J Supercomput (2012) 62:461–485
DOI 10.1007/s11227-011-0729-7

Characterization of the iterative application of makespan heuristics on non-makespan machines in a heterogeneous parallel and distributed environment

Luis Diego Briceño · Howard Jay Siegel · Anthony A. Maciejewski · Mohana Oltikar

Published online: 29 March 2012

Abstract Heterogeneous computing (HC) is the coordinated use of different types of machines, and networks to process a diverse workload in a manner that will maximize the combined performance and/or cost effectiveness of the system. Heuristics for allocating resources in an HC system are based on some optimization criterion. A common optimization criterion is to minimize the completion time of the machine that finishes last (makespan). In this study, we consider an iterative approach that repeatedly runs a mapping heuristic to minimize the makespan of the considered machines and tasks. For each successive iteration, the makespan machine of the previous iteration and the tasks assigned to it are removed from the set of considered machines and tasks. This study focuses on understanding the different mathematical characteristics of resource allocation heuristics that cause them to behave differently when combined with this iterative approach. This paper has three main contributions. The first contribution is the study of an iterative technique used in conjunction with resource allocation heuristics. The second contribution is the definition and mathematical characterization of "iteration invariant" heuristics. The third contribution is to determine the characteristics of a heuristic that will cause the mapping to change across iterations.

L.D. Briceño (✉) · H.J. Siegel · A.A. Maciejewski · M. Oltikar
Department of Electrical & Computer Engineering, Colorado State University, Fort Collins, CO 80523, USA
e-mail: ldbricen@colostate.edu

H.J. Siegel
e-mail: hj@colostate.edu

A.A. Maciejewski
e-mail: aam@colostate.edu

M. Oltikar
e-mail: mohana@colostate.edu

H.J. Siegel
Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA

Keywords Heterogeneous computing · Resource allocation · Heuristics · Parallel computing

1 Introduction

The use of heuristics for resource allocation in a heterogeneous parallel and distributed computing environment is an important area of research and has been widely studied (e.g., [4, 9, 40, 45]). Static and dynamic mapping of tasks to machines are both used to do resource allocation [1]. Static heuristics can be used in production environments, where the tasks to be executed are known in advance; dynamic heuristics are used to allocate resources on-line, without prior knowledge of when or which tasks arrive. In this study, we will consider static task mapping. One metric for evaluating the performance of heuristics is the time to complete a set of tasks on a heterogeneous suite of machines, i.e., *makespan*. The *makespan machine* is defined as the machine with the largest completion time.

Many of the static heuristics analyzed in this study can be used in dynamic batch mode. For example, collections of tasks that arrive during each fixed time interval between mapping events are collected together in a batch. This batch can then be mapped to resources using the types of heuristics discussed here.

In this study, our goal is to understand the behavior of different heuristic techniques when combined with an iterative procedure. The iterative procedure we consider here is to repeatedly minimize the makespan among the non-makespan machines. For each successive iteration, the makespan machine of the previous iteration and the tasks assigned to it are removed from the set of considered machines and tasks. Then the selected heuristic technique generates a new resource allocation with that reduced set of tasks and machines. This study focuses on understanding the different mathematical characteristics of resource allocation heuristics that cause them to behave differently when combined with the iterative approach.

This paper has three main contributions. The first contribution is the study of an iterative technique used in conjunction with resource allocation heuristics. The second contribution is the definition and mathematical characterization of "iteration invariant" heuristics. The third contribution is to determine the characteristics of a heuristic that will cause the mapping to change across iterations. The heuristics considered for this study were Minimum Execution Time, Minimum Completion Time, Min–Min, Genetic Algorithm, Switching Algorithm, Sufferage, and K-Percent Best.

Makespan is often the performance feature in the study of resource allocation in a heterogeneous computing system [10–12, 14, 30, 31, 36, 42]. Many studies explore different methods of reducing the makespan of the given set of tasks. The literature was examined to select a set of heuristics appropriate for the HC environment considered in this study. The Minimum Execution Time (MET), Minimum Completion Time (MCT) [2, 9, 10, 20, 34], and Min–Min [5, 7, 10, 16, 20, 22, 23, 25, 27, 34, 45] heuristics implemented here are adapted from [22]. The K-percent Best (KPB) [2, 10, 20, 32] and Switching Algorithm (SWA) [20, 27, 32, 34] were adapted from [32] and the Sufferage Algorithm [5, 6, 13, 16, 19, 23, 25, 27, 32, 37, 39, 43] was adapted from [32]. The variation of the Genetic Algorithm (GA) implemented here is an adaptation of the GA in [38].

In this study, we use a heuristic and iteratively apply it to a set of tasks and machines that becomes smaller after each iteration. This approach has not been studied in the past. Note that the iterative approach is different from simply running a heuristic multiple times on the same set of machines and tasks (e.g., multiple runs of a GA). In this study, the iterative approach is presented as a possible complement to any heuristic for resource allocation in an HC system.

The remainder of the paper is organized as follows. Section 2 describes the problem statement in detail. In Sect. 3, we describe three heuristics that when used in conjunction with the iterative approach their iterative mappings are the same as their original mapping. Some examples of heuristics where the original and iterative mappings may be different are described in Sect. 4. The observations from Sects. 3 and 4 are analyzed in Sects. 5, and 6 concludes the work.

2 Problem statement

Let T be the set of tasks that must be executed on a set of machines M. The estimated time to compute (ETC) each task on each machine is assumed to be known in advance and contained in an ETC matrix [9]. The ETC values can be based on user supplied information, experimental data, or task profiling and analytical benchmarking [1, 18, 21, 26, 33, 47]. Determination of ETC values is a separate research problem; the assumption of such ETC information is a common practice in resource allocation research (e.g., [3, 15, 21, 24, 26, 29, 41, 46]). Programs that are found in government laboratories and industry tend to be executed frequently (i.e., the same program with different data sets). For example, programs being executed at the National Center for Atmospheric Research (NCAR), Oak Ridge National Laboratory, and DigitalGlobe run the same program with different data sets [8]. Thus, historical or experimental information can be collected to characterize a task.

The *initial ready time* for a machine is the time at which the machine will become available to begin processing its first task from T. Tasks are assumed to be independent, i.e., no inter-task communication is required. We make the common simplifying assumption that each machine can only execute one task at a time, i.e., multitasking is not allowed (e.g., [17, 28]).

For each heuristic, the mapping it produces when all tasks and machines are available is called the *original mapping*. After each iteration (of the iterative approach), the makespan machine and the tasks assigned to it are removed from consideration, and the ready times for all remaining machines are reset to their initial ready times. The tasks that are available for mapping (were not mapped to the makespan machine in the original mapping) are mapped again, using the same heuristic to minimize makespan among the remaining machines; this mapping is called the *iterative mapping*. In the context of this paper, *mappable tasks* are those tasks that a heuristic can select from to assign at a given point during the allocation. The *available machines* are the machines that can be assigned a mappable task. It is important to note that both mappable tasks and available machines are dependent on the heuristic used in the iterative approach.

1. A task list is generated that includes all unmapped tasks in a given arbitrary order.
2. The first task in the list is assigned to its minimum execution time machine.
3. The task selected in step 2 is removed from the task list.
4. Steps 2–3 are repeated until all tasks have been mapped.

Fig. 1 Procedure for using MET to generate a resource allocation

Whether the iterative approach will change a mapping often depends on how ties are broken within a heuristic. A tie in a resource allocation heuristic is when a heuristic must choose from two equally good solutions, i.e., the heuristic determines both task assignments are the best possible task assignments. Two types of methods to break ties will be considered for this study. The first method is to break ties deterministically, e.g., the task and machine with the lowest identification number are chosen. The second method is to break ties randomly, e.g., if multiple machines are tied each will have a an equal probability of being chosen.

In the following section, we describe three common heuristic techniques where, if ties are broken deterministically, the iterative approach will not change the mapping. We will also formalize the properties that allow this to occur. In Sect. 4, we present four heuristics from the literature where the iterative approach may change the mapping for better or for worse regardless of how ties are broken.

3 Heuristics that will not improve with the iterative approach

3.1 Minimum execution time (MET) with deterministic tie breaking

The details of the Minimum Execution Time (MET) heuristic [2, 9, 20] are shown in Fig. 1. The MET heuristic will not change its mapping from iteration to iteration. The generalized proof of why this occurs will be presented in Sect. 3.5.

3.2 Minimum completion time with deterministic tie breaking (MCT)

The procedure to implement the *Minimum Completion Time* (MCT) heuristic [2, 9, 20] is shown in Fig. 2. With the iterative approach, the individual completion time for each machine does not improve over iterative mappings if ties are broken deterministically (will be demonstrated in Sect. 3.5).

3.3 Min–Min with deterministic tie breaking

The *Min–Min* heuristic [16, 20, 22, 23, 25, 27, 45] is a two-phase greedy heuristic. The procedure for this heuristic is given in Fig. 3. The performance of the Min–Min heuristic using the iterative approach will depend on the method used to break ties. If the ties are broken deterministically, the individual completion times for each machine do not improve (will be demonstrated in Sect. 3.5).

1. A task list is generated that includes all unmapped tasks in a given arbitrary order.
2. The first task in the list is assigned to its minimum completion time machine (machine ready time plus estimated computation time of the task on that machine).
3. The task selected in step 2 is removed from the task list.
4. The ready time of the machine on which the task is assigned is updated.
5. Steps 2–4 are repeated until all the tasks have been mapped.

Fig. 2 Procedure for using MCT to generate a resource allocation

1. A task list is generated that includes all the tasks as unmapped tasks.
2. For each task in the task list, the machine that gives the task its minimum completion time (first "Min") is determined (ignoring other unmapped tasks).
3. Among all task-machine pairs found in 2, the pair that has the minimum completion time (second "Min") is determined.
4. The task selected in 3 is removed from the task list and is assigned to the paired machine.
5. The ready time of the machine on which the task is mapped is updated.
6. Steps 2–5 are repeated until all tasks have been mapped.

Fig. 3 Procedure for using Min–Min to generate a resource allocation

3.4 Generalized completion time function and iteration invariant heuristics

The MET, MCT, and Min–Min heuristics are minimizing very similar performance features. We will relate the different performance functions by a generalized one based on machine ready time and task execution time. Let $RT_{k,n}(m)$ be the ready time of machine m at the nth *mapping event* (assignment of a task to a machine) of the kth iteration, and $ETC(t, m)$ be the estimated time to compute task t on machine m. A generalized completion time (GCT) function of task t on machine m (where λ and η are arbitrary values) is

$$GCT(t, m, n, k) = \lambda \cdot ETC(t, m) + \eta \cdot RT_{k,n}(m). \tag{1}$$

We can then define the completion time, CT, of a new task t on machine m with (1) and $\lambda = \eta = 1$. For both MCT and Min–Min, the values of λ, η are equal to 1, the difference between the set of machines considered for the MCT and Min–Min heuristics. For the MET, the value of λ is equal to 1, and η is equal to 0. For GCT functions (1), the ETC and *initial* ready times ($RT_{k,1}(m)$) do not vary across iterations (i.e., $RT_{1,1}(m) = RT_{2,1}(m) = \cdots = RT_{M,1}(m)$).

We define an Iteration Invariant Heuristic (IIH) as a heuristic whose mapping will not change across all iterations. Let $T_{k,n}$ be the set of mappable tasks and $M_{k,n}$ be the set of available machines at the nth mapping event in the kth iteration. We will show in the next section that a specific type of IIH (which encompasses MCT, MET, and

Min–Min) can be defined that at every mapping event n a task ($t_{\min}$) is assigned to machine $m_{\min}$ (with an argmin[1] approach), where

$$t_{\min}, m_{\min} = \underset{t \in T_{k,n}, m \in M_{k,n}}{\operatorname{argmin}} \; GCT(t, m, n, k). \tag{2}$$

It follows that at each mapping event these heuristics assign one task to one machine.

Additionally, this type of IIH (*GCT IIH*) breaks ties deterministically, i.e., they will always pick the same pair. In particular, we select the $t_{\min}$ with the lowest task identification number, and if necessary, the lowest numbered $m_{\min}$.

3.5 Properties of iteration invariant heuristics

In GCT IIHs, the assignment of tasks to machines does not change across iterations, and thus these heuristics are not well suited for use with the iterative approach, that is, no improvement is made over the original mapping. To design heuristics that can cause the mapping to change with the iterative approach, it is important to understand the properties of GCT IIHs.

Consider the ETC and the example resource allocation done by the Min–Min heuristic with three tasks (t_1, t_2, t_3) and three machines (m_1, m_2, m_3) shown in Fig. 4. We can observe that for the Min–Min heuristic the relative order in which tasks are assigned to machines does not change, i.e., if a task $\Gamma_{k,i}$ (ith task assigned in iteration k) was assigned before a task $\Gamma_{k,j}$ ($i < j$) then $\Gamma_{k,i}$ is always assigned before $\Gamma_{k,j}$ in every iteration, assuming both tasks are available for mapping. This property will be proved later in this sub-section.

We cannot use n to compare the assignment of a task across the kth and $(k+1)$th iterations, because the n in the $(k+1)$th iteration could represent the mapping event of different task. Assume task t is available for mapping at iterations k and $k+1$, i.e., it is not on the makespan machine for iterations $z \leq k$. To compare across iterations, we use the function $\omega(k, t)$ to represent the mapping event where task t is assigned in the kth iteration, i.e., by definition $n = \omega(k, \Gamma_{k,n})$. For convenience, we will define $\hat{n} = \omega(k+1, \Gamma_{k,n})$ to compare the mapping of $\Gamma_{k,n}$ in the kth and the $(k+1)$th iteration (at the nth and $\hat{n}$th mapping events, respectively). Thus, $\Gamma_{k,n}$ is equal to $\Gamma_{k+1,\hat{n}}$ by definition.

We will show that if $\Gamma_{k,n}$ is mapped to machine $\beta_{k,n}$ in the kth iteration, i.e.,

$$\Gamma_{k,n}, \beta_{k,n} = \underset{t \in T_{k,n}, m \in M_{k,n}}{\operatorname{argmin}} \; GCT(t, m, n, k), \tag{3}$$

and $\Gamma_{k+1,\hat{n}}$ (recall that $\Gamma_{k,n} = \Gamma_{k+1,\hat{n}}$) is mapped to machine $\beta_{k+1,\hat{n}}$ in the $(k+1)$th iteration, i.e.,

$$\Gamma_{k,n}, \beta_{k+1,\hat{n}} = \underset{t \in T_{k+1,\hat{n}}, m \in M_{k+1,\hat{n}}}{\operatorname{argmin}} \; GCT(t, m, \hat{n}, k+1), \tag{4}$$

[1] argmin: returns the arguments at which a function attains its minimum value.

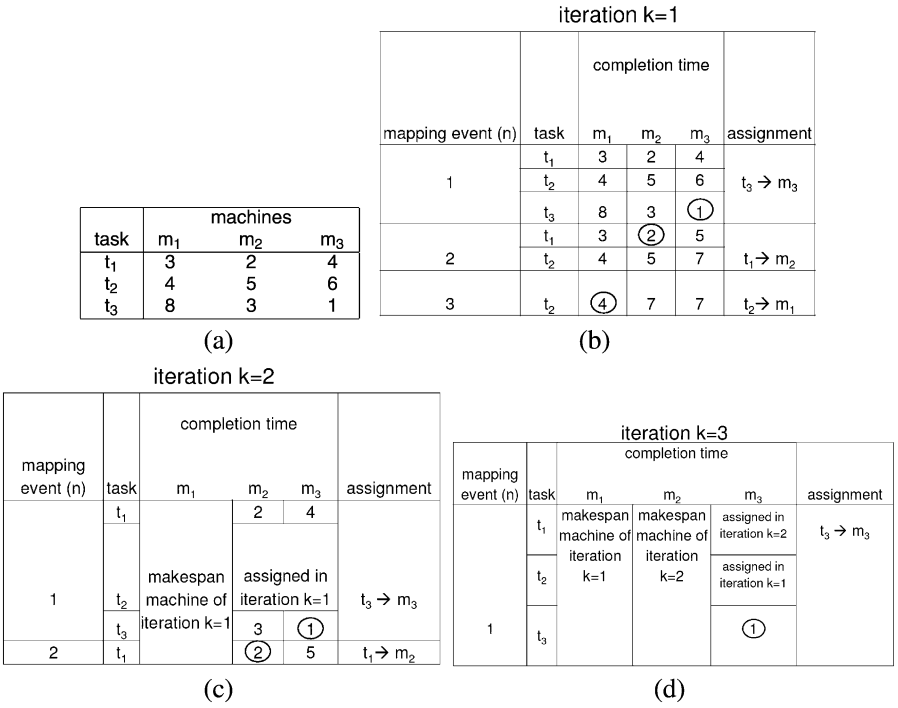


Fig. 4 Example of a resource allocation using the Min–Min heuristic. In (**a**), the ETC matrix used for this example is shown. The allocation of the first iteration is shown in (**b**), the second iteration in (**c**), and the final iteration in (**d**). This example illustrates that iterations 2 and 3 do not change the original mapping

then $\beta_{k+1,\hat{n}} = \beta_{k,n}$. The relationship of $T_{k,n}$ to $T_{k+1,\hat{n}}$, and $M_{k,n}$ to $M_{k+1,\hat{n}}$ is an important aspect of GCT IIHs. Let us define the makespan machine of the kth iteration as μ_k, and the set of tasks assigned to the makespan machine in the kth iteration as T_{μ_k}. Between the sets of tasks $T_{k,n}$ and $T_{k+1,\hat{n}}$ two conditions are needed for the mapping to be iteration invariant:

$$T_{k+1,\hat{n}} \subseteq T_{k,n}, \quad \text{and} \tag{5}$$

$$T_{k,n} - T_{k+1,\hat{n}} \subseteq \{T_{\mu_k}, \varnothing\}. \tag{6}$$

Two additional conditions for the relationship between $M_{k,n}$ and $M_{k+1,n_{k+1}}$ are:

$$M_{k+1,\hat{n}} \subset M_{k,n}, \quad \text{and} \tag{7}$$

$$M_{k,n} - M_{k+1,\hat{n}} = \{\mu_k\}. \tag{8}$$

A GCT IIH is a heuristic that has the properties shown in (1), (5)–(8), and breaks ties deterministically. The proof that a mapping generated by GCT IIHs will not change across iterations is shown in Theorem 1 using Lemma 1. Lemma 1 states that if ready times are identical at the nth mapping event of the kth and the $\hat{n}$th mapping event of the $(k+1)$th iteration, then the task/machine assignment will be the

n	mapping event of the kth iteration
$RT_{k,n}(m)$	ready time for machine m at the nth mapping event of the kth iteration
$ETC(t,m)$	estimated time to compute task t on machine m
$GCT(t,m,n,k)$	$GCT(t,m,n,k) = \lambda \cdot ETC(t,m) + \eta \cdot RT_{k,n}(m)$
$T_{k,n}$	set of mappable tasks at the nth mapping event of the kth iteration
$M_{k,n}$	set of available machines at the nth mapping event of the kth iteration
T_{μ_k}	tasks assigned to the makespan machine in the kth iteration
$\Gamma_{k,n}$	nth task mapped at the kth iteration
$\beta_{k,n}$	machine to which $\Gamma_{k,n}$ is mapped
μ_k	makespan machine of iteration k
$\omega(k, \Gamma_{k,n})$	represents the mapping event where task $\Gamma_{k,n}$ is assigned in the kth iteration
$\hat{n}$	$\hat{n} = \omega(k+1, \Gamma_{k,n})$
$\Gamma_{k+1,\hat{n}}$	$\hat{n}$th task mapped at the $(k+1)$th iteration, by definition it is equal to $\Gamma_{k,n}$
$\beta_{k+1,\hat{n}}$	machine to which $\Gamma_{k+1,\hat{n}}$ is mapped (proved in Theorem 1 to be equal to $\beta_{k,n}$)

Fig. 5 Glossary of notation

same. Theorem 1 uses the result from Lemma 1 to show that the mapping will not change across iterations. A glossary of relevant notation is shown in Fig. 5.

Lemma 1 *Consider tasks not assigned to μ_k for a GCT IIH. If the ready times are identical at the nth mapping event of the kth iteration and the $\hat{n}$th mapping event of the $(k+1)$th iteration then the same task/machine assignment is chosen in both iterations.*

Proof From (2), we know that the GCT function of the task/machine pairs in the kth iteration will have the following property:

$$GCT(\Gamma_{k,n}, \beta_n, \hat{n}, k) \leq GCT(t,m,n,k) \quad \forall t \in T_{k,n},\ \forall m \in M_{k,n}, \tag{9}$$

and the $(k+1)$th iteration has the following property:

$$GCT(\Gamma_{k+1,\hat{n}}, \beta_{k+1,\hat{n}}, \hat{n}, k+1) \leq GCT(t,m,\hat{n},k+1) \quad \forall t \in T_{k+1,\hat{n}},\ \forall m \in M_{k+1,\hat{n}}. \tag{10}$$

Given the assumption that $RT_{k+1,\hat{n}}(m) = RT_{k,n}(m)$, and the ETC values are fixed it follows that:

$$GCT(t,m,\hat{n},k+1) = GCT(t,m,n,k) \quad \forall t \in T_{k+1,\hat{n}},\ \forall m \in M_{k+1,\hat{n}}. \tag{11}$$

Because $\{\Gamma_{k+1,\hat{n}},\beta_{k+1,\hat{n}}\}$ and $\{\Gamma_{k,n},\beta_{k,n}\}$ are the task-machine pairs that minimize the GCT function in the $(k+1)$th and kth iterations, and ties are broken deterministically then $\beta_{k+1,\hat{n}} = \beta_{k,n}$, i.e., respectively

$$\underset{t\in T_{k+1,\hat{n}}, m\in M_{k+1,\hat{n}}}{\operatorname{argmin}} GCT(t,m,\hat{n},k+1) = \underset{t\in T_{k,n}, m\in M_{k,n}}{\operatorname{argmin}} GCT(t,m,n,k), \quad (12)$$

$$\Gamma_{k+1,\hat{n}}, \beta_{k+1,\hat{n}} = \Gamma_{k,n}, \beta_{k,n}. \quad (13)$$

□

Theorem 1 *The mapping generated by a GCT IIH will not change across iterations.*

Proof (Inductive hypothesis). Consider the nth task mapped by a GCT IIH in the kth iteration ($\Gamma_{k,n}$). Let $P(n)$ be the statement that $\Gamma_{k,n}$ will have the same assignment in both the kth iteration and the $(k+1)$th iteration. For the basis and inductive steps, there are two cases to consider: the case when the task is mapped to the makespan machine in the kth iteration (μ_k) and the case when the task is not mapped to machine μ_k.

To prove $P(n)$ is true $\forall n \geq 1$ we need to prove:

(1) $P(1)$ is true
(2) $(\forall n)$ [$P(n)$ is true for all r, $1 \leq r \leq n \Rightarrow P(n+1)$ is true].

Basis step: Prove that $P(1)$ is true.

Case 1: The task $\Gamma_{k,1}$ was assigned to machine μ_k. Because $\Gamma_{k,1}$ was assigned to μ_k, it will remain assigned to μ_k in the $(k+1)$th iteration because the makespan machine is removed from consideration; thus, the $P(1)$ statement is true for case 1 of the basis step.

Case 2: The *initial* ready times of the kth and the $(k+1)$th iterations are identical. Therefore, using Lemma 1, we can prove case 2 is true. Thus the $P(1)$ statement is true for case 2 of the basis step.

Inductive step: For the inductive step assume that $\forall r$ $P(1 \leq r \leq n)$ is true and prove $P(n+1)$ is true. The assumption that $P(1 \leq r \leq n)$ is true implies that ready times, when $\Gamma_{n+1,k}$ is mapped, are equal in the kth mapping and the $(k+1)$th mapping.

Case 1: The $(n+1)$th task to be mapped in the kth iteration was assigned to machine μ_k. Because $\Gamma_{n+1,k}$ was assigned to μ_k, it will remain assigned to μ_k in the $(k+1)$th iteration because the makespan machine is removed from consideration; thus, the $P(n+1)$ statement is true for case 1 of the inductive step.

Case 2: Because of the inductive assumption, all the previous task to machine assignments are identical between the kth and $(k+1)$th iteration. This implies that the ready times, for all machines in both $M_{k,n}$ and $M_{k+1,\hat{n}}$ are the same. Therefore, using Lemma 1, we prove case 2 is true. Thus, the $P(n+1)$ statement is true for case 2 of the inductive step. This proves that all assignments remain identical between iterations. □

It is interesting to note that the GCT function (when used with (2)) specifies a subset of possible functions that define IIHs. In general, the function used to build a mapping does not have to be a linear combination of execution and ready times (e.g., the function could be a polynomial function of execution and ready times and still be an IIH). In this paper, we only use GCT functions because all the heuristics considered in this study use completion time to build a mapping.

GCT IIH such as MET, MCT, and Min–Min, are indeed IIH when ties are broken deterministically and the mappings do not change with the iterative approach. It is important to note that in practice, depending on precision of completion times, ties can be rare. In the next section, we explore heuristics that may improve the iterative approach.

4 Heuristics that may improve with the iterative approach

4.1 MET, MCT, and Min–Min with random tie breaking

4.1.1 Overview

The MET, MCT, and Min–Min heuristics are GCT IIH; however, a requirement of GCT IIH is to select deterministically from solutions that are equally good. If there are multiple argmin solutions to (12) and (13), and one argmin solution is randomly selected then there can be no guarantee that the properties of GCT IIHs apply. Therefore, it is possible for the solution to change from one iteration to the next. As an example of this, we will consider the MCT heuristic.

4.1.2 Example of reducing the makespan among non-makespan machines with MCT

This is the same MCT as described in Sect. 3.2 except that ties are broken randomly. For this example of improvement, the initial ready times are 0. Consider the following mapping order for the MCT heuristic: t_1, t_2, t_3, and t_4. The ETC matrix used for this example is shown in Fig. 6a. In Figs. 6c and 6b, the term "Machine CT" denotes the Machine completion time of the task in the corresponding row, based on previous task assignments. This example relies on a tie in the mapping of task t_3 between m_2 and m_3. In the original mapping, t_3 is assigned to m_2.

In the original mapping shown in Figs. 6b and 6d, t_3 is assigned to machine m_3. However, in the first iterative mapping shown in Figs. 6c and 6e, t_3 is assigned to machine m_2. Thus, the makespan of the remaining machines m_2 and m_3 went from 5 in the original mapping to 4 in the first iterative mapping.

4.1.3 Example of increasing overall makespan with MCT

For this example, the initial ready times of machines are 0. Consider the following mapping order for the MCT heuristic: t_1, t_2, t_3, and t_4. The ETC matrix used for this example is shown in Fig. 7a.

task	machine (m_1, m_2, m_3)
t_1	8, 9, 10
t_2	3, 2, 5
t_3	5, 2, 4
t_4	3, 3, 1

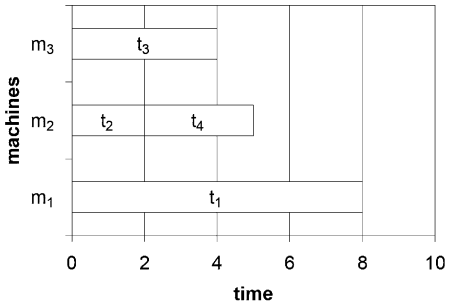
(a)

assignment	machine CT (m_1, m_2, m_3)
$t_1 \rightarrow m_1$	**8**, 9, 10
$t_2 \rightarrow m_2$	11, **2**, 5
$t_3 \rightarrow m_3$	13, **4**, **4**
$t_4 \rightarrow m_2$	11, **5**, **5**

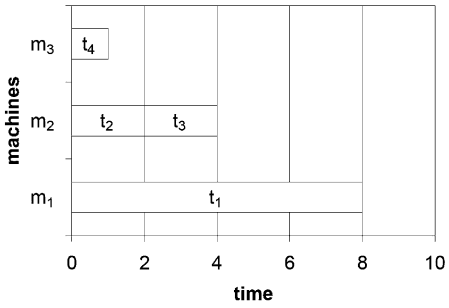
(b)

assignment	machine CT (m_2, m_3)
$t_2 \rightarrow m_2$	**2**, 5
$t_3 \rightarrow m_2$	**4**, **4**
$t_4 \rightarrow m_3$	7, **1**

(c)



(d)



(e)

Fig. 6 Example of makespan among non-makespan machines being reduced with MCT: (**a**) ETC matrix, (**b**) details of original mapping, (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping

task	machines (m_1, m_2, m_3)
t_1	3, 2, 2
t_2	4, 1, 4
t_3	5, 4, 3
t_4	4, 5, 4

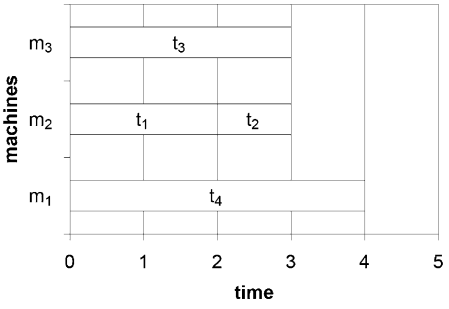
(a)

assignment	machine CT (m_1, m_2, m_3)
$t_1 \rightarrow m_2$	3, **2**, **2**
$t_2 \rightarrow m_2$	4, **3**, 4
$t_3 \rightarrow m_3$	5, 7, **3**
$t_4 \rightarrow m_1$	**4**, 8, 7

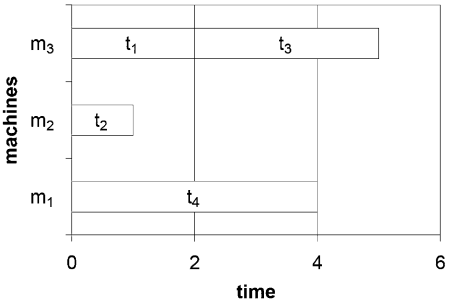
(b)

assignment	machine CT (m_2, m_3)
$t_1 \rightarrow m_3$	**2**, **2**
$t_2 \rightarrow m_2$	**1**, 6
$t_3 \rightarrow m_3$	**5**, **5**

(c)



(d)



(e)

Fig. 7 Example of overall makespan increasing for MCT: (**a**) ETC matrix, (**b**) details of original mapping, (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping

1. An initial population of mappings is generated.
2. The mappings in the population are ordered based on makespan.
3. While the stopping criteria is not met:
   (a) An intermediate population is created using a selection mechanism (i.e., rank based selection).
   (b) Two chromosomes in the intermediate population are probabilistically selected as parents for crossover.
      i A random cut-off point is generated.
      ii The machine assignments of the tasks below the cut-off point are exchanged.
   (c) Each offspring has a probability of being mutated. For the chosen chromosome, a random task is chosen and its machine assignment is arbitrarily modified.
   (d) The resultant population of the crossover and mutation replaces the original population. Because of elitism the best chromosome remains in the population.
4. The best solution is output.

Fig. 8 Summary of one possible procedure that can be used to implement a GA

The original mapping is shown in Figs. 7b and 7e. This example relies on a tie in the mapping of task t_1 between m_2 and m_3. In the original mapping, t_1 is assigned to m_2. The resource allocations for the first iterative mapping are shown in Figs. 7c and 7d. For the first iterative mapping, we assign task t_1 to m_3. This change causes the makespan to become greater than that of the original full mapping.

4.2 Genetic Algorithm (GA)

Genetic Algorithms (GAs) have been shown to work well for numerous problem domains, including resource allocation and job shop scheduling. GAs use *chromosomes* to represent possible solutions, e.g., all tasks and the machines to which they are assigned. A GA has a *population* that consists of multiple chromosomes and typically has two operators to search for better solutions. The first operator is *crossover*, an operator that combines two chromosomes to produce two new chromosomes. The second operator is *mutation*; this operator may randomly change tasks assignments within a chromosome. A GA can be summarized by the procedure shown in Fig. 8. This variation of a GA is adapted from [38].

For each iteration (of the iterative approach), the best mapping found by GA in the previous iteration, excluding the makespan machine and the tasks assigned to it, is included in current population (i.e., it is used as a "seed"). Using elitism in the GA guarantees that the final mapping is either the seeded mapping or a mapping with a smaller makespan, among the machines considered in the current iteration. Thus, for GA the iterative approach will result in either an improvement or no change. The iterative technique also works with steady state GAs such as Genitor [44], because Genitor uses ranking to keep the best chromosome.

1. A task list is generated that includes all unmapped tasks in a given arbitrary order.
2. A subset is formed by picking the $M \cdot (\frac{k}{100})$ best machines based on the execution times for the task.
3. The task is assigned to a machine that provides the earliest completion time in the subset.
4. The task is removed from the unmapped task list.
5. The ready time of the machine on which the task is assigned is updated.
6. Steps 2–5 are repeated until all tasks have been assigned.

Fig. 9 Procedure for using K-percent Best to generate a resource allocation

4.3 K-Percent best algorithm

4.3.1 Overview of K-percent best algorithm

The *K-Percent Best Algorithm* [2, 20, 32] (KPB) is a hybrid of MET and MCT. The procedure to implement K-percent Best is shown in Fig. 9. If the percentage is (100/number of machines)% then the K-percent Best is identical to the MET heuristic, however, if the percentage is 100% then it is identical to the MCT heuristic. An example of the K-percent Best algorithm with the iterative approach improving the makespan among non-makespan machines, and in contrast, an example of increasing the overall makespan, can be found even for cases when no special consideration is used to break ties. The percentage K for a given environment is found by experimentation.

The K-percent best heuristic uses the MCT heuristic that Theorem 1 proved would not change mappings unless ties were broken randomly, however its mappings do change. This introduces an important question: how can a heuristic that uses MCT to do its assignments change when ties are broken deterministically? The characteristic that K-percent Best has is that it limits the number of machines considered for assignment depending on the size of the set of machines that are available at that iteration. This implies that if the percentage was such that only one machine is selected, i.e., KPB would work like MET or 100% (K-percent Best would work like MCT) the mapping would not change across iterations. If the value of K is between these two extreme values then the KPB will violate (8). Thus, despite the MCT part, the K-percent Best is not a GCT IIH. For values of K that result in a group of machines whose size is greater than one, the reduction in machines after an iteration can cause the minimum completion time machine in the original mapping to be left out of the machines the heuristic allows for assignment in the first iterative mapping.

4.3.2 Example of reducing the makespan among non-makespan machines

For this example, the initial ready times of machines are 0. Consider the ETC matrix shown in Fig. 10a for three machines and the following mapping order: t_1, t_2, t_3, and t_4. The percent for this example is set to 70%. This implies that for the original mapping the best two machines are considered for mapping, and for the first iterative mapping only one machine is considered. Allowing only one machine forces

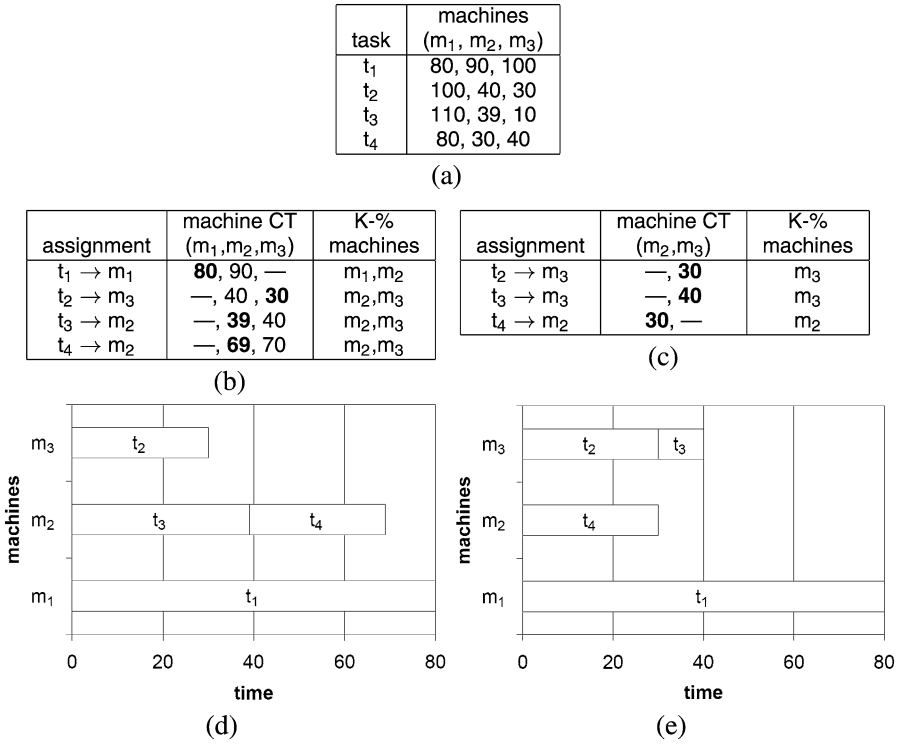


Fig. 10 Example of makespan among non-makespan machines being reduced for K-percent best: (**a**) ETC matrix, (**b**) details of original mapping, (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping

the K-percent Best Algorithm to map tasks to the MET machine in the first iterative mapping.

The original mapping using the K-percent Best heuristic is shown in Figs. 10b and 10d. The result of the first iterative mapping is shown in Figs. 10c and 10e. In the original mapping, t_3 is assigned to m_2. However, in the first iterative mapping t_3 is assigned to m_3. This change in mapping is because the K-percent Best Algorithm considers only one machine for resource allocation in the first iterative mapping, while it had used two machines in the original mapping. The makespan for machines m_2 and m_3 is reduced from 69 in the original mapping to 40 in the first iterative mapping.

4.3.3 Example of increasing makespan

For this example, the initial ready times of machines are 0. Consider the ETC matrix shown in Fig. 11a for three machines and the following mapping order: t_1, t_2, t_3, t_4, and t_5. The percent, for this example, is set to 70%. This implies that for the original mapping the best two machines are used for mapping, and for the first iterative mapping only one machine is considered. This is the critical difference between the first iterative mapping and the original mapping. Considering one machine in the

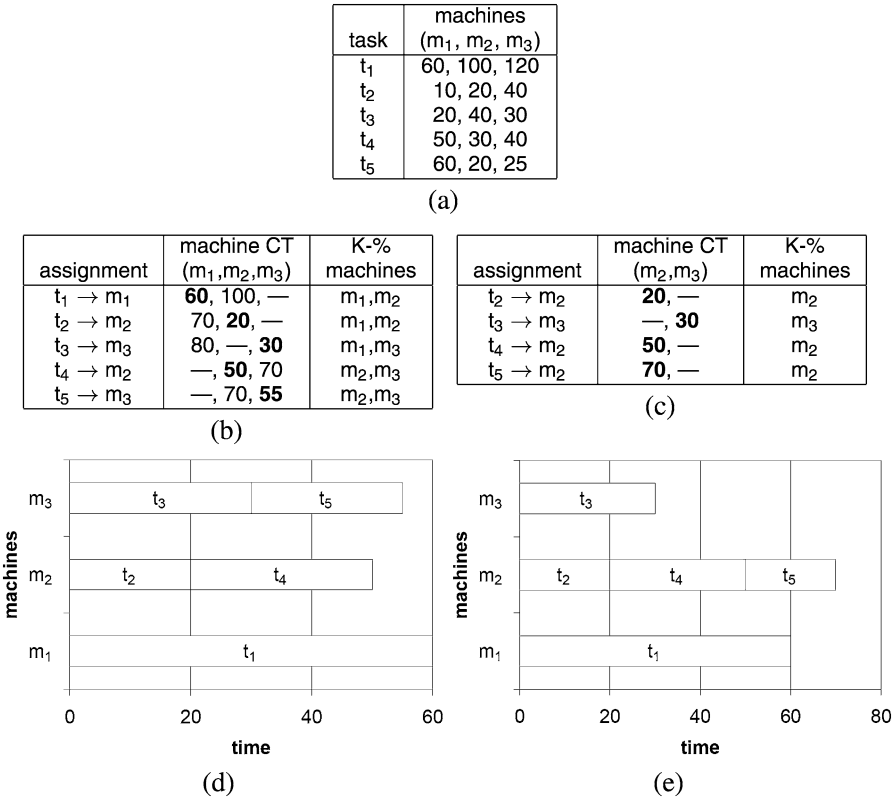


Fig. 11 Example of makespan increasing for K-percent Best: (**a**) ETC matrix, (**b**) details of original mapping, (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping

first iterative mapping forces the K-percent Best algorithm to perform like the MET heuristic.

The results of the original mapping are shown in Figs. 11b and 11d. The results of the first iterative mapping are shown in Figs. 11c and 11e. In the original mapping t_5 is assigned to m_3; however, in the first iterative mapping, task t_5 is assigned to m_2. The overall makespan increased from 60, in the original mapping, to 70 in the first iterative mapping. This is because the number of K-% Best machines went down from two to one. This example shows that for K-percent Best Algorithm the makespan can increase, specifically *overall makespan*.

4.4 Switching Algorithm (SWA)

4.4.1 Overview of SWA

The *Switching Algorithm* (SWA) is adapted from [32]. It was designed for use in dynamic environments, but can be used in static environments as well. The switching algorithm is a hybrid of the MET and MCT heuristics. The procedure for SWA is

1. A task list is generated that includes all unmapped tasks in a given arbitrary order.
2. The first task in the list is assigned using the MCT heuristic.
3. The load balance index is calculated for the system (minimum ready time / maximum ready time over all machines).
4. The heuristic used to map the task is determined as follows:
   i If the load balance index $\geq$ "high threshold," MET is selected to map the next task.
   ii If the load balance index $\leq$ "low threshold," MCT is selected to map the next task.
   iii Otherwise, the current heuristic remains selected.
5. Steps 3–4 are repeated until all tasks have been mapped.

Fig. 12 Procedure for using SWA to generate a resource allocation

shown in Fig. 12. The high and low thresholds are determined experimentally. Examples of the SWA with the iterative approach both improving the makespan among non-makespan machines (see Fig. 13) and increasing the overall makespan (Fig. 14) can be found even for cases when no special consideration is used to break ties.

The SWA, like the K-percent Best, uses heuristics (MCT and MET) that we proved would not change mappings if ties were broken deterministically (Theorem 1). However, in SWA the heuristic used to assign a task in one iteration (MET or MCT) can change when assigning the same task in the next iteration (e.g., t_4 in Figs. 14b and 14c). In SWA, the choice of using MCT or MET to map as task at a given mapping event depends on the load balance between the machine with the largest computation time, and the machine with the smallest computation time. In general, the calculation of the load balance index when a given task is mapped will change from iteration i to $i + 1$. As a result, different heuristics may be used to map that task at different iterations. Thus, the objective function of this heuristic cannot be written as a GCT with fixed λ and η (i.e., λ and η would change as the heuristic used changes between MCT and MET).

4.4.2 Example of reducing the makespan among non-makespan machines

Consider the ETC matrix shown in Fig. 13a and the following mapping order: t_1, t_2, t_3, t_4, t_5, and t_6. The initial ready times for all the machines is 0. SWA will switch from MCT to MET when the balance index (BI) is greater than or equal to the high threshold of 0.8, and will switch from MET to MCT when the BI is less than or equal to the low threshold of 0.7.

The original mapping for the SWA is shown in Figs. 13b and 13d. In the original mapping, the BI never exceeds 0.8; therefore, all mappings are done using the MCT heuristic.

The first iterative mapping for the SWA example of improvement is shown in Figs. 13c and 13e. The difference is that t_4 is assigned to m_3 in the first iterative mapping (instead of m_2). This was different because t_4 was assigned with the MCT heuristic in the original mapping, and in the first iterative mapping t_4 was assigned

task	machines (m_1, m_2, m_3)
t_1	40, 70, 50
t_2	60, 50, 80
t_3	70, 90, 60
t_4	100, 30, 21
t_5	110, 30, 30
t_6	60, 120, 110

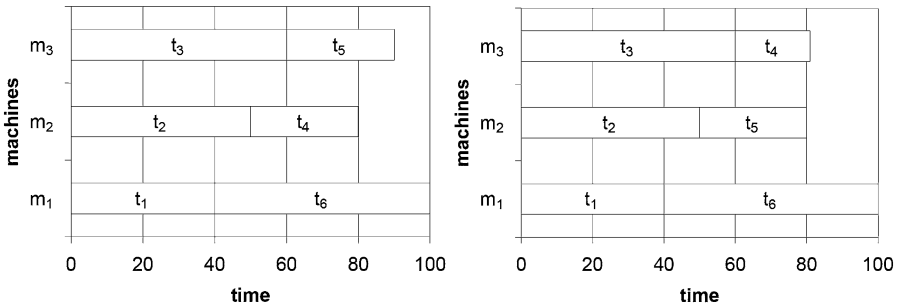
(a)

assignment	machine CT (m_1,m_2,m_3)	ready times (m_1,m_2,m_3)	BI	mapper
$t_1 \rightarrow m_1$	**40**, 70, 50	0, 0, 0	x	MCT
$t_2 \rightarrow m_2$	100, **50**, 80	*40*, 0, *0*	0	MCT
$t_3 \rightarrow m_3$	110, 90, **60**	*40*, 50, *0*	0	MCT
$t_4 \rightarrow m_2$	140, **80**, 81	*40*, 50, *60*	2/3	MCT
$t_5 \rightarrow m_3$	150, 110, **90**	*40*, *80*, 60	1/2	MCT
$t_6 \rightarrow m_1$	**100**, 200, 200	*40*, 80, *90*	4/9	MCT

(b)

assignment	machine CT (m_2,m_3)	ready times (m_2,m_3)	BI	mapper
$t_2 \rightarrow m_2$	**50**, 80	0, 0	x	MCT
$t_3 \rightarrow m_3$	140, **60**	*50*, *0*	0	MCT
$t_4 \rightarrow m_3$	—, **21**	*50*, *60*	5/6	MET
$t_5 \rightarrow m_2$	**80**, 111	*50*, *81*	50/81	MCT

(c)



(d) (e)

Fig. 13 Example of makespan among non-makespan machines being reduced with SWA: (**a**) ETC matrix, (**b**) details of original mapping (largest and smallest RTs are in italic for a given mapping event), (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping (for t_4 mapping event MET is used and CTs do not matter)

using MET. The makespan among the remaining machines (m_2 and m_3) was reduced from 90 to 81.

4.4.3 Example of increasing makespan

Consider the ETC matrix shown in Fig. 14a and the following mapping order: t_1, t_2, t_3, t_4, and t_5. The initial ready times for all the machines is 0. SWA will switch from MCT to MET when the BI is greater than or equal to the high threshold of 0.5 and

task	machines (m_1, m_2, m_3)
t_1	60, 100, 120
t_2	10, 20, 40
t_3	50, 25, 40
t_4	60, 30, 25
t_5	40, 20, 10

(a)

assignment	machine CT (m_1,m_2,m_3)	ready times (m_1,m_2,m_3)	BI	mapper
$t_1 \rightarrow m_1$	**60**, 100, 120	0, 0, 0	x	MCT
$t_2 \rightarrow m_2$	70, **20**, 40	*60*, 0, *0*	0	MCT
$t_3 \rightarrow m_3$	110, 45, **40**	*60*, 20, *0*	0	MCT
$t_4 \rightarrow m_2$	120, **50**, 65	*60*, *20*, 40	1/3	MCT
$t_5 \rightarrow m_3$	100, 70, **50**	*60*, 50, *40*	2/3	MET

(b)

assignment	machine CT (m_1,m_2,m_3)	ready times (m_1,m_2,m_3)	BI	mapper
$t_2 \rightarrow m_2$	**20**, 40	0, 0	x	MCT
$t_3 \rightarrow m_3$	45, **40**	*20*, *0*	0	MCT
$t_4 \rightarrow m_3$	—, **65**	*20*, *40*	1/2	MET
$t_5 \rightarrow m_2$	**40**, 85	*20*, *65*	4/13	MCT

(c)

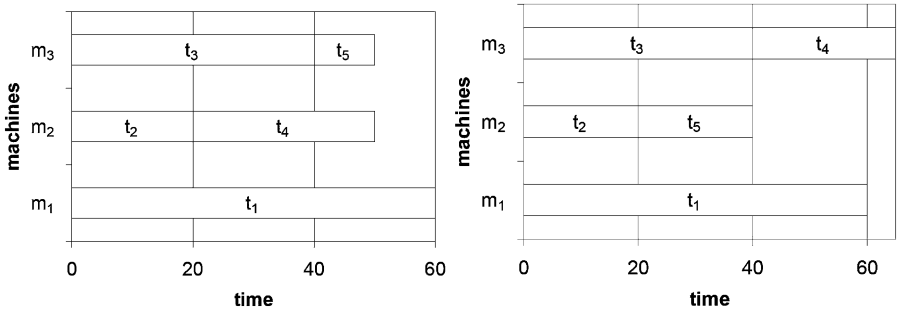


Fig. 14 Example of makespan increasing for SWA: (**a**) ETC matrix, (**b**) details of original mapping (largest and smallest RTs are in italic for a given mapping event), (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping (for t_4 mapping event MET is used and CTs do not matter)

will switch from MET to MCT when the BI is less than or equal to the low threshold of 0.4.

The original mapping is shown in Figs. 14b and 14d, and the first iterative mapping is shown in Figs. 14c and 14e. Task t_4 is assigned to m_3 (in the first iterative mapping) because the resource allocation of t_4 has a different BI. The makespan machine after the first iterative mapping is m_3 (i.e., the completion time of m_3 becomes greater than the completion time of m_1). The overall makespan increased from 60 in the original mapping to 65 in the first iterative mapping.

1. A task set (S) is generated that includes all unmapped tasks in a given arbitrary order.
2. While there are still unmapped tasks:
   i For each machine find the set of tasks that have their minimum completion time on this machine.
      (a) If the set of tasks is size one, then assign the corresponding task to the machine and remove the task from S.
      (b) If the size of the set of tasks is greater than one, then assign task with the highest sufferage value, and remove that task from S.
   ii The ready times for all machines are updated.

Fig. 15 Procedure for using Sufferage to generate a resource allocation

4.5 Sufferage algorithm

4.5.1 Overview of the Sufferage algorithm

The *Sufferage algorithm* [6, 13, 16, 19, 23, 25, 27, 32, 37, 39, 43] is shown in Fig. 15. In this context, the *sufferage value* of a task is the difference between its second smallest completion time among all machines and its smallest completion time among all machines. That is, it is the increase in completion time that occurs if the task cannot use its best machine but must use its second best machine instead. Thus, the Sufferage algorithm is a greedy algorithm that does a limited local search. An example of Sufferage with the iterative approach improving the makespan among non-makespan machines and, in contrast, an example of increasing the overall makespan, can be found even for cases when ties are broken deterministically.

In the Sufferage heuristic, the assignment of a task A to machine B depends on how much it would "suffer" if it was assigned to the machine where A has the second best completion time. The reason why the mapping generated by this heuristic can change is because the machine where A has the second best completion time in the original mapping could easily be the makespan machine. This implies the sufferage value for one or more tasks will be different, and thus change the mapping decision. The Sufferage heuristic is not a GCT IIH, because the function it uses to map cannot be written in the form of (1). Note that for the sufferage algorithm there are two types of ties (a) for a given task to a different machine both have the minimum completion time (step 2.i in Fig. 15); and (b) for a given machine two tasks may have the same sufferage values (step 2.i.b in Fig. 15).

4.5.2 Example of reducing the makespan among non-makespan machines

For this example of improvement, the initial ready times of machines are 0. Consider the ETC matrix shown in Fig. 16a. The original mapping is shown in Figs. 16b and 16d, and the first iterative mapping is shown in Figs. 16c and 16e. The difference between the original mapping and the first iterative mapping is that in the original mapping t_4 was assigned to m_1 in the first pass; however, in the first iterative mapping t_4 was assigned to m_2 in the second pass. This is because the sufferage for t_3

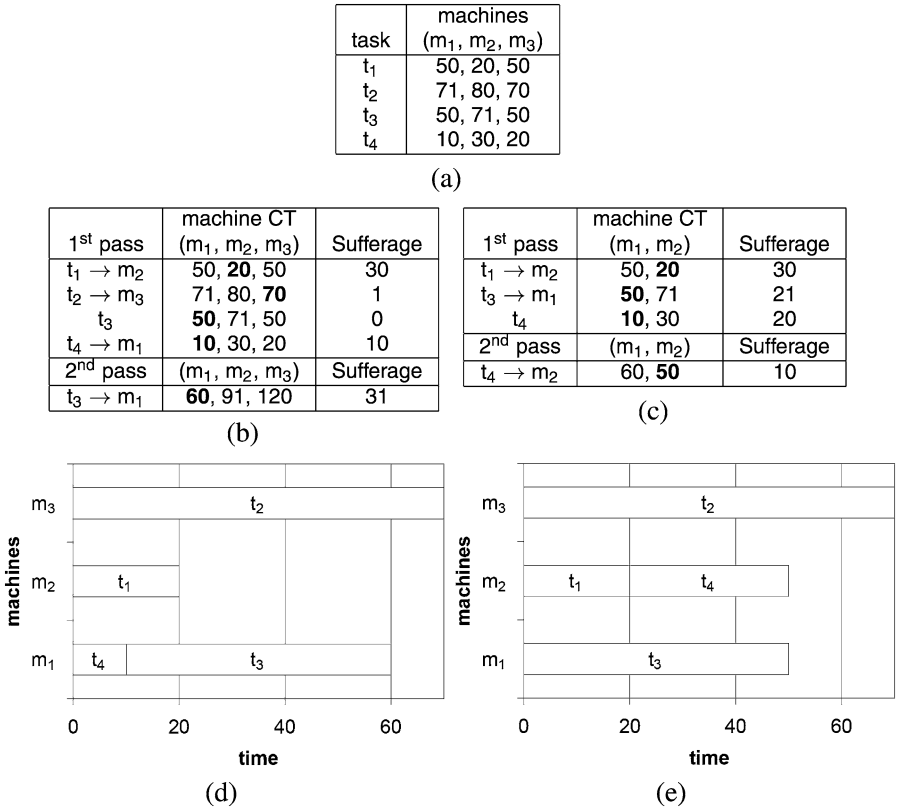


Fig. 16 Example of makespan among non-makespan machines being reduced for Sufferage: (**a**) ETC matrix, (**b**) details of original mapping, (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping

increased from 0 to 21 from the original mapping to the first iteration as a result of m_3 being removed. This caused t_3 (instead of t_4) to be mapped to m_1. The makespan for machines m_1 and m_2 is reduced from 60 in the original mapping to 50 in the first iterative mapping.

4.5.3 Example of increasing makespan

For this example, the initial ready times of machines are 0. Consider the ETC matrix shown in Fig. 17a. The original mapping is shown in Figs. 17b and 17d, and the first iterative mapping is shown in Figs. 17c and 17e. In the original mapping, t_2 is assigned to m_2 in the first pass. However, task t_2 is assigned to m_3 in the second pass of the first iterative mapping, because the sufferage of t_4 increased in the first iterative mapping as a result of m_1 being removed. This increase caused t_4 be assigned at a different pass of the heuristic. This in turn forces t_1 to be mapped in the 3rd pass, instead of the second pass, and to a different machine, i.e., m_2 resulting in larger makespan. The overall makespan increases from 100 in the original mapping to 108 in the first iterative mapping.

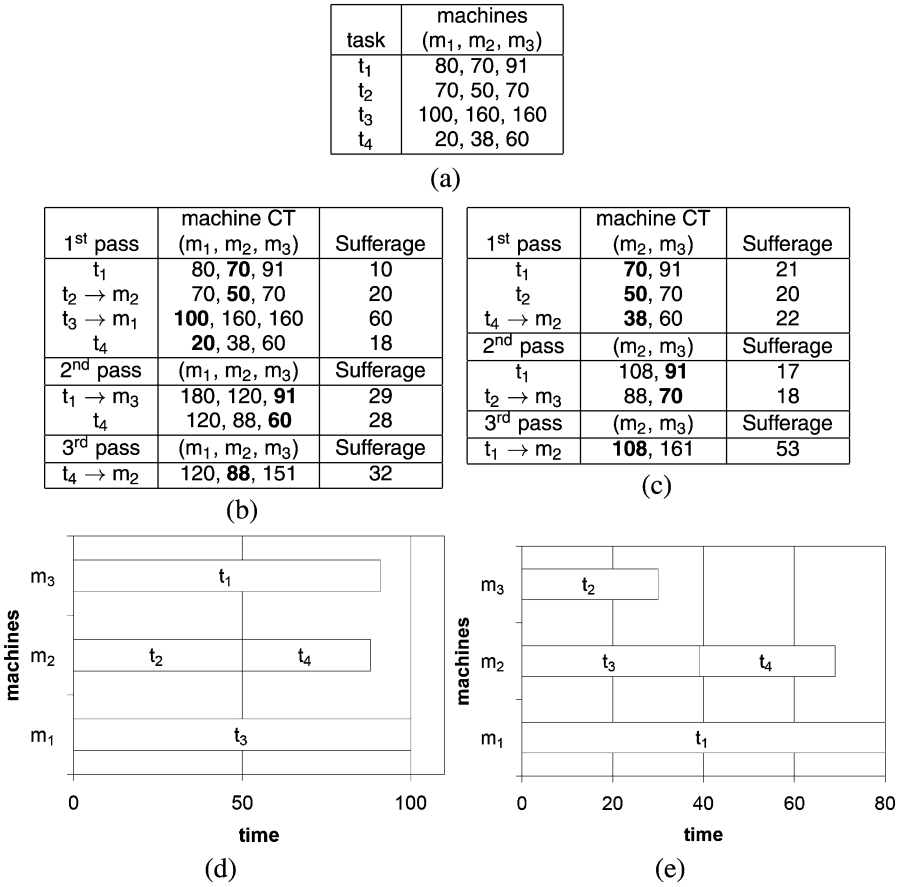


Fig. 17 Example of overall makespan increasing for Sufferage: (**a**) ETC matrix, (**b**) details of original mapping, (**c**) details of first iterative mapping, (**d**) graphical representation of original mapping, and (**e**) graphical representation of first iterative mapping

5 Analysis of characteristics of heuristics

In the analysis of the heuristics that use the iterative approach, we found that there are three main situations in which a mapping can change. The first situation is when a mapping will change from one iteration to the next if ties are broken randomly. This situation occurred with the Min–Min, MCT, and MET heuristics. If ties are instead broken deterministically, then these heuristics are GCT IIHs (as proved in Sect. 3.5). It is important to use these observations and apply them to heuristics that were not considered in this study. An example of a different heuristic that is also a GCT IIH is the Opportunistic Load Balancing (OLB) [9, 32, 35] heuristic. OLB assigns each task, in a given arbitrary order, to the machine with the smallest ready time. The OLB heuristic uses a GCT function with $\lambda = 0$ and $\eta = 1$ (recall that MET has $\lambda = 1$ and $\eta = 0$). There are other IIHs, and similar proofs can be derived for them.

The second situation is when a mapping can change from one iteration to the next independently of how ties are broken. This situation occurred with the K-percent Best, SWA, and Sufferage heuristics. The main characteristic of the second category of heuristics is that additional information, other than a GCT function, is used to determine assignments. Intuitively, this means that the assignment of task A to machine B depends on the state of the other available machines; however, the set of available machines changes from one iteration to the next. An additional example of this second type of heuristic is the Max–Min heuristic [5, 22]. In Max–Min, we find the machine that has the minimum completion time for each task and from these task-machine pairs we select the pair that has the maximum completion time. Consider a situation with multiple tasks and machines where t_1 has its minimum completion time on m_1 in the original mapping, and t_2 has its minimum completion time on m_2. In this example, the completion time of t_2 on m_2 is greater than the completion time of t_1 on m_1 ($GCT(t_1, m_1, 1, 1) < GCT(t_2, m_2, 1, 1)$). Therefore, t_2 gets assigned to m_2 ($t_2 \rightarrow m_2$). However, assume m_1 was the makespan machine of the original mapping and t_1 has a new minimum on m_2. This new minimum is greater than the completion time of assigning t_1 to m_1. This could cause $t_1 \rightarrow m_2$ to be greater than $t_2 \rightarrow m_2$ ($GCT(t_1, m_2, 1, 2) > GCT(t_2, m_2, 1, 2)$) causing t_1 to be assigned to m_2 and t_2 to be assigned to a different machine. Max–Min is not a GCT IIH because its task to machine assignment is done with the following equation:

$$\Gamma_{k,n}, \beta_{k,n} = \underset{t \in T_{k,n}}{\operatorname{argmax}} \left\{ \underset{m \in M_{k,n}}{\operatorname{argmin}} \left(GCT(t, m, n, k) \right) \right\}. \tag{14}$$

This equation is different from (2), therefore, it is not a GCT IIH.

The third situation is when a mapping will result in either an improvement or no change. The GA based approach was the only heuristic considered in this study where the overall makespan could not increase; however, no improvement can be guaranteed. This was accomplished using the best chromosome from the previous iteration and seeding it into the current iteration. This usage of seeding comes very naturally to the GA heuristic; however, the same concept can also be applied to the other heuristics in this study, e.g., KPB, SWA, Sufferage. If we have heuristics where the mapping can change (with the iterative approach), then we can compare the mapping of the non-makespan machines from iteration k and the mapping from iteration $k + 1$ and keep the mapping with the smallest makespan. This would cause the best solutions to be preserved across iterations, thus changing the mapping only if a better makespan is found.

6 Conclusions

An iterative approach for minimizing the finishing times of machines in a heterogeneous computing environment was proposed. The performance of several heuristics was analyzed for such an approach. The greedy heuristics studied (Min–Min, MCT, MET, Switching Algorithm, K-percent Best, and Sufferage) did not guarantee an improvement in the completion time among non-makespan machines.

The Min–Min, MCT, and MET heuristics with *deterministic tie breaking* were characterized as GCT IIHs, and were mathematically proven to not change mappings across iterations. However, the Min–Min, MCT, and MET heuristics *with random tie breaking* can change the makespan among remaining machines (i.e., improve or get worse, even causing the overall makespan to increase).

The Switching algorithm, K-percent Best, and Sufferage heuristics all produced mappings with the iterative approach where the makespan among remaining machines can increase or decrease. For MET, MCT, and Min–Min decisions are based on selecting the available machine with the minimal GCT value (1). In the Sufferage, SWA, K-percent Best heuristics, information in addition to the minimal GCT machine is used. For example, Sufferage also considers the second "best" machine.

The GA-based approach using elitism will keep the same mapping or produce a better mapping and, therefore, guarantees the overall makespan will not increase. Among the heuristics studied, it is the only one to guarantee improvement or no change. This is because of elitism. Thus, to employ the iterative approach, it is useful to include elitism in other heuristics.

This study demonstrated and proved that the initially intuitive thought that the iterative approach would improve the makespan of the non-makespan machines is incorrect. Because the iterative technique intuitively seemed promising but was not (except for its use in the GA), we felt it was worthwhile to explore why it does not work. We feel both the negative result and the formal mathematical analysis to prove the limitations of the iterative techniques are contributions to the field.

Acknowledgements A preliminary version of portions of this material appeared in the 16th Heterogeneity in Computing Workshop. The authors thank Ye Hong and Vladimir Shestak for their valuable comments.

This research was supported by the NSF under grants CNS-0615170 and CNS-0905399, and by the Colorado State University George T. Abell Endowment.

References

1. Ali S, Braun TD, Siegel HJ, Maciejewski AA, Beck N, Boloni L, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B (2005) Characterizing resource allocation heuristics for heterogeneous computing systems. In: Parallel, distributed, and pervasive computing. Advances in computers, vol 63, pp 91–128
2. Al-Azzoni I, Down DG (2007) Linear programming based affinity scheduling for heterogeneous computing systems. In: International conference on parallel and distributed processing techniques and applications (PDPTA'07), June 2007
3. Barada H, Sait SM, Baig N (2001) Task matching and scheduling in heterogeneous systems using simulated evolution. In: 10th IEEE heterogeneous computing workshop (HCW 2001) in the proceedings of the 15th international parallel and distributed processing symposium (IPDPS 2001), Apr 2001, pp 875–882
4. Barbulescu L, Whitley LD, Howe AE (2004) Leap before you look: an effective strategy in an oversubscribed scheduling problem. In: 19th national conference on artificial intelligence, July 2004, pp 143–148
5. Berman F, Casanova H, Chien A, Cooper K, Dail H, Dasgupta A, Deng W, Dongarra J, Johnsson L, Kennedy K, Koelbel C, Liu B, Liu X, Mandal A, Marin G, Mazina M, Mellor-Crummey J, Mendes C, Olugbile A, Patel M, Reed D, Shi Z, Sievert O, Xia H, YarKhan A (2005) New grid scheduling and rescheduling methods in the GrADS project. Int J Parallel Program 33(2–3):209–229

6. Berman F, Wolski R, Casanova H, Cirne W, Dail H, Faerman M, Figueira S, Hayes J, Obertelli G, Schopf J, Shao G, Smallen S, Spring S, Su A, Zagorodnov D (2003) Adaptive computing on the grid using AppLeS. IEEE Trans. Parallel Distrib. Syst. 14(4):369–382
7. Blythe J, Jain S, Deelman E, Gil Y, Vahi K, Mandal A, Kennedy K (2005) Task scheduling strategies for workflow-based applications in grids. In: IEEE international symposium on cluster computing and the grid (CCGrid 2005), May 2005, vol 2, pp 759–767
8. Briceno LD, Khemka B, Siegel HJ, Maciejewski AA, Groer C, Koenig G, Okonski G, Poole S (2011) Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing system. In: 20th international heterogeneity in computing workshop (HCW'11) in the proceedings of the parallel and distributed processing workshops and Phd forum (IPDPSW 2011), pp 7–19
9. Braun TD, Siegel HJ, Beck N, Boloni L, Freund RF, Hensgen D, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B (2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distrib Comput 61(6):810–837
10. Briceño LD, Siegel HJ, Maciejewski AA, Oltikar M, Brateman J, White J, Martin J, Knapp K (2011) Heuristics for robust resource allocation of satellite weather data processing on a heterogeneous parallel system. IEEE Trans Parallel Distrib Syst 22(11):1780–1787
11. Caron E, Garonne V, Tsaregorodtsev A (2007) Definition, modelling and simulation of a grid computing scheduling system for high throughput computing. Future Gener Comput Syst 23(8):968–976
12. Caniou Y, Jeannot E (2006) Multicriteria scheduling heuristics for Gridrpc systems. Int J High Perform Comput Appl 20(1):443–454
13. Casanova H, Legrand A, Zagorodnov D, Berman F (2000) Heuristics for scheduling parameter sweep applications in grid environments. In: 9th IEEE heterogeneous computing workshop (HCW'2000), Mar. 2000, pp 349–363
14. Curran O, Shearer A (2009) A workflow model for heterogeneous computing environments. Future Gener Comput Syst 25(4):414–425
15. Dhodhi MK, Ahmad I, Yatama A (2002) An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems. J Parallel Distrib Comput 62(9):1338–1361
16. Ding Q, Chen G (2001) A benefit function mapping heuristic for a class of meta-tasks in grid environments. In: 1st international symposium on cluster computing and the grid (CCGRID '01), May 2001
17. Dogan A, Ozguner F (2004) Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems. Clust Comput 7(2):177–190
18. Freund RF, Siegel HJ (1993) Heterogeneous processing. Computer 26(6):13–17
19. Ghanbari S, Meybodi MR (2005) Learning automata based algorithms for mapping of a class of independent tasks over highly heterogeneous grids. In: European grid conference (EGC2005), Feb 2005, pp 681–690
20. Ghanbari S, Meybodi MR (2005) On-line mapping algorithms in highly heterogeneous computational grids: a learning automata approach. In: International conference on information and knowledge technology (IKT2005), May 2005
21. Ghafoor A, Yang J (1993) A distributed heterogeneous supercomputing management system. Computer 26(6):78–86
22. Ibarra OH, Kim CE (1977) Heuristic algorithms for scheduling independent tasks on non-identical processors. J ACM 24(2):280–289
23. Jinquan Z, Lina N, Changjun J (2005) A heuristic scheduling strategy for independent tasks on grid. In: Eighth international conference on high-performance computing in Asia–Pacific region 2005, Nov 2005
24. Kafil M, Ahmad I (1998) Optimal task assignment in heterogeneous distributed computing systems. IEEE Concurr 6(3):42–51
25. Kaya K, Ucar B, Aykanat C (2007) Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories. J Parallel Distrib Comput 67(3):271–285
26. Khokhar A, Prasanna VK, Shaaban ME, Wang C (1993) Heterogeneous computing: challenges and opportunities. Computer 26(6):18–27
27. Kim J-K, Siegel HJ, Maciejewski AA, Eigenmann R (2008) Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling. IEEE Trans Parallel Distrib Syst 19(11):1445–1457 (Special issue on power-aware parallel and distributed systems)
28. Kumar A, Shorey R (1993) Performance analysis and scheduling of stochastic fork-join jobs in a multicomputer system. IEEE Trans Parallel Distrib Syst 4(10):1147–1164

29. Leangsuksun C, Potter J, Scott S (1995) Dynamic task mapping algorithms for a distributed heterogeneous computing environment. In: 4th IEEE heterogeneous computing workshop (HCW'1995), Apr 1995, pp 30–34
30. Lee YC, Zomaya AY, Siegel HJ (2010) Robust task scheduling for volunteer computing systems. J Supercomput 53:163–181
31. Lindberg P, Leingang J, Lysaker D, Khan SU, Li J (2012) Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed systems. J Supercomput 59:323–360
32. Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF (1999) Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J Parallel Distrib Comput 59(2):107–121
33. Maheswaran M, Braun TD, Siegel HJ (1999) Heterogeneous distributed computing. In: Webster JG (ed) Encyclopedia of electrical and electronics engineering, vol 8. Wiley, New York, pp 679–690
34. Magoulès F, Pan J, Tan K-A, Kumar A (2009) Introduction to grid computing. Numerical anal & scient comp series. Chapman & Hall/CRC, London
35. Machtans E, Sato L, Deppman A (2009) Improvement on scheduling dependent tasks for grid applications. In: International conference on computational science and engineering (CSE'09), Aug 2009, vol 1, pp 95–102
36. Mehta AM, Smith J, Siegel HJ, Maciejewski AA, Jayaseelan A, Ye B (2007) Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness. J Supercomput 42(1):33–58 (Special issue on grid technology)
37. Paranhos D, Cirne W, Brasileiro F (2003) Trading cycles for information: using replication to schedule bag-of-tasks applications on computational grids. In: International conference on parallel and distributed computing, Aug. 2003
38. Russell S, Norvig P (2005) Artificial intelligence a modern approach, 2nd edn. Prentice Hall, New York
39. SaiRanga P, Baskiyar S (2005) A low complexity algorithm for dynamic scheduling of independent tasks onto heterogeneous computing systems. In: 43rd annual southeast regional conference, Mar 2005, vol 1, pp 63–68
40. Shestak V, Chong EKP, Siegel HJ, Maciejewski AA, Benmohamed L, Wang I-J, Daley R (2008) A hybrid branch-and-bound and evolutionary approach for allocating strings of applications to heterogeneous distributed computing systems. J Parallel Distrib Comput 68(4):410–426
41. Singh H, Youssef A (1996) Mapping and scheduling heterogeneous task graphs using genetic algorithms. In: 5th IEEE heterogeneous computing workshop (HCW '1996), Apr 1996, pp 86–97
42. Sonmez OO, Gursoy A (2007) A novel economic-based scheduling heuristic for computational grids. Int J High Perform Comput Appl 21(1):21–29
43. Wei B, Fedak G, Cappello F (2005) Scheduling independent tasks sharing large data distributed with bittorrent. In: The 6th IEEE/ACM international workshop on grid computing, Nov 2005
44. Whitley D (1989) The GENITOR algorithm and selective pressure: why rank based allocation of reproductive trials is best. In: 3rd international conference on genetic algorithms, June 1989, pp 116–121
45. Wu M, Shu W (2000) Segmented min-min: a static mapping algorithm for meta-tasks on heterogeneous computing systems. In: 9th IEEE heterogeneous computing workshop (HCW'2000), Mar. 2000, pp 375–385
46. Xu D, Nahrstedt K, Wichadakul D (2001) QoS and contention-aware multi-resource reservation. Clust Comput 4(2):95–107
47. Yang J, Ahmad I, Ghafoor A (1993) Estimation of execution times on heterogeneous supercomputer architectures. In: International conference on parallel processing, Aug 1993, vol I, pp 219–225