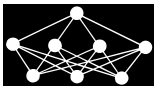

PERFORMANCE COMPARISON OF SIX EFFICIENT PURE HEURISTICS FOR SCHEDULING META-TASKS ON HETEROGENEOUS DISTRIBUTED ENVIRONMENTS

Hesam Izakian[*], *Ajith Abraham*[*], *Václav Snášel*[†]

Abstract: Scheduling is one of the core steps to efficiently exploit the capabilities of heterogeneous distributed computing systems and represents an NP-complete problem. Therefore, using meta-heuristic algorithms is a suitable approach in order to cope with its difficulty. In many meta-heuristic algorithms, generating individuals in the initial step has an important effect on the convergence behavior of the algorithm and final solutions. Using some pure heuristics for generating one or more near-optimal individuals in the initial step can improve the final solutions obtained by meta-heuristic algorithms. Pure heuristics may be used solitary for generating schedules in many real-world situations in which using the meta-heuristic methods are too difficult or inappropriate. Different criteria can be used for evaluating the efficiency of scheduling algorithms, the most important of which are makespan and flowtime. In this paper, we propose an efficient pure heuristic method and then we compare the performance with five popular heuristics for minimizing makespan and flowtime in heterogeneous distributed computing systems. We investigate the effect of these pure heuristics for initializing simulated annealing meta-heuristic approach for scheduling tasks on heterogeneous environments.

Key words: *Heterogeneous distributed environments, scheduling, makespan, flowtime, pure heuristic*

Received: October 12, 2008
Revised and accepted: September 22, 2009

[*]Hesam Izakian, Ajith Abraham – Corresponding Author
Machine Intelligence Research Labs – MIR Labs, `http://www.mirlabs.org`, E-mail: `ajith.abraham@ieee.org, hesam.izakian@gmail.com`
[†]Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Czech Republic, E-mail: `vaclav.snasel@vsb.cz`

1. Introduction

Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different high-performance machines, interconnected with high-speed links, to perform different computationally intensive applications that have diverse computational requirements [1, 2]. To exploit the different capabilities of a suite of heterogeneous resources, typically a resource management system (RMS) allocates the resources to the tasks and the tasks are ordered for execution on the resources. At a time interval in HC environment a number of tasks are received by RMS from different users. Different tasks have different requirements and different resources have different capabilities. Optimally scheduling is mapping a set of tasks to a set of resources to efficiently exploit the capabilities of such systems and is one of the key problems in HC environments. As mentioned in [3, 4], optimally mapping tasks to machines in an HC suite is an NP-complete problem and, therefore, the use of meta-heuristics is one of the suitable approaches. The most popular of meta-heuristic algorithms are genetic algorithm (GA) [14], Tabu search (TS) [15], simulated annealing (SA) [16], ant colony optimization (ACO) [17], and particle swarm optimization (PSO) [18]. Ritchie and Levine [8] used a hybrid ant colony optimization and Yarkhan and Dongarra [9] used simulated annealing approach for task scheduling in HC systems. Genetic algorithm addressed for this problem in several works (Page and Naughton [5], Singh and Youssef [6] and Wang et al. [7]). Also Izakian et al. used PSO for task scheduling in HC systems [19, 20].

In many meta-heuristic algorithms and based on the problem that is to be solved, generating individuals in the initial step has an important effect on the convergence behavior of the algorithm and final solutions. Therefore using some pure heuristics for generating one or more near-optimal individuals in the initial step can improve the final solutions obtained by meta-heuristic algorithms. On the other hand, in some real-world situations the meta-heuristic methods are too difficult or inappropriate, for example in fully-automated systems (where we cannot tune parameters manually) or where the execution time should be very short, or for extremely large problems, etc. Therefore using pure heuristics in such situations is an appropriate solution.

Existing scheduling heuristics can be divided into two classes [12]: on-line mode (immediate mode) and batch-mode heuristics. In the on-line mode, a task is mapped onto a host as soon as it arrives at the scheduler. In the batch mode, tasks are not mapped onto hosts immediately and they are collected into a set of tasks that is examined for mapping at prescheduled times called mapping events. The on-line mode heuristic is suitable for the low arrival rates, while batch-mode heuristics can achieve higher performance when the arrival rate of tasks is high because there will be a sufficient number of tasks to keep hosts busy between the mapping events, and scheduling is according to the resource requirement information of all tasks in the set [12]. In this paper, we considered batch-mode heuristics.

Different criteria can be used for evaluating the efficiency of scheduling algorithms, the most important of which are makespan and flowtime. Makespan is the time when HC system finishes the latest task, and flowtime is the sum of finalization times of all the tasks. An optimal schedule will be the one that minimizes the flowtime and makespan.

In this paper, we propose an efficient heuristic called min-max. Also we investigate the efficacy of min-max and 5 popular pure heuristics for minimizing makespan and flowtime. These heuristics are min-min, max-min, LJFR-SJFR, sufferage, and WorkQueue. These heuristics are popular, effective, and are used in many studies. Also we investigate the effect of these pure heuristics for initializing simulated annealing meta-heuristic approach for task scheduling on heterogeneous environments.

So far, some of works have been done for investigating a number of these heuristics for minimizing makespan, yet no attempts has been made to minimize flowtime or both flowtime and makespan. Also the efficiency of these heuristics is investigated on simple benchmarks, and the various characteristics of machines and tasks in HC environments are not considered. In this paper, we investigate the efficiency of these heuristics on HC environments with various characteristics of both machines and tasks. In this study, we were able to identify which heuristic is suitable for generating near-optimal solutions or to initialize meta-heuristics based on the objective function and the characteristics of HC environments.

The remainder of this paper is organized in the following manner: Section 2 formulates the problem, in Section 3 we provide the definitions of heuristics, and Section 4 reports the experimental results. Finally Section 5 concludes this work.

2. Problem Formulation

An HC environment is composed of computing resources where these resources can be a single PC, a cluster of workstations or a supercomputer. Let $T = \{T_1,\ T_2, \ldots, T_n\}$ denotes the set of tasks that is submitted to RMS in a specific time interval. Assume the tasks are independent of each other (with no inter-task data dependencies) and preemption is not allowed (they cannot change the resource they have been assigned to). Also assume at the time of receiving these tasks by RMS, m machines $M = \{M_1,\ M_2, \ldots, M_m\}$ are within the HC environment. In this paper, scheduling is done at machine level and it is assumed that each machine uses First-Come, First-Served (FCFS) method for performing the received tasks. We assume that each machine in HC environment can estimate how much time is required to perform each task. In [2] Expected Time to Compute (ECT) matrix is used to estimate the required time for executing a task in a machine. An ETC matrix is an $n \times m$ matrix in which n is the number of tasks and m is the number of machines. One row of the ETC matrix contains the estimated execution time for a given task on each machine. Similarly, one column of the ETC matrix consists of the estimated execution time of a given machine for each task. Thus, for an arbitrary task T_j and an arbitrary machine M_i, $ETC(T_j, M_i)$ is the estimated execution time of T_j on M_i. In ETC model, we take the usual assumption that we know the computing capacity of each resource, an estimation or prediction of the computational needs of each task, and the load of prior work of each resource.

Assume that $E_{ij} (i \in \{1, 2, \ldots, m\}, j \in \{1, 2, \ldots, n\})$ is the execution time for performing jth task in ith machine and $W_i (i \in \{1, 2, \ldots, m\})$ is the previous workload of M_i (the time required for performing the tasks given to it in the previous steps), then Eq. (1) shows the time required for M_i to complete the tasks included in it. According to the aforementioned definition, makespan and flowtime can be

estimated using Eq. (2) and Eq. (3) respectively.

$$\sum_{E_{ij}+W_i}, j \in \{1, 2, \ldots, n\} \tag{1}$$

$$makespan = \max \left\{ \sum_{E_{ij}+W_i} \right\}, \ i \in \{1, 2, \ldots, m\}, \ j \in \{1, 2, \ldots, n\} \tag{2}$$

$$flowtime = \sum_{i=1}^{m} E_{ij}, \quad j \in \{1, 2, \ldots, n\} \tag{3}$$

As mentioned in the previous section, the goal of the scheduler in this paper is to minimize makespan and flowtime.

3. Heuristic Descriptions

This section provides the description of 5 popular heuristics for mapping tasks to available machines in HC environments. Then we propose an efficient heuristic called min-max.

3.1 Min-min heuristic

Min-min heuristic uses minimum completion time (MCT) as a metric, means that the task which can be completed earliest is given priority. This heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times, $M = \{\min(completion\_time(T_i, M_j)) \text{ for } (1 \leq i \leq n, 1 \leq j \leq m)\}$, is found. M consists of one entry for each unmapped task. Next, the task with the overall minimum completion time from M is selected and assigned to the corresponding machine. Then the workload of the selected machine will be updated and finally the newly mapped task is removed from U. This process repeats until all tasks are mapped (i.e. U is empty) [2, 10]. Fig. 1 shows the pseudo code of min-min heuristic.

3.2 Max-min heuristic

Max-min heuristic is very similar to min-min and its metric is MCT too. It begins with the set U of all unmapped tasks. Then the set of minimum completion times, $M = \{\min(completion\_time(T_i, M_j)), \text{ for } (1 \leq i \leq n, 1 \leq j \leq m)\}$, is found. Next, the task with the overall maximum completion time from M is selected and assigned to the corresponding machine, and the workload of the selected machine will be updated. Finally the newly mapped task is removed from U and the process repeats until all tasks are mapped [2, 10]. Fig. 2 shows the pseudo code of max-min heuristic.

3.3 LJFR-SJFR heuristic

Longest Job to Fastest Resource – Shortest Job to Fastest Resource (LJFR-SJFR) [11] heuristic begins with the set $U$of all unmapped tasks. Then the set of minimum

```
// min-min heuristic
U = set of unmapped tasks
while U ≠ empty do
        Z = empty;
        for each task T_j ∈ U do
                for each machine M_i, i = 1, 2, ..., m do
                        C_ij = W_i + E_ij;
                end
                C_xj = min(C_ij),  ∀i = 1, 2, ..., m;
                Z = Z ∪ C_xj;
        end
        Select C_qp = min(C_xy),   C_xy ∈ Z;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
end
```

Fig. 1 *The pseudo code of min-min heuristic.*

```
// max-min heuristic
U = set of unmapped tasks
while U ≠ empty do
        Z = empty;
        for each task T_j ∈ U do
                for each machine M_i, i = 1, 2, ..., m do
                        C_ij = W_i + E_ij;
                end
                C_xj = min(C_ij),  ∀i = 1, 2, ..., m;
                Z = Z ∪ C_xj;
        end
        Select C_qp = max(C_xy),   C_xy ∈ Z;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
end
```

Fig. 2 *The pseudo code of max-min heuristic.*

completion times, $M = \{\min(completion\_time(T_i, M_j))$ for $(1 \leq i \leq n, 1 \leq j \leq m)\}$, is found the same as min-min. Next, the task with the overall minimum completion time from M is considered as the shortest job in the fastest resource (SJFR). Also the task with the overall maximum completion time from M is considered as the longest job in the fastest resource (LJFR). At the beginning, this method assigns the m longest tasks to the m available fastest resources (LJFR). Then this method assigns the shortest task to the fastest resource, and the longest task to the

fastest resource alternatively. After each allocation, the workload of each machine will be updated. Fig. 3 shows the pseudo code of LJFR-SJFR heuristic.

```
// LJFR-SJFR heuristic
U = set of unmapped tasks
for m times do
        Select task T_p and machine M_q the same as max-min heuristic;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
end
while U ≠ empty do
        Select task T_p and machine M_q the same as min-min heuristic;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
        if U = empty then
                Terminate;
        end
        Select task T_p and machine M_q the same as max-min heuristic;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
end
```

Fig. 3 *The pseudo code of LJFR-SJFR heuristic.*

3.4 Sufferage heuristic

In this heuristic for each task, the minimum and second minimum completion time are found in the first step. The difference between these two values is defined as the sufferage value. In the second step, the task with the maximum sufferage value is assigned to the corresponding machine with minimum completion time. Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would "suffer" most in terms of expected completion time if that particular machine is not assigned to it [12]. Fig. 4 shows the pseudo code of Sufferage heuristic.

3.5 WorkQueue heuristic

This heuristic is a straightforward and adaptive scheduling algorithm for scheduling sets of independent tasks [13]. In this method, the heuristic selects a task randomly and assigns it to the machine as soon as it becomes available (in other word, machine with minimum workload). Fig. 5 shows the pseudo code of WorkQueue heuristic.

```
// Sufferage heuristic
U = set of unmapped tasks
while U ≠empty do
        Z = empty;
        for each task T_j ∈ U do
                for each machine M_i, i = 1, 2, ..., m do
                        C_ij = W_i + E_ij;
                end
                C_xj = min(C_ij),  ∀i = 1, 2, ..., m;
                C_kj = min(C_ij),  ∀i = 1, 2, ..., m, i ≠ x
                Suffer_xj = C_kj − C_xj;
                Z = Z ∪ Suffer_xj;
        end
        Select Suffer_qp = max(Suffer_xy),   Suffer_xy ∈ Z;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
end
```

Fig. 4 *The pseudo code of Sufferage heuristic.*

```
// WorkQueue heuristic
U = set of unmapped tasks
while U ≠ empty do
        Select a task T_p ∈ U randomly
        W_q = min(W_i),  ∀i = 1, 2, ..., m;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
end
```

Fig. 5 *The pseudo code of WorkQueue heuristic.*

3.6 Proposed heuristic

This heuristic (called min-max) is composed of two steps for mapping each task and uses the minimum completion time in the first step, and the minimum execution time in the second as metric. In the first step, this heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times, $M = \{\min(completion\_time(T_i, M_j)) \text{ for } (1 \leq i \leq n, 1 \leq j \leq m)\}$, is found the same as min-min heuristic. In the second step, the task whose minimum execution time (time for executing task on the fastest machine) divided by its execution time on the selected machine (in the first step) has the maximum value, will be selected for mapping. The intuition behind this heuristic is that we select pair machines and tasks from the first step that the machine can executes its corresponding task

effectively with a lower execution time in comparison with other machines. Fig. 6 shows the pseudo code of proposed min-max heuristic.

```
// proposed min-max heuristic
U = set of unmapped tasks
while U ≠ empty do
        Z = empty;
        for each task T_j ∈ U do
                for each machine M_i, i=1,2,..., m do
                        C_ij = W_i + E_ij;
                end
                C_xj = min(C_ij),  ∀i = 1, 2, ..., m;
                E_hj = min(E_ij),  ∀i = 1, 2, ..., m;
                K_xj = E_xj / E_hj;
                Z = Z ∪ K_xj;
        end
        Select K_qp = max(K_xy),   K_xy ∈ Z;
        Allocate task T_p to machine M_q;
        W_q = W_q + E_qp;
        U = U − T_p;
end
```

Fig. 6 *The pseudo code of proposed min-max heuristic.*

4. Experimental Results

In this section, after describing the benchmark problems, we compared the performance of 6 pure heuristics for minimizing makespan and flowtime. Also we investigate the effect of this pure heuristics for initializing the simulated annealing algorithm (which is a popular meta-heuristic algorithm) for scheduling independent tasks on HC environments.

4.1 Benchmark problems

In this paper, we used the benchmark proposed in [2]. The simulation model in [2] is based on expected time to compute (ETC) matrix for 512 tasks and 16 machines. The instances of the benchmark are classified into 12 different types of ETC matrices according to the three following metrics: task heterogeneity, machine heterogeneity, and consistency. In ETC matrix, the amount of variance among the execution times of tasks for a given machine is defined as task heterogeneity. Machine heterogeneity represents the variation that is possible among the execution times for a given task across all the machines. Also an ETC matrix is said to be consistent whenever a machine M_j executes any task T_i faster than machine M_k; in this case, machine M_j executes all tasks faster than machineM_k. In contrast, inconsistent matrices characterize the situation where machine M_j may be faster

than machine M_k for some tasks and slower for others. Partially-consistent matrices are inconsistent matrices that include a consistent sub-matrix of a predefined size [2]. Instances consist of 512 tasks and 16 machines and are labeled as u-yy-zz-x as follows:

- u means uniform distribution used in generating the matrices
- yy indicates the heterogeneity of the tasks; hi means high and lo means low
- zz represents the heterogeneity of the machines; hi means high and lo means low
- x shows the type of inconsistency; c means consistent, i means inconsistent, and p means partially-consistent.

For example, u-lo-hi-c means low heterogeneity in tasks, high heterogeneity in machines, and consistent environment.

4.2 Comparison of mentioned heuristics

These pure heuristics are implemented using C++ programming language and are run on 12 different types of ETC matrices. The obtained makespan and flowtime using mentioned heuristics are compared in Tabs. I and II respectively. The results are obtained as an average of five simulations. In these tables, the first column indicates the instance name, and the second, third, fourth, fifth and sixth columns indicate the makespan and flowtime of WorkQueue, max-min, LJFR-SJFR, Sufferage, min-min and min-max heuristics. Fig. 7 and Fig. 8 show the geometric mean of makespan and flowtime for the 12 considered cases. As is evident from the figures, min-max, the proposed heuristic, can minimize the makespan better than others in most cases. Also min-min heuristic can minimize flowtime better than others.

Instance	WorkQueue	Max-Min	LJFR-SJFR	Sufferage	Min-Min	Min-Max
u-lo-lo-c	7332	6753	6563	5461	5468	**5310**
u-lo-lo-p	8258	5947	5179	3433	3599	**3327**
u-lo-lo-i	9099	4998	4251	2577	2734	**2523**
u-lo-hi-c	473353	400222	391715	333413	279651	**273467**
u-lo-hi-p	647404	314048	279713	163846	157307	**146953**
u-lo-hi-i	836701	232419	209076	121738	113944	**102543**
u-hi-lo-c	203180	203684	202010	170663	164490	**164134**
u-hi-lo-p	251980	169782	155969	105661	106322	**103321**
u-hi-lo-i	283553	153992	138256	**77753**	82936	77873
u-hi-hi-c	13717654	11637786	11305465	9228550	8145395	**7878374**
u-hi-hi-p	18977807	9097358	8027802	4922677	4701249	**4368071**
u-hi-hi-i	23286178	7016532	6623221	3366693	3573987	**2989993**

Tab. I *Comparison of results on makespan (Seconds).*

Instance	WorkQueue	Max-Min	LJFR-SJFR	Sufferage	Min-Min	Min-Max
u-lo-lo-c	108843	108014	102810	86643	**80354**	84717
u-lo-lo-p	127639	95091	81861	54075	**51399**	52935
u-lo-lo-i	140764	79882	66812	40235	**39605**	39679
u-lo-hi-c	7235486	6400684	6078313	5271246	**3918515**	4357089
u-lo-hi-p	10028494	5017831	4383010	2568300	**2118116**	2323396
u-lo-hi-i	12422991	3710963	3303836	1641220	**1577886**	1589574
u-hi-lo-c	3043653	3257403	3153607	2693264	**2480404**	2613333
u-hi-lo-p	3776731	2714227	2461337	1657537	**1565877**	1640408
u-hi-lo-i	4382650	2462485	2181042	1230495	1214038	**1205625**
u-hi-hi-c	203118678	185988129	173379857	145482572	**115162284**	125659590
u-hi-hi-p	282014637	145337260	126917002	76238739	**63516912**	69472441
u-hi-hi-i	352446704	112145666	104660439	47237165	**45696141**	46118709

Tab. II *Comparison of results on flowtime (Seconds).*

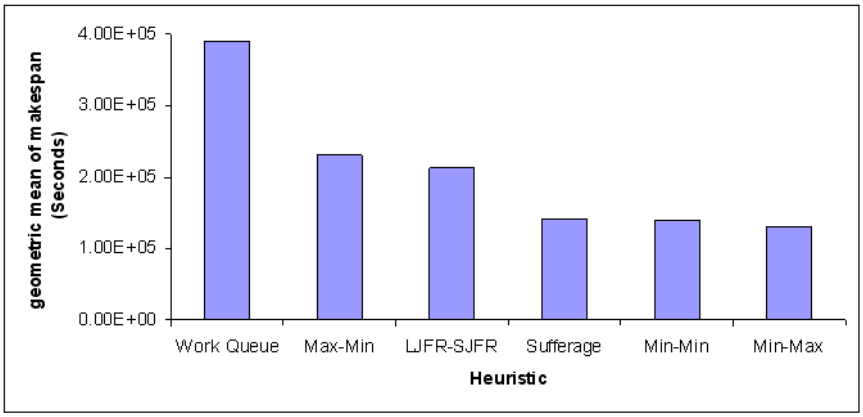

Fig. 7 *Comparison results between heuristics on makespan.*

4.3 Using the mentioned heuristics for initializing simulated annealing

Simulated annealing (SA) is an optimization technique inspired from Monte Carlo methods in statistical mechanics. In this sub-section, we investigate the effect of the mentioned pure heuristics to initialize the simulated annealing meta-heuristic for scheduling tasks on heterogeneous environments. In this method, an initial solution is generated randomly or using a pure heuristic and the system temperature is set to a high value. In each step of the algorithm, the system temperature is decreased based on a predefined cooling factor and in the last step finalizes to the freezing temperature. Each step of the algorithm can include one or more iterations. In each iteration, a new solution is generated by adding small changes to current solution (e.g. using mutation operator). In this paper, this operator selects a task randomly and moves it from its resource to another one, so that the new machine is assigned to be different. If the new solution improves the objective function (in

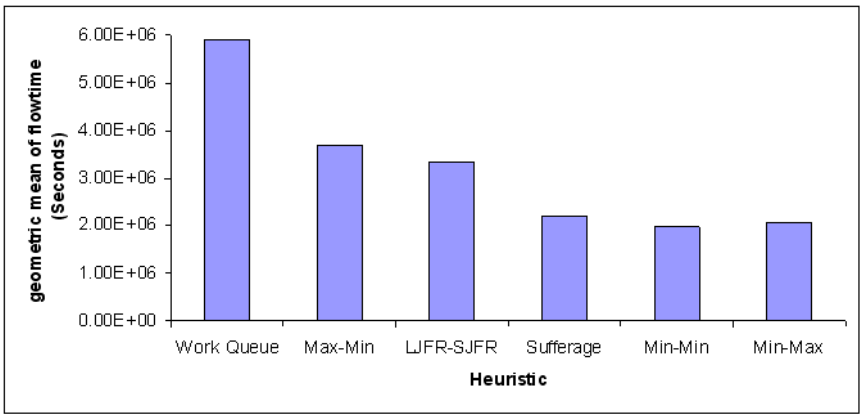

Fig. 8 *Comparison results between heuristics on flowtime.*

this paper decreases the objective function), then it will be accepted and preserved as current solution. On the other hand, if the objective function does not improve, a random number $z \in [0,\ 1)$ is selected. Then z is compared with y, where

$$y = \frac{1}{1 + e^{\frac{\text{(old objective function - new objective function)}}{\text{temperature}}}} \tag{4}$$

If $z > y$ the new solution is accepted; otherwise it is rejected, and the old solution is kept. When the temperature is very large the poorer solutions can be accepted by a high probability. In contrast, if the system temperature is very small, the poorer solutions will usually be rejected. By this procedure the SA can explore the problem search space effectively and escape from local optima. In this paper, makespan and flowtime are used to evaluate the performance of scheduler simultaneously. Since makespan and flowtime values are in incomparable ranges and the flowtime has a higher magnitude order over the makespan (as can be seen in Tab. I and Tab. II), the value of mean flowtime, *flowtime / m*, is used to evaluate flowtime where m is the number of machines. The objective function of each solution can be estimated using Eq. (5).

$$f(x) = (\lambda\,.\,makespan\ +\ (1-\lambda).\,mean\_flowtime), \tag{5}$$

where λ controls the effectiveness of parameters used in this equation. The greater is λ, more attention is paid by the scheduler in minimizing makespan and vise versa. The smaller makespan and flowtime in Eq. (5) leads to a smaller objective function value, and hence a better solution is regarded. In order to optimize the performance of the SA, fine tuning has been performed and best values for its parameters are selected. Tab. III shows the selected parameters for SA. Also Fig. 9 shows the pseudo code of proposed SA.

The obtained makespan and flowtime using SA are compared in Tabs. IV and V respectively. In these tables the effect of initialization using 6 mentioned heuristics and random method is compared. The results are obtained as an average of ten independent runs for five simulations.

Parameter	Description	Value
λ	In Eq. (5)	0.7
$T(0)$	Initial temperature	Objective function value obtained by initial solution
T(freeze)	Final temperature	10^{-10}
Itr	Number of iterations in each temperature	100
C	Cooling rate	0.9

Tab. III *Parameter tuning for proposed SA.*

```
Generate initial solution, x(0) randomly or using one of pure heuristics;
Set initial temperature = f(x(0));
while temperature > T(freeze) do
        repeat Itr times
                Generate a new solution, x(1) using mutation operator;
                Determine quality f(x(1));
                if f(x(1)) < f(x(0)) then
                        x(0) ← x(1);
                else
                        z = rand [0, 1);
                        Calculate acceptances probability, y using Eq (4);
                        if z > y then
                                x(0) ← x(1);
                        else
                                preserve x(0) and reject x(1);
                        end
                end
        end
        temperature = C× temperature;
end
```

Fig. 9 *The pseudo code of the proposed SA.*

As can be seen in Tab. IV and Tab. V:

- Using pure heuristics (instead of random method) for generating initial solutions has an important effect on obtained final solutions and more effective solutions can be generated.
- In some instances, SA scheduler can't improve the solutions which generated using pure heuristics.

– Minimizing makespan can lead to increasing in flowtime amount and vise versa.

– For minimizing makespan the proposed min-max method performs better than others in partially-consistent and inconsistent environments, while min-min heuristic performs better than others in consistent environments.

– For minimizing flowtime the min-min method performs better than others in consistent and partially-consistent environments, while the proposed min-max heuristic performs better than others in inconsistent environments.

Instance	Random	Work-Queue	Max-Min	LJFR-SJFR	Sufferage	Min-Min	Min-Max
u-lo-lo-c	6267	5300	5301	5353	5303	**5298**	5312
u-lo-lo-p	3604	3555	3520	3458	3423	3542	**3327**
u-lo-lo-i	2590	2581	2565	2599	2563	2560	**2523**
u-lo-hi-c	545098	270529	267760	299709	311897	**264281**	269135
u-lo-hi-p	273568	180663	154830	155222	158623	155861	**148949**
u-lo-hi-i	108364	105978	105343	107426	105649	104022	**102543**
u-hi-lo-c	198499	161805	161621	164656	163298	**161406**	163125
u-hi-lo-p	108141	110412	107305	105578	104629	107979	**103580**
u-hi-lo-i	78733	78450	77968	77902	**77625**	77814	77809
u-hi-hi-c	15439648	7935460	7820153	8603537	8808243	**7721432**	7853547
u-hi-hi-p	6816472	5052386	4561310	4595143	4814050	4605232	**4437986**
u-hi-hi-i	3079057	3032995	3025251	3103927	3026366	3019183	**2989993**

Tab. IV *Comparison of results on makespan obtained by SA and one of initializing methods (Seconds).*

Instance	Random	Work-Queue	Max-Min	LJFR-SJFR	Sufferage	Min-Min	Min-Max
u-lo-lo-c	94560	79024	78816	84352	83856	**78656**	80560
u-lo-lo-p	55088	53280	50352	50912	52480	**48112**	52928
u-lo-lo-i	40848	40784	40672	40976	40368	40544	**39664**
u-lo-hi-c	7482864	3798864	3765920	4727440	4933840	**3644272**	3886560
u-lo-hi-p	3130864	2632784	2081440	2304624	2416320	**1958848**	2225536
u-lo-hi-i	1661728	1638240	1632048	1660192	1633952	1622672	**1589568**
u-hi-lo-c	2966336	2425504	2425776	2606080	2571872	**2417152**	2480688
u-hi-lo-p	1683840	1642560	1529360	1585472	1617136	**1477856**	1630672
u-hi-lo-i	1244576	1242576	1234864	1234816	1229392	1232448	**1207520**
u-hi-hi-c	214952928	110491936	108906160	136272016	138255728	**104842480**	112613840
u-hi-hi-p	87220128	71689360	61012464	68732480	72772688	**57206896**	65564240
u-hi-hi-i	47367664	46694112	46681648	48080256	46840608	46708112	**46118704**

Tab. V *Comparison of results on flowtime obtained by SA and one of initializing methods (Seconds).*

Fig. 10 and Fig. 11 show the comparison of obtained results using different heuristics to generate initial solutions in SA for geometric mean of makespan and flowtime respectively. As is evident in these figures, the SA can improve the initial solutions which generated using some pure heuristics more in comparison with other pure heuristics. For example, SA can improve the generated initial solutions using WorkQueue and max-min more than others, while it can't improve the generated initial solutions using Sufferage effectively.

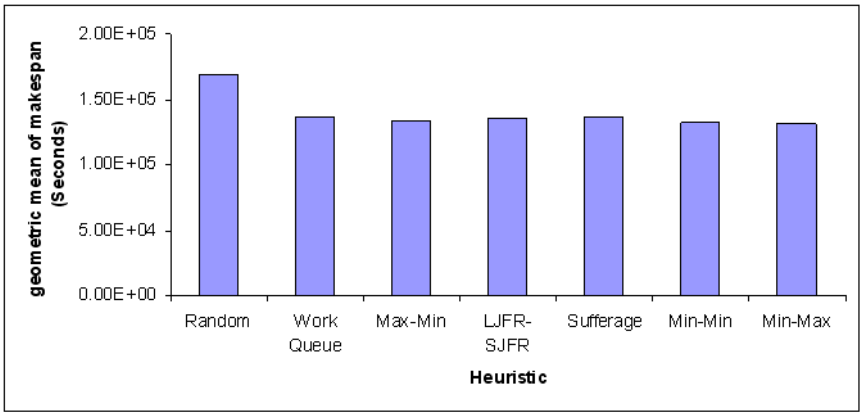

Fig. 10 *Comparison results on makespan obtained by SA and one of initializing methods.*

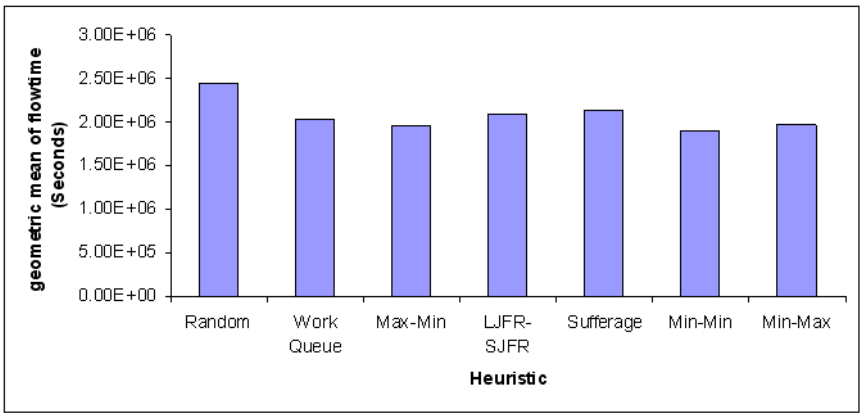

Fig. 11 *Comparison results on flowtime obtained by SA and one of initializing methods.*

5. Conclusions

Scheduling is one of the core steps to efficiently exploit the capabilities of heterogeneous distributed computing systems and represents an NP-complete problem. Using heuristics for scheduling in HC environments is an appropriate solution. In this paper, we compared six heuristics for scheduling in HC environments. The goal of the scheduler in this paper is minimizing makespan and flowtime. The experimental results show that the min-min heuristic can obtain the best results for minimizing flowtime and the proposed heuristic (min-max) can obtain the best results for minimizing makespan too. Also we investigate the effect of the mentioned pure heuristics for initializing simulated annealing meta-heuristic for scheduling tasks on heterogeneous environments. The experimental results show that the min-min and min-max heuristics are more effective than others for generating initial solutions in SA. Also we can find that min-min heuristic is more suitable for consistent environments, while min-max heuristic is more suitable for inconsistent environments for initializing meta-heuristic schedulers such as SA.

References

[1] Ali S., Braun T. D., Siegel H. J., Maciejewski A. A.: Heterogeneous computing. Encyclopedia of Distributed Computing, Kluwer Academic, 2001.

[2] Braun T. D., Siegel H. J., Beck N., Boloni L. L., Maheswaran M., Reuther A. I., Robertson J. P., Theys M. D., Yao B.: A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. Journal of Parallel and Distributed Computing, **61**, 2001, pp. 810-837.

[3] Fernandez-Baca D.: Allocating modules to processors in a distributed system. IEEE Transactions on Software Engineering, **15**, 1989, pp. 1427-1436.

[4] Ibarra O. H., Kim C. E.: Heuristic algorithms for scheduling independent tasks on nonidentical processors. J. Assoc. Comput. Mach, 1977, pp. 280-289.

[5] Page J., Naughton J.: Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. Artificial Intelligence Review, 2005, pp. 415-429.

[6] Singh H., Youssef A.: Mapping and scheduling heterogeneous task graphs using genetic algorithms. 5th IEEE Heterogeneous Computing Workshop, 1996, pp. 86-97.

[7] Wang L., Siegel H. J., Roychowdhury V. P., Maciejewski A. A.: Task matching and scheduling in heterogeneous computing environments using a genetic algorithm based approach. J. Parallel Distrib. Comput., **47**, 1997, pp. 1-15.

[8] Ritchie G., Levine J.: A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. 23rd Workshop of the UK Planning and Scheduling Special Interest Group, 2004.

[9] Yarkhan A., Dongarra J.: Experiments with scheduling using simulated annealing in a grid environment. 3rd International Workshop on Grid Computing, 2002, pp. 232–242.

[10] Freund R. F., Gherrity M., Ambrosius S., Campbell M., Halderman M., Hensgen D., Keith E., Kidd T., Kussow M., Lima J. D., Mirabile F., Moore L., Rust B., and Siegel H. J.: Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. 7th IEEE Heterogeneous Computing Workshop, 1998, pp. 184-199.

[11] Abraham A., Buyya R., Nath B.: Nature's heuristics for scheduling jobs on computational grids. The 8th IEEE International Conference on Advanced Computing and Communications, 2000.

[12] Macheswaran M., Ali S., Siegel H. J., Hensgen D., Freund R. F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J. Parallel Distribut. Comput., **59**, 1999, pp. 107-131.

[13] Hagerup T.: Allocating Independent Tasks to Parallel Processors: An Experimental Study. Journal of Parallel and Distributed Computing, **47**, 1997, pp. 185-197.

[14] Goldberg D. E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley. Reading. MA, 1997.

[15] Glover F.: Tabu search. ORSA Journal of Computing, **1**, 1989, pp. 190-206.

[16] Kirkpatrick S., Gelatt Jr. C., Vecchi M.: Optimization by simulated annealing. Science, **220**, 1983, pp. 671-680.

[17] Dorigo M.: Optimization, learning, and natural algorithms. Ph.D. Thesis, Dip. Elettronica e Informazione, Politecnico di Milano, Italy, 1992.

[18] Kennedy J., Eberhart R.C.: Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks, 1995, pp. 1942-1948.

[19] Izakian H., Ladani B. T., Zamanifar K., and Abraham A.: A Novel Particle Swarm Optimization Approach for Grid Job Scheduling. Springer-Verlag Berlin Heidelberg, 2009, pp. 100-109.

[20] Izakian H., Abraham A., Snasel V.: Metaheuristic Based Scheduling Meta-Tasks in Distributed Heterogeneous Computing Systems. Sensors, **9**, 2009, pp. 5339-5350.