	0	6	0	0	0	0	0	0

Table 9
NB and NEB table for machine full-consistent ETCs with fixed machine COV 0.6

COV of tasks		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	84	123	0	87	73	0	82	0	56	81	0	85	324	0	2	0
	NEB	0	0	0	0	0	0	3	0	0	0	0	3	0	0	0	0	0	0
0.2	NB	0	0	54	75	0	61	66	0	43	0	41	52	0	123	477	0	6	0
	NEB	0	0	0	0	0	0	2	0	0	0	0	2	0	0	0	0	0	0
0.3	NB	0	0	26	38	0	22	31	0	39	0	26	34	0	200	571	5	8	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.4	NB	0	0	13	14	0	21	20	0	16	0	12	12	0	235	643	6	8	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.5	NB	0	0	10	6	0	8	7	0	9	0	10	5	0	246	688	3	8	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.6	NB	0	0	3	3	0	0	6	0	5	0	1	5	0	292	654	14	16	0
	NEB	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0

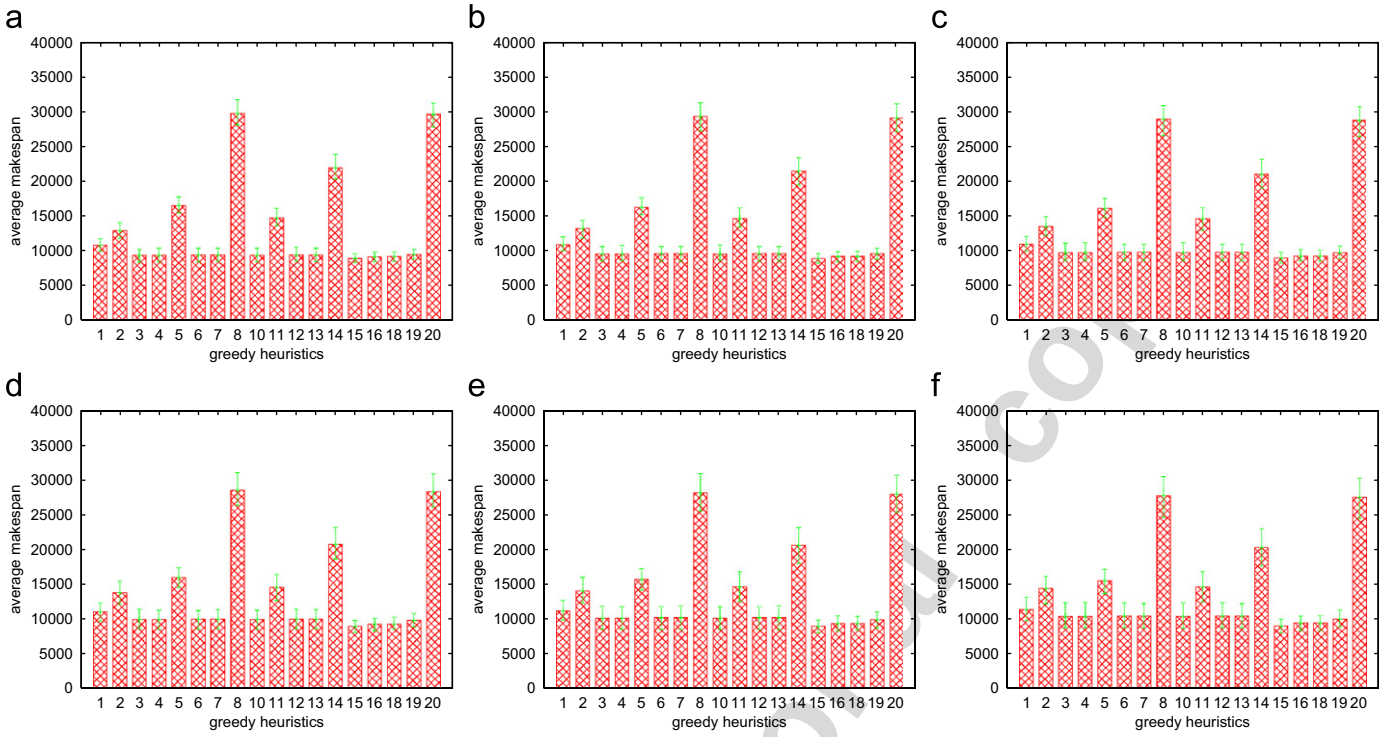

Fig. 9. Average makespan of the greedy heuristics when $V_{\text{machine}} = 0.6$ and ETCs are machine original-consistent. (a) $V_{\text{task}} = 0.6$, (b) $V_{\text{task}} = 0.7$, (c) $V_{\text{task}} = 0.8$, (d) $V_{\text{task}} = 0.9$, (e) $V_{\text{task}} = 1.0$, (f) $V_{\text{task}} = 1.1$.

5.1. *Increasing* V_{task}, *fixing* V_{machine}

In this part two experimental situations are considered while increasing V_{task} and fixing V_{machine}. Firstly, V_{machine} is fixed at 0.1 while V_{task} increases from 0.1 to 0.6. Secondly, V_{machine} is fixed at 0.6 while V_{task} increases from 0.6 to 1.1.

Tables 4 through 6 record the NB and NEB of the heuristics used for the ETCs in the first situation, with machine original-consistency, machine partial-consistency and machine full-consistency, respectively. Figs. 6 through 8 show the average makespan of the heuristics applied to the ETCs in the first situation, with machine original-consistency, machine partial-consistency and machine full-consistency, respectively. Tables 7 through 9 and Figs. 9 through 11 record the corresponding experimental results for ETCs in the second situation.

For the first situation Tables 4 through 6 and Figs. 6 through 8 show that H_{15} is the best among all the heuristics with the following two exceptions:

- H_3 outperforms H_{15} if d_{task} of an ETC is small when V_{task} is relatively small at the range of [0.1, 0.2] and V_{machine} is fixed at 0.1.
- H_{16} always outperforms H_{15} when ETCs are machine full-consistent.

By calculating the average makespan of H_3 minus that of H_{15} along the increase of V_{task} for three types of *machine consistency*, Fig. 12(a) shows that the superiority of H_{15} over H_3 is more and more clear along the increase of d_{task} (d_{task} increases with the increase of V_{task} when V_{machine} is fixed), because the average makespan of H_3 minus that of H_{15} increases monotonically in this situation. Additionally, this result occurs regardless of the types of *machine consistency*. By calculating the average makespan of H_{15} minus that of H_{16} along the increase of V_{task} for three types of *machine consistency*, Fig. 13(a) reconfirms that H_{16} is better than H_{15} when ETCs are machine full-consistent, as the line for machine full-consistency in it is always above $y = 0$.

For the second situation the results in Tables 7 through 9 and Figs. 9 through 11 coincide with the conclusion of the first situation with the following extra findings:

- H_{15} outperforms H_3 clearly even if d_{task} of an ETC is relatively small when V_{task} and V_{machine} are both fixed at 0.6.
- The superiority of H_{15} over H_3 in the second situation (Fig. 12(b)) is clearer than that in the first situation (Fig. 12(a)).
- The predominance of H_{16} over H_{15} in the second situation (Table 9 and Fig. 13(b)) is clearer than that in the first situation (Table 6 and Fig. 13(a)).

Fig. 13 also shows that the line for machine partial-consistency is in the middle of the other two lines. It indicates that the superiority of H_{16} over H_{15} is related to the degree of machine consistency d_{machine}.

5.2. *Increasing* V_{machine}, *fixing* V_{task}

In this subsection, the experiments are performed while increasing V_{task} and fixing V_{machine}. V_{task} is fixed at 0.6 while V_{machine} increases from 0.1 to 0.6.

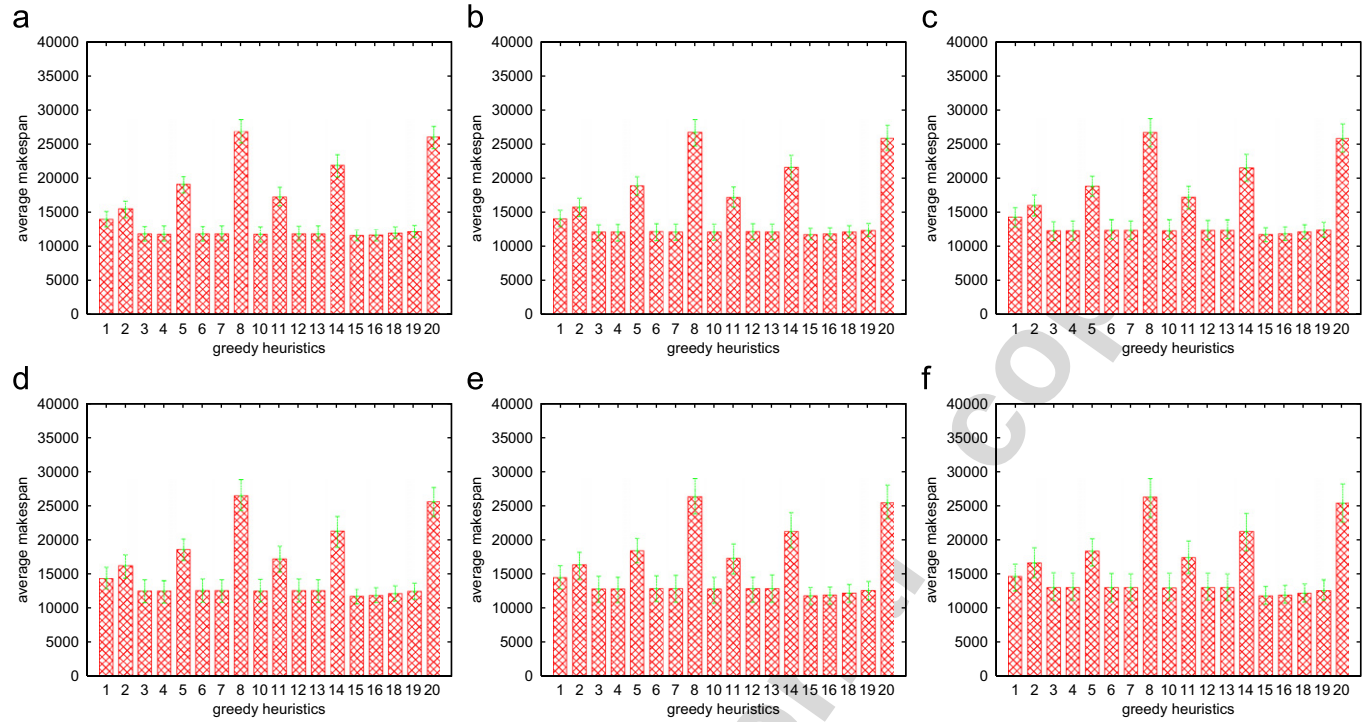

Fig. 10. Average makespan of the greedy heuristics when $V_{\text{machine}} = 0.6$ and ETCs are machine partial-consistent. (a) $V_{\text{task}} = 0.6$, (b) $V_{\text{task}} = 0.7$, (c) $V_{\text{task}} = 0.8$, (d) $V_{\text{task}} = 0.9$, (e) $V_{\text{task}} = 1.0$, (f) $V_{\text{task}} = 1.1$.

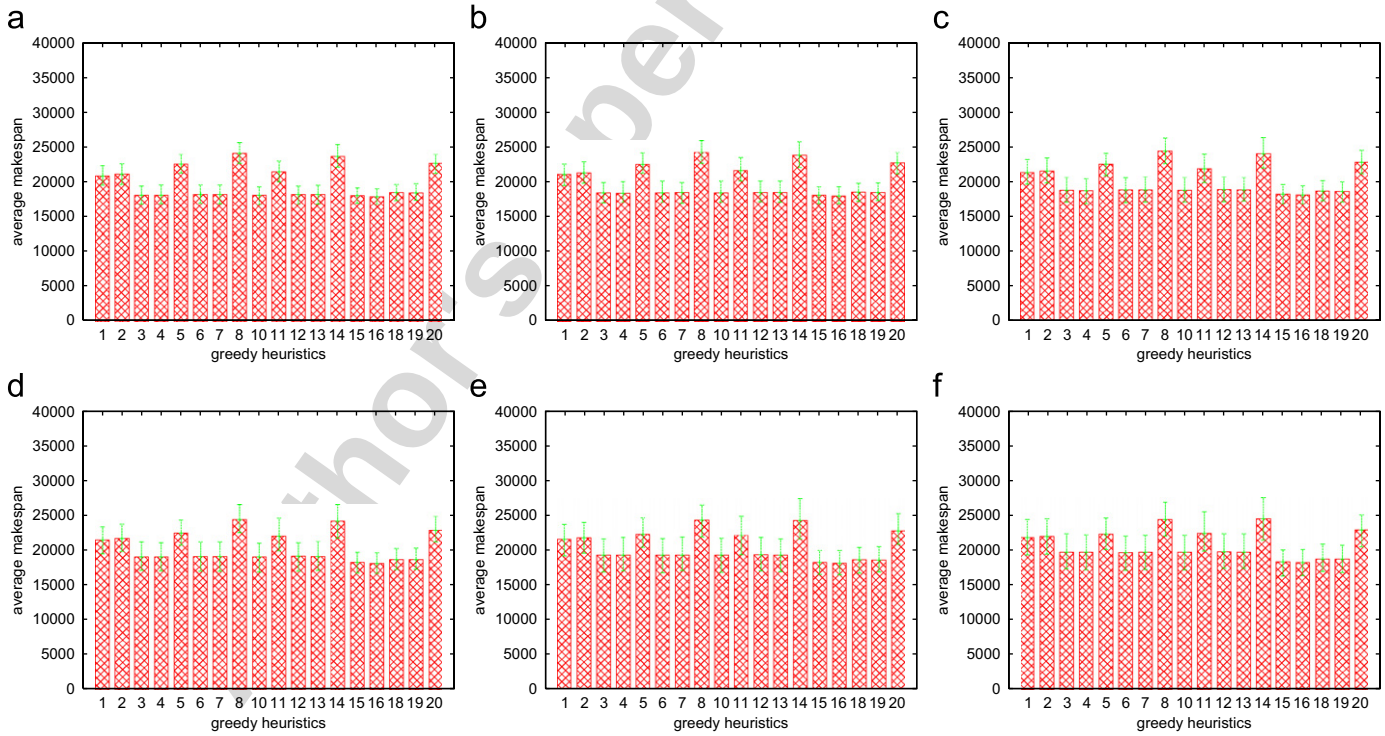

Fig. 11. Average makespan of the greedy heuristics when $V_{\text{machine}} = 0.6$ and ETCs are machine full-consistent. (a) $V_{\text{task}} = 0.6$, (b) $V_{\text{task}} = 0.7$, (c) $V_{\text{task}} = 0.8$, (d) $V_{\text{task}} = 0.9$, (e) $V_{\text{task}} = 1.0$, (f) $V_{\text{task}} = 1.1$.

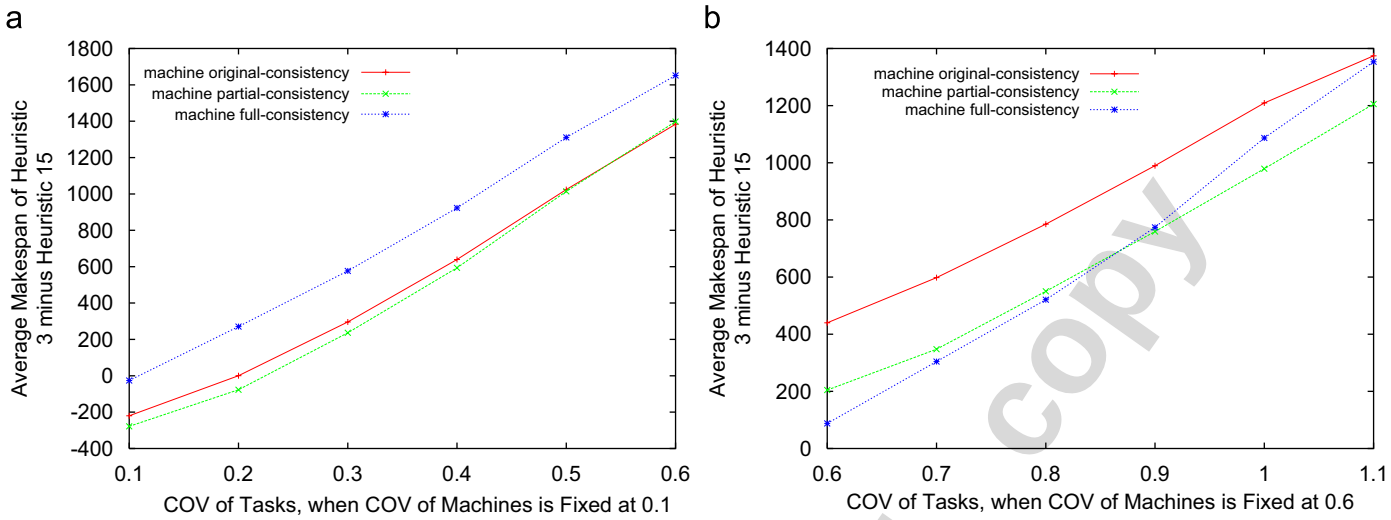

Fig. 12. Comparison between H_3 and H_{15} along the increase of V_{task}. (a) $V_{\text{machine}} = 0.1$, (b) $V_{\text{machine}} = 0.6$.

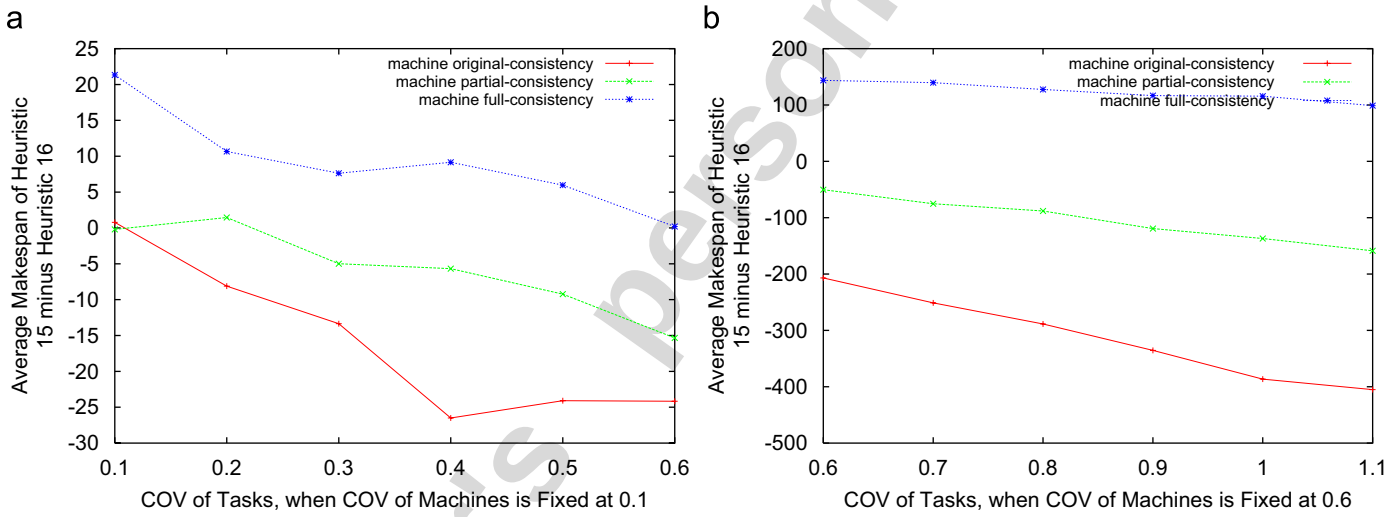

Fig. 13. Comparison between H_{15} and H_{16}. (a) $V_{\text{machine}} = 0.1$, (b) $V_{\text{machine}} = 0.6$.

Table 10
NB and NEB table for machine original-consistent ETCs with fixed task COV 0.6

COV of machines		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	563	370	41	26	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.2	NB	0	0	2	0	0	0	0	0	0	0	0	0	0	673	300	12	6	0
	NEB	0	0	2	4	0	3	0	0	4	0	3	0	0	0	0	0	0	0
0.3	NB	0	0	7	0	0	0	0	0	0	0	1	0	0	742	197	13	1	0
	NEB	0	0	18	17	0	13	12	0	18	0	13	12	0	0	0	0	0	0
0.4	NB	0	0	10	0	0	0	0	0	0	0	1	0	0	815	95	14	0	0
	NEB	0	0	19	33	0	17	26	0	34	0	17	26	0	0	0	0	0	0
0.5	NB	0	0	7	0	0	0	0	0	0	0	0	0	0	856	33	13	0	0
	NEB	0	0	26	47	0	36	38	0	47	0	37	38	0	0	0	0	0	0
0.6	NB	0	0	12	0	0	1	0	0	1	0	0	0	0	846	24	9	0	0
	NEB	0	0	44	60	0	35	36	0	63	0	35	36	0	0	0	0	0	0

Table 11
NB and NEB table for machine partial-consistent ETCs with fixed task COV 0.6

COV of machines		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	526	377	54	43	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.2	NB	0	0	7	1	0	1	0	0	1	0	2	0	0	472	488	10	14	0
	NEB	0	0	0	3	0	0	1	0	3	0	0	1	0	0	0	0	0	0
0.3	NB	0	0	27	29	0	23	0	0	20	0	24	0	0	460	360	3	1	0
	NEB	0	0	0	10	0	3	40	0	10	0	3	40	0	0	0	0	0	0
0.4	NB	0	0	64	55	0	44	0	0	57	0	39	0	0	395	268	0	3	0
	NEB	0	0	0	18	0	5	52	0	18	0	5	52	0	0	0	0	0	0
0.5	NB	0	0	93	73	0	66	0	0	57	0	68	0	0	317	210	0	0	0
	NEB	0	0	0	23	0	19	75	0	23	0	19	75	0	0	0	0	0	0
0.6	NB	0	0	100	84	0	79	1	0	65	0	66	6	0	287	130	1	0	0
	NEB	0	0	3	55	0	29	98	0	54	0	28	99	0	0	0	0	0	0

Table 12
NB and NEB table for machine full-consistent ETCs with fixed task COV 0.6

COV of machines		simple-*heuristic*		NTPD-*heuristic*						TPD↑-*heuristic*					TPD↓-*heuristic*				
		H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8	H_{10}	H_{11}	H_{12}	H_{13}	H_{14}	H_{15}	H_{16}	H_{18}	H_{19}	H_{20}
0.1	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	423	429	77	71	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.2	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	284	685	6	25	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.3	NB	0	0	0	0	0	0	0	0	0	0	0	0	0	187	809	0	4	0
	NEB	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0.4	NB	0	0	7	3	0	9	3	0	5	0	2	3	0	143	818	0	2	0
	NEB	0	0	0	0	0	0	5	0	0	0	0	5	0	0	0	0	0	0
0.5	NB	0	0	48	47	0	35	40	0	23	0	25	40	0	98	631	1	2	0
	NEB	0	0	0	0	0	0	10	0	0	0	0	10	0	0	0	0	0	0
0.6	NB	0	0	84	123	0	87	73	0	82	0	56	81	0	85	324	0	2	0
	NEB	0	0	0	0	0	0	3	0	0	0	0	3	0	0	0	0	0	0

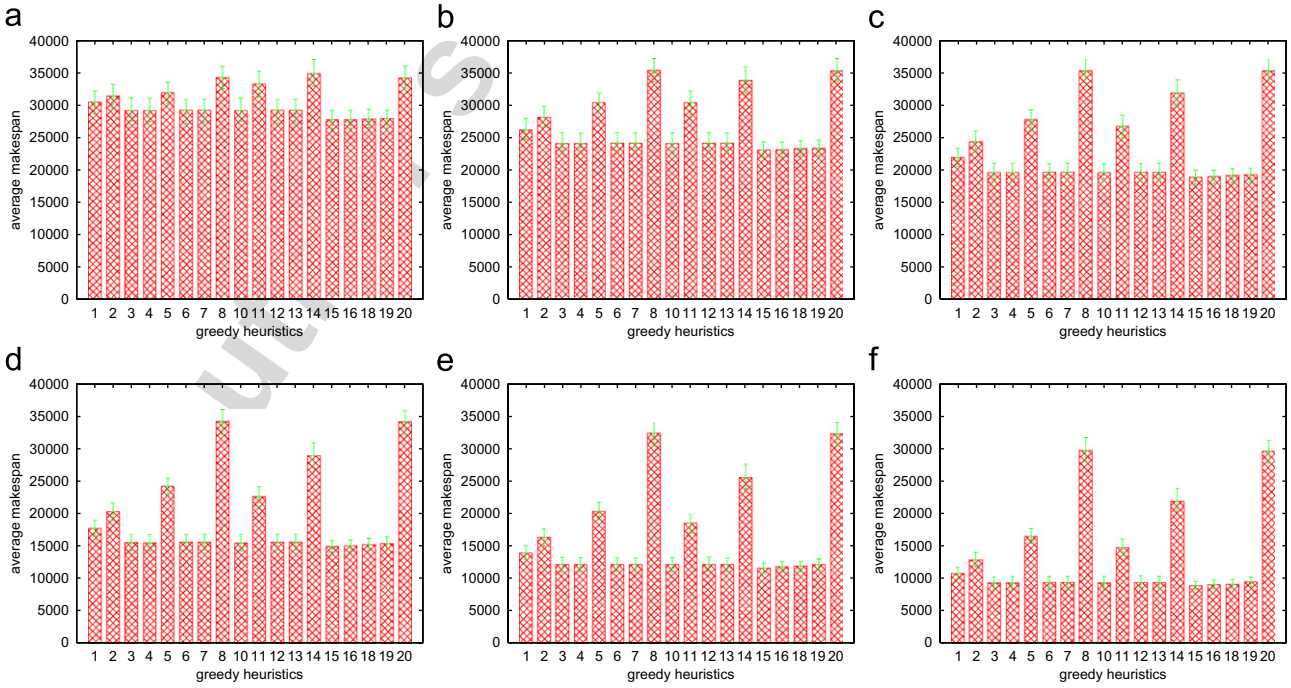

Fig. 14. Average makespan of the greedy heuristics when $V_{task} = 0.6$ and ETCs are machine original-consistent. (a) $V_{machine} = 0.1$, (b) $V_{machine} = 0.2$, (c) $V_{machine} = 0.3$, (d) $V_{machine} = 0.4$, (e) $V_{machine} = 0.5$, (f) $V_{machine} = 0.6$.

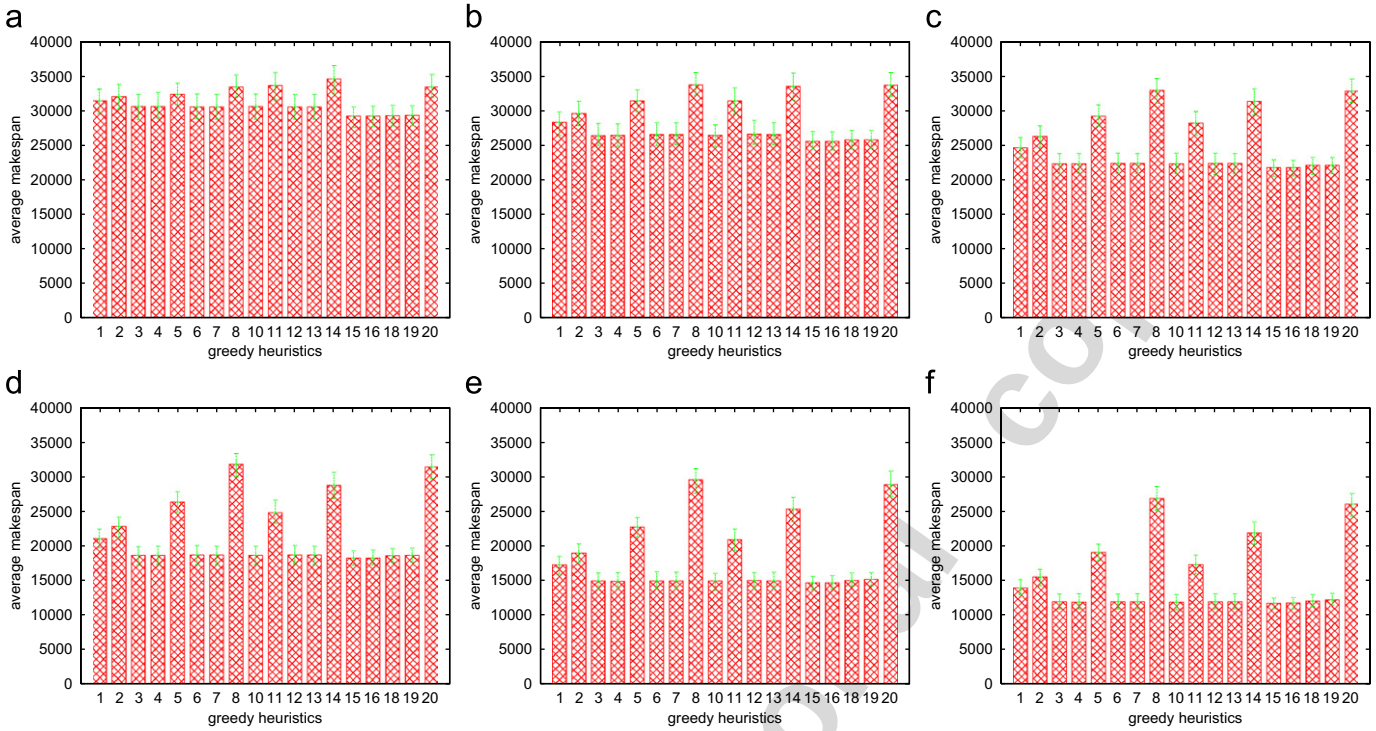

Fig. 15. Average makespan of the greedy heuristics when $V_{\text{task}} = 0.6$ and ETCs are machine partial-consistent. (a) $V_{\text{machine}} = 0.1$, (b) $V_{\text{machine}} = 0.2$, (c) $V_{\text{machine}} = 0.3$, (d) $V_{\text{machine}} = 0.4$, (e) $V_{\text{machine}} = 0.5$, (f) $V_{\text{machine}} = 0.6$.

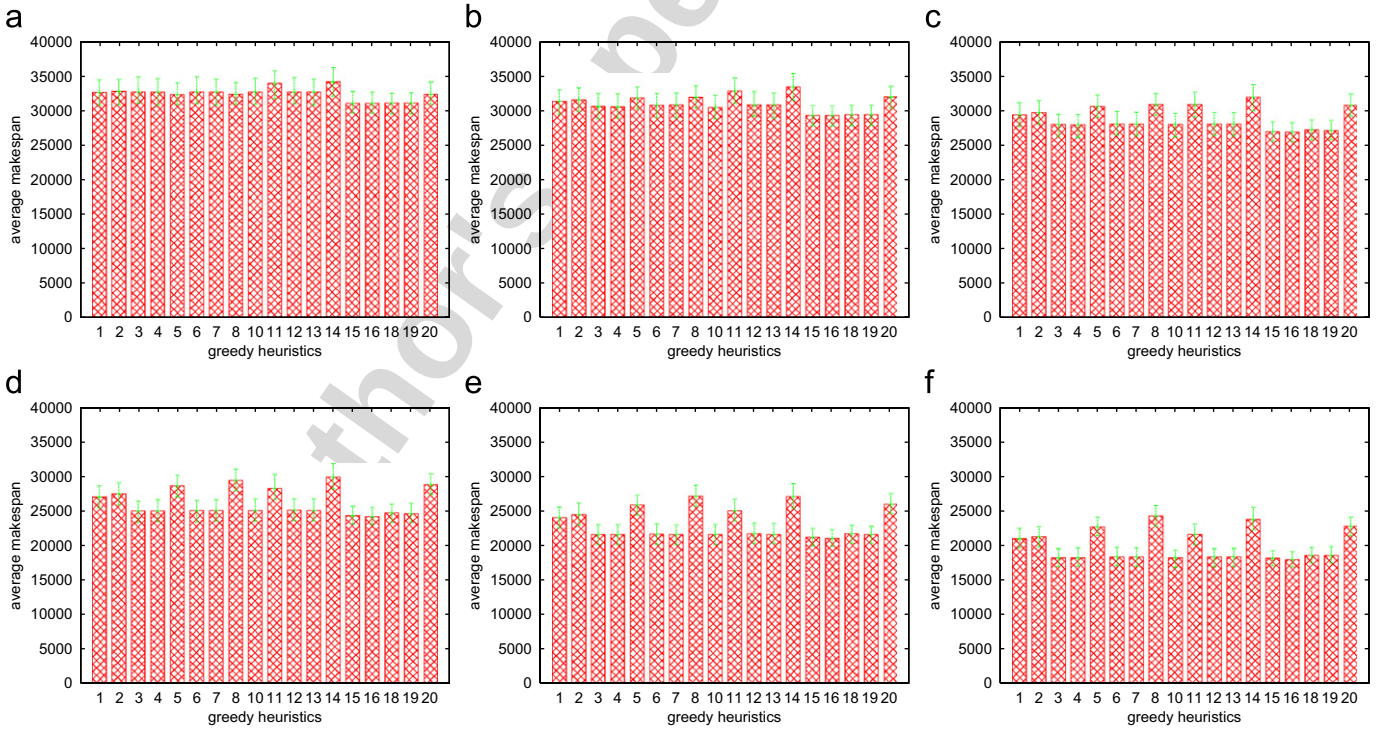

Fig. 16. Average makespan of the greedy heuristics when $V_{\text{task}} = 0.6$ and ETCs are machine full-consistent. (a) $V_{\text{machine}} = 0.1$, (b) $V_{\text{machine}} = 0.2$, (c) $V_{\text{machine}} = 0.3$, (d) $V_{\text{machine}} = 0.4$, (e) $V_{\text{machine}} = 0.5$, (f) $V_{\text{machine}} = 0.6$.

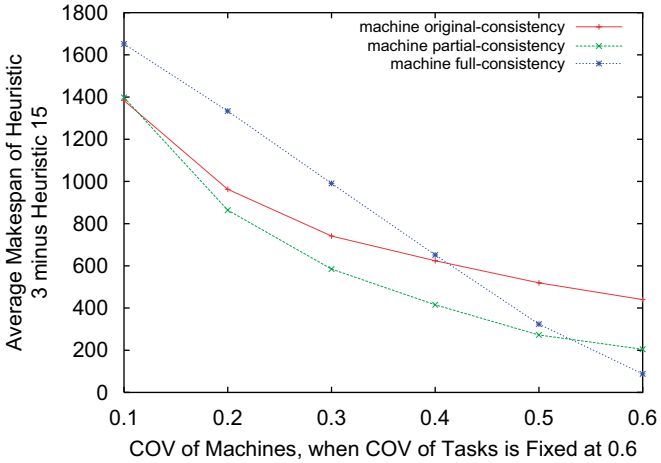

Fig. 17. Comparison between H_3 and H_{15} along the increase of V_{machine} when $V_{\text{task}} = 0.6$.

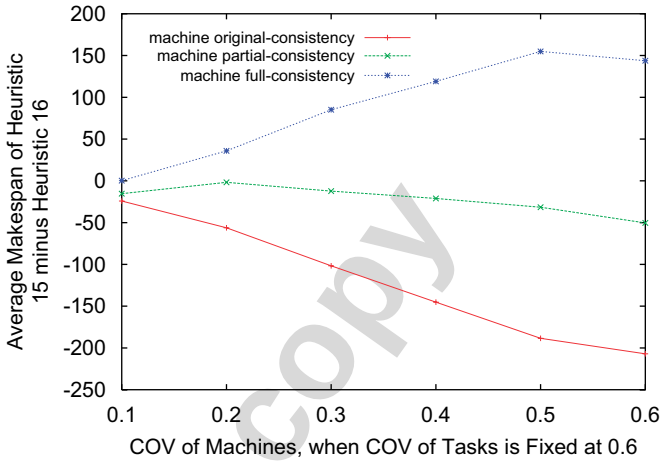

Fig. 18. Comparison between H_{15} and H_{16} when $V_{\text{task}} = 0.6$.

Tables 10 through 12 record the NB and NEB of the heuristics used for the ETCs in this situation, with machine original-consistency, machine partial-consistency and machine full-consistency, respectively. Figs. 14 through 16 show the average makespan of the heuristics applied to the ETCs in this situation, also with machine original-consistency, machine partial-consistency and machine full-consistency, respectively.

The results from this experiment part coincide with that in Section 5.1. By calculating the average makespan of H_3 minus that of H_{15} along the increase of V_{machine} for three types of *machine consistency*, Fig. 17 shows that the superiority of H_{15} over H_3 becomes weaker along the decrease of d_{task} (d_{task} decreases with the increase of V_{machine} when V_{task} is fixed), because the average makespan of H_3 minus that of H_{15} decreases monotonically in this situation. By calculating the average makespan of H_{15} minus that of H_{16} along the increase of V_{machine} for three types of *machine consistency*, the line for machine full-consistency in Fig. 18 is always above $y = 0$, which also indicates that H_{16} is better than H_{15} when ETCs are machine full-consistent. Fig. 18 also shows that the line for machine partial-consistency is in the middle of the other two lines, which coincides with the situations in Fig. 13.

5.3. Summary of experiment

Based on the analysis of the experiment results, two conclusions can be drawn:

1. H_{15} is the best among all the heuristics with two exceptions:
   - H_3 acts better than H_{15} when V_{task} and V_{machine} are both small at the range of $[0.1, 0.2]$.
   - H_{16} outperforms H_{15} when d_{machine} reaches 1 (machine full-consistency).
2. The superiority of H_{15} over H_3 demonstrates more clearly along the increase of d_{task}. This conclusion is regardless of d_{machine}.

The above conclusion coincides with the previous analysis in Section 2: if the task with a bigger *task effort* is mapped at the end, it is more likely to break the balance of loads among machines and generate a worse mapping result. Additionally, when the increase of V_{task} and d_{task} enlarges the heterogeneity of task execution times among the given tasks on each machine, this phenomenon happens more frequently. Hence, this conclusion holds.

Therefore, for a practical mapping problem Algorithm 5.1 generates the best fast greedy heuristic according to different types of a given ETC.

Algorithm 5.1 The best fast greedy heuristic

```
if The ETC is machine full-consistency then
    return H16
else
    if Vtask and Vmachine of the input ETC are both small then
        return H3
    else
        return H15
    end if
end if
```

6. Conclusions

The *fast greedy* mapping heuristic is important to the HC environments when the scheduling process performs during the execution of mapped tasks. In this study we introduce the concept of *task consistency*. We claim that this trait is significant in distinguishing the full functions of different mapping heuristics as it is closely related to the load balance of different machines. We claim that both *task consistency* and *machine consistency* must be considered to tackle the mapping problem. We claim further that previous works only considered *machine consistency*, and *task consistency* was ignored. In this paper, we investigated 20 *fast greedy* algorithms, including 17 newly proposed

heuristics, and compared them within a unified ETC model with four parameters: *task consistency*, *machine consistency*, *task heterogeneity* and *machine heterogeneity*. The goal of this study was to give an insight into the circumstances when a specific greedy heuristic would outperform the other 19 greedy heuristics. Based on the experimental results, we mainly identified that H_{15} with the support of a *task priority graph* outperforms the other heuristics, including *min–min*, in most situations.

During the experimental process, we also found that the COV-based generating method for ETCs is better than the widely used range-based method of previous studies, since it provides greater control over spread of the execution time values. The systematic comparison method used in this paper, considering both *task consistency* and the other three parameters of an ETC, provides a general approach to study the mapping heuristics and can be used to conduct further investigations in this area.

Acknowledgments

The authors wish to acknowledge the careful and judicious review of the anonymous reviewers who helped them to greatly improve the presentation and substance of this paper. This work was supported by the 973 National Basic Research Program of China (No. 2003CB317004), the National Science Foundation of China (Nos. 60435010, 60675010, and 90604017) and the Nature Science Foundation of Beijing (No. 4052025). Kevin Lü would like to express his appreciation to the Wang Kuan Cheng Science Foundation, Chinese Academy of Sciences for the funding that enabled him to conduct this research.

References

[1] S. Ali, H.J. Siegel, M. Maheswaran, S. Ali, D. Hensgen, Task execution time modelling for heterogeneous computing systems, in: Proceedings of the Ninth Heterogeneous Computing Workshop, 2000, pp. 185–200.
[2] T.D. Braun, D. Hensgen, R.F. Freund, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel and Distributed Comput. 61 (6) (2001) 810–837.
[3] R.C.H. Cheng, The generation of gamma variables with non-integral shape parameter, Appl. Statist. 26 (1977) 71–75.
[4] D. Fernandez-Baca, Allocating modules to processors in a distributed system, IEEE Trans. Software Eng. 15 (11) (1989) 1427–1436.
[5] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufman, New York, 1998.
[6] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on non-identical processors, J. ACM 24 (2) (1977) 280–289.
[7] M. Iverson, F. Ozguner, Dynamic, competitive scheduling of multiple dags in a distributed heterogeneous environment, in: Proceedings of the Eighth Heterogeneous Computing Workshop, 1999.
[8] M.A. Iverson, F. Ozguner, L. Potter, Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment, IEEE Trans. Comput. 48 (12) (1999) 1374–1379.
[9] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Comput. Surveys 31 (4) (1999) 406–471.
[10] Y.-K. Kwok, I. Ahmad, Benchmarking and comparison of the task graph scheduling algorithms, J. Parallel and Distributed Comput. 59 (3) (1999) 381–422.
[11] G.Q. Liu, K.L. Poh, M. Xie, Iterative list scheduling for heterogeneous computing, J. Parallel and Distributed Comput. 65 (5) (2005) 654–665.
[12] P. Luo, K. Lü, R. Huang, Q. He, Z. Shi, A heterogeneous computing system for data mining workflows in multi-agent environments, Expert Systems: the J. Knowledge Eng. 23 (5) (2007) 258–272.
[13] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: Proceedings of the Eighth Heterogeneous Computing Workshop, 1999.
[14] G. Ritchie, J. Levine, A fast, effective local search for scheduling independent jobs in heterogeneous computing environments, in: Proceedings of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group, 2003.
[15] G. Ritchie, J. Levine, A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments, in: Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004.
[16] R. Sakellariou, H. Zhao, A hybrid heuristic for dag scheduling on heterogeneous systems, in: Proceedings of the 13th Heterogeneous Computing Workshop, 2004.
[17] S. Shivle, P. Sugavanam, H.J. Siegel, A.A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Smith, J. Sousa, J. Sridharan, J. Velazco, Mapping subtasks with multiple versions on an ad hoc grid, Parallel Comput. Special Issue on Heterogeneous Comput. 31 (7) (2005) 671–690.
[18] SSJ: Stochastic Simulation in Java. ⟨http://www.iro.umontreal.ca/simardr/ssj/⟩.
[19] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel and Distributed Comput. 47 (1) (1997) 1–15.

Ping Luo is a PhD student in the Intelligent Science Group at the Institute of Computing Technology, the Chinese Academy of Sciences. His research interests include algorithms and computing architecture of distributed data mining, P2P data mining, machine learning, and novel applications in data mining.

Kevin Lü is a senior lecturer at Brunel University, Uxbridge, UK. He has been working on a number of projects on parallel databases, data mining and multi-agent systems. His current research interests include multi-agent systems, data management, data mining, distributed computing and parallel techniques. He has published more than 40 research papers.

Zhongzhi Shi is a professor at the Institute of Computing Technology, the Chinese Academy of Sciences, leading the Research Group of Intelligent Science. His research interests include intelligence science, multi-agent systems, semantic Web, machine learning and neural computing. He has won a second-Grade National Award at Science and Technology Progress of China in 2002, two second-Grade Awards at Science and Technology Progress of the Chinese Academy of Sciences in 1998 and 2001, respectively. He is a senior member of IEEE, member of AAAI and ACM, Chair for the WG 12.2 of IFIP. He serves as Vice President for Chinese Association of Artificial Intelligence, Executive President of Chinese Neural Network Council.