Evolving Schedules of Independent Tasks by Differential Evolution

Pavel Krömer, Václav Snášel, Jan Platoš, Ajith Abraham, and Hesam Ezakian

Abstract. Scheduling is one of the core steps to efficiently exploit the capabilities of heterogeneous distributed computing systems and it is also an appealing NP-complete problem. There is a number of heuristic and meta-heuristic algorithms that were tailored to deal with scheduling of independent jobs. In this paper we investigate the efficiency of differential evolution for the scheduling problem and compare it with existing approaches. The analysis shows that the differential evolution is a promising method that can compete with well-established scheduling algorithms.

1 Introduction

Grid computing and distributed computing, dealing with large scale and complex computing problems, is a hot topic in the computer science and research. Mixed-machine heterogeneous computing (HC) environments utilize a distributed suite of different machines connected via the computer network to perform different computationally intensive applications that have diverse requirements [1, 2]. Miscellaneous resources ought to be orchestrated to perform a number of tasks in parallel, or to solve complex tasks atomized to a variety of independent subtasks [3]. Proper

Pavel Krömer · Václav Snášel · Jan Platoš
Department of Computer Science, VŠB - Technical University of Ostrava,
17. listopadu 15, 708 33, Ostrava-Poruba, Czech Republic
e-mail: {pavel.kromer,vaclav.snasel,jan.platos}@vsb.cz

Ajith Abraham
Machine Intelligence Research Labs (MIR Labs), Washington 98071, USA
e-mail: ajith.abraham@ieee.org

Hesam Izakian
Department of Computer Engineering, University of Isfahan,
Hezar Jerib Avenue, Isfahan, Iran
e-mail: hesam.izakian@gmail.com

S. Caballé, F. Xhafa & A. Abraham (Eds.): INCoS 2009, SCI 329, pp. 79–94.
springerlink.com

scheduling of the tasks on available resources is one of the main challenges of a mixed-machine HC environment.

To exploit the different capabilities of a suite of heterogeneous resources, 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, number of tasks are received by the RMS. Task scheduling is mapping of a set of tasks to a set of resources to efficiently exploit the capabilities of such.

It is known that an optimal mapping of computational tasks to available machines in a HC suite is a NP-complete problem [4] and as such, it is a subject to various heuristic and meta-heuristic algorithms. The heuristics applied to the task scheduling problem include min-min heuristic, max-min heuristic, longest job to fastest resource- shortest job to fastest resource heuristic, sufferage heuristic, work queue heuristic and others [2, 5, 6]. The meta-heuristics applied to the task scheduling problem include hybrid ant colony optimization [7], simulated annealing [8] and genetic algorithms [9, 10, 11]. The meta-heuristic algorithms usually operate with a population of prospective problem solutions - task schedules - that are evolved (optimized) in order to obtain an improved schedule which is optimized according to some criteria.

In this paper, we apply a powerful populational meta-heuristic algorithm - the differential evolution - to the task scheduling problem. Moreover, to improve the efficiency of the general meta-heuristic solver, several widely used heuristic algorithms for scheduling in the HC environments were used to improve the initial population for differential evolution.

2 Heuristic Algorithms for Mapping Tasks in HC Environment

There is a number of heuristic algorithms designed to schedule independent tasks in heterogeneous computing environments. Each algorithm exploits a heuristic based on certain intuition that helps mapping tasks to machines so that selected objective is optimized. Unfortunately, different heuristics perform under various circumstances differently [12, 6].

In [12] was presented an analysis of 16 heuristic algorithms for the task scheduling problem. The authors have shown that different scheduling heuristics perform differently for different expected time to compute (ETC) matrices. The ETC matrices were generated with various values of task heterogeneity V_{task} and machine heterogeneity $V_{machine}$. A series of computational experiments has shown that the result of different heuristic algorithms depends on the values of V_{task} and $V_{machine}$. For the ETC matrices with both, high V_{task} and high $V_{machine}$, the Heaviest Task First (HTF) [12] heuristic performed better than other heuristic algorithms. For the ETC matrices with either medium V_{task} or medium $V_{machine}$ obtained best results heuristic algorithm called Task Partitioning Min (TPMin) introduced in the same study. The ETC matrices with low V_{task} or low $V_{machine}$ were solved by the heuristic algorithm Task Partitioning Std (TPStd) with greatest success. The details on TPMin, TPStd and the description of the experiments can be found in [12]. In short, to obtain best

schedules, the heuristic algorithm had to be chosen according to the ETC matrix properties V_{task} and $V_{machine}$. Such an observation encourages the investigation of adaptive, meta-heuristic, approaches to the scheduling problem that are supposed to yield good performance for ETC matrices with various values of V_{task} and $V_{machine}$.

Efficient heuristic algorithms for scheduling in HC environments include [5, 6, 2]:

- Min-min heuristic that prioritizes tasks that can be completed earliest.
- Max-min heuristic that prioritizes tasks with the maximum earliest completion time. It aims to overlap long-running tasks with short-running ones.
- Sufferage heuristic that 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.
- Min-max heuristic that combines two metrics, the minimum execution time and the minimum completion time. It aims to assign the task to a machine that can handle it with lower execution time in comparison with other machines.

We provide brief description of discussed scheduling heuristics.

2.1 Min-min Heuristics

Min-min heuristic uses minimum completion time (MCT) as a metric, so that the task which can be completed the earliest is given priority. This heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times $M = \{min(completiontime(T_i, M_j)),\ i \in [1,n], j \in [1,m]\}$ 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 and the workload of the selected machine will be updated. And finally the newly mapped task is removed from U and the process repeats until all tasks are mapped (i.e. U is empty)[2, 13].

2.2 Max-min Heuristic

The Max-min heuristic is very similar to min-min and its metric is MCT as well. It starts with the set of all unmapped tasks U . Then, the set of minimum completion times $M = \{min(completiontime(T_i, M_j)), i \in [1,n], j \in [1,m]\}$ 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 is be updated. Finally, the newly mapped task is removed from U and the process repeats until all tasks are mapped [2, 13].

2.3 Sufferage Heuristic

In this heuristic, the fastest and second fastest completion times are found for each task in the first step. The difference between these two values is called the sufferage

value. In the second step, the task with the maximum sufferage value is assigned to the corresponding machine with minimum completion time. The 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 [14].

2.4 Min-max Heuristic

This heuristic is composed of two steps for mapping each task [6]. It uses the minimum completion time in the first step and the minimum execution time in the second step as a metric. In the first step, this heuristic begins with the set U of all unmapped tasks. Then the set of minimum completion times, $M = \{min(completiontime(T_i, M_j)), i \in [1,n], j \in [1,m]\}$, is found in the same manner as in 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 (from first step) has the maximum value, is selected for mapping. The intuition behind this heuristic is that it selects pairs of machines and tasks so the machine can execute assigned task effectively with a lower execution time in comparison to other machines.

2.5 Scheduling Heuristics – a Summary

From the optimization point of view, each heuristic algorithm represents a strategy that finds a local optimum among all possible schedules. Despite the fact that the heuristic methods obtain quickly good suboptimal results, its inconsistent behavior (i.e. different efficiency of the algorithms for ETC matrices with different values of $V_{machine}$ and V_{task}) encourages the research of global optimization methods for scheduling in HC environments. In this paper, we investigate scheduling of independent tasks as a combinatorial optimization problem and we use differential evolution to find good schedules.

3 Differential Evolution

Differential evolution (DE) is a reliable, versatile and easy to use stochastic evolutionary optimization algorithm [15]. DE is a population-based optimizer that evolves real encoded vectors representing the solutions to given problem. The real-valued nature of population vectors differentiates the DE notably from GAs that were designed to evolve solution encoded into binary or finite alphabets.

The DE starts with an initial population of N real-valued vectors. The vectors are initialized with real values either randomly or so, that they are evenly spread over the problem domain. The latter initialization usually leads to better results of the optimization process [15].

During the optimization, DE generates new vectors that are perturbations of existing population vectors. The algorithm perturbs vectors with the scaled difference of two randomly selected population vectors and adds the scaled random vector difference to a third randomly selected population vector to produce so called trial vector. The trial vector competes with a member of the current population with the same index. If the trial vector represents a better solution than the population vector, it takes its place in the population [15].

Differential evolution is parameterized by two parameters [15]. Scale factor $F \in (0, 1+)$ controls the rate at which the population evolves and the crossover probability $C \in [0,1]$ determines the ratio of bits that are transferred to the trial vector from its opponent. The number of vectors in the population is also an important parameter of the population. The outline of DE is shown in Figure 1.

```
1  Initialize the population P consisting of M vectors
2  Evaluate an objective function ranking the vectors in the population
3  while Termination criteria not satisfied do
4      for i ∈ {1,...,M} do
5          Create trial vector v_t^i = v_r^1 + F(v_r^2 − v_r^3), where F ∈ [0,1] is a parameter
           and v_r^1, v_r^2 and v_r^3 are three random vectors from the population P. This
           step is in DE called mutation.
6          Validate the range of coordinates of v_t^i. Optionally adjust coordinates of v_t^i
           so, that v_t^i is valid solution to given problem.
7          Perform uniform crossover. Select randomly one point (coordinate) l in v_t^i.
           With probability 1 − C let v_t^i[m] = v^i[m] for each m ∈ {1,...,N} such that
           m ≠ l
8          Evaluate the trial vector. If the trial vector v_t^i represent a better solution
           than population vector v^i, replace v^i in P by v_t^i
9      end
10 end
```

Fig. 1 A summary of Differential Evolution

There are more variants of differential evolution. They differ mostly in the way new vectors are generated.

Differential evolution represents an alternative to the more traditional concept of genetic algorithms. As well as genetic algorithms, it represents a highly parallel population based stochastic search metaheuristic. In contrast to GA, differential evolution uses real encoding of chromosomes and different operations to maintain and evolve the population. It results in different search strategy and different directions found by DE when crawling the fitness landscape of a particular problem domain.

4 Differential Evolution for Scheduling Optimization

In general, a combinatorial optimization problem is a problem whose solution is discrete or can be reduced to a discrete one. More formally, a combinatorial optimization (CO) problem $\mathrm{P} = \{I, \{sol(i)\}_{i \in I}, m\}$ can be defined as a minimization or maximization problem that consists of a set of problem instances I, a set of feasible solutions $sol(i)$ for every instance $i \in I$ and a function $m : \{(i,q) | i \in I, q \in sol(i)\} \rightarrow \mathrm{Q}_+$, where Q_+ is the set of positive rational numbers and $m(i,q)$ is the value of solution q for the problem instance i [16]. An optimal solution to an instance of a combinatorial optimization problem is such solution that has maximum (or minimum) value among all other solutions. Famous combinatorial optimization problems include among others the traveling salesman problem, the knapsack problem, and the linear ordering problem [16]. For the job scheduling problem, the set of feasible solutions corresponds to the set of all possible schedules and an optimal schedule is the schedule with lowest value of selected fitness function.

A HC environment is composed of computing resources. These resources can be a single PC, a cluster of workstations or a supercomputer. Let $T = \{T_1, T_2, \ldots, T_n\}$ denote a set of tasks that is at a specific time interval submitted to the RMS. Assume that the tasks are independent of each other with no inter-task data dependencies and preemption is not allowed, i.e. the tasks 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. For our purpose, scheduling is done on 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 the HC environment can estimate how much time is required to perform each task. In [2], the expected time to compute (ETC) matrix is used to estimate the required time for executing a task in a machine. The ETC matrix is a $n \times m$ 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 , $[ETC]_{j,i}$ is the estimated execution time of T_j on M_i. In the 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 job, and the load of prior work of each resource.

The objectives to optimize during the task mapping are makespan and flowtime. Optimum makespan (metatask execution time) and flowtime of a set of jobs can be defined as:

$$makespan = \min_{S \in Sched} \{ \max_{j \in Jobs} F_j \} \tag{1}$$

$$flowtime = \min_{S \in Sched} \{ \sum_{j \in Jobs} F_j \} \tag{2}$$

where *Sched* is the set of all possible schedules, *Jobs* stands for the set of all jobs and F_j represents the time in which job j finalizes. Assume that $[C]_{j,i}$ $(j = 1, 2, \ldots, n,$

$i = 1,2,\ldots,m$) is the completion time for performing j-th task in i-th machine and W_i ($i = 1,2,\ldots,m$) is the previous workload of M_i, then $\sum(C_i + W_i)$ is the time required for M_i to complete the tasks included in it. According to the aforementioned definition, makespan and flowtime can be evaluated using Eq. (3) and Eq. (4) respectively.

$$makespan = \min_{i\in\{1,\ldots,m\}} \{\sum C_i + W_i\} \tag{3}$$

$$flowtime = \sum_{i=1}^{m} C_i \tag{4}$$

Minimizing makespan aims to execute the whole metatask as fast as possible while minimizing flowtime aims to utilize the computing environment efficiently.

4.1 Schedule Encoding

A schedule of n independent tasks executed on m machines can be naturally expressed as a string of n integers $S = (s_1, s_2, \ldots, s_n)$ that are subject to $s_i \in 1,\ldots,m$. The value at i-the position in S represents the machine on which is the i-the job scheduled in schedule S. This encoding is in the literature known as direct encoding [10].

Because the differential evolution encodes candidate solutions as real vectors, real coordinates have to be used instead of discrete machine numbers. The real-encoded DE vector is translated to schedule representation by truncation of its elements.

4.2 Schedule Evaluation

Assume a schedule S from the set of all possible schedules $Sched$. For the purpose of the differential evolution, we define a fitness function $fit(S) : Sched \rightarrow \mathbb{R}$ that evaluates each schedule:

$$fit(S) = \lambda \cdot makespan(S) + (1-\lambda) \cdot \frac{flowtime(S)}{m} \tag{5}$$

The function $fit(S)$ is a sum of two objectives, the makespan of the schedule S and the flowtime of the schedule S divided by the number of machines m to keep both objectives in approximately the same magnitude. The influence of makespan and flowtime in $fit(S)$ is parameterized by the variable weight λ. The same schedule evaluation was used also in [10].

Flowtime and makespan are computed using a binary schedule matrix $B(S) : Sched \rightarrow \{0,1\}^2$ which is constructed as follows: for a $n \times m$ ETC matrix that describes estimated execution times of n jobs on m machines, the $m \times n$ schedule matrix $B(S)$ has in i-th row and j-th column 1 iff the task j is scheduled for execution

on machine i. Otherwise, $B(S)_{i,j}$ is equal to 0. Then $flowtime(S) : Sched \rightarrow \mathbb{R}$ and $makespan(S) : Sched \rightarrow \mathbb{R}$ can be defined with the help of matrix multiplication as:

$$makespan(S) = \sum [B(S) \cdot ETC]_{j,j} \tag{6}$$

$$flowtime(S) = \max_{j \in \{1,\ldots,m\}} \sum [B(S) \cdot ETC]_{j,j} \tag{7}$$

Less formally, makespan equals to the sum of all elements on the main diagonal of $B(S) \cdot ETC$ and flowtime equals to maximal value on the main diagonal on $B(S) \cdot ETC$. An example of makespan and flowtime computation for a particular schedule of 3 tasks on 2 machines is shown in (8).

$$ETC = \begin{pmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{pmatrix}, \qquad S = \begin{pmatrix} 1 & 2 & 1 \end{pmatrix}, \qquad B(S) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

$$B(S) \cdot ETC = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 1 & 2 \end{pmatrix} \tag{8}$$

4.3 Related Work: Genetic Algorithms for Job Scheduling

Many nature insired meta-heuristic algorithms were suggested for the task scheduling problem [11, 10]. Among others, simmulated annealing, genetic algorithms, ant colony optimization and particle swarm optimization were used to find optimal schedules for the tasks in computational grids.

Genetic algorithms are a meta-heuristic optimization strategy that is perhaps closest to the differential evolution. In the following, we shortly summarize the differences and similarities between recent approaches to task scheduling based on the genetic algorithms and our implementation of the differential evolution for the task scheduling problem.

4.3.1 Encoding

In [10], two encoding schemes of the task schedules were considered. The direct representation encodes each schedule into a string of n integers $S = (s_1, s_2, \ldots, s_n)$. For the genetic algorithms, S can be used as a chromosome directly, whereas for DE, S can contain real values. It is translated to the schedule by concatenation of the elements of S.

The alternative permutation-based representation is more complex. For each machine m_i, the sequence S_i of jobs assigned to it is constructed. Next, the sequences S_i are concatenated. The resulting vector is a permutation of the jobs assigned to the machines. Such a representation requires maintaining additional information on the number of jobs assigned to each machine.

The direct representation is an intuitive and simple encoding of schedules. Moreover, there is a simple transformation from the direct representation to the permutation-based representation and vice-versa. Therefore, we have used only direct representation modified to suit DE in this study.

4.3.2 Fitness Function

The fitness function for the task scheduling problem in GA was based on the optimization of makespan and flowtime. The two criteria, establishing a multi-objective optimization problem, were in [10] merged using either a hierarchical or a simultaneous approach.

In the hierarchical approach, the criteria were prioritized (makespan received higher priority than flowtime) and in the optimization, criterion with higher importance was not allowed to vary while optimizing according to the criterion with lower importance.

In the simultaneous approach, the schedules were optimized according to both criteria at the same time and a weighted sum as in our fitness function (5) was used to create one scalar fitness value.

5 Experiments

We have implemented differential evolution for scheduling of independent tasks in heterogeneous independent environments. The differential evolution algorithm was implemented in its classic variant referred to as $DE/rand/1/bin$ [15]. To evaluate the performance of the DE for minimizing the makespan and flowtime, we have used the benchmark proposed in [2]. The simulation model is based on the ETC matrix for 512 jobs and 16 machines. The instances of the benchmark are classified into 12 different types of ETC matrices according to the following properties [2]:

- *task heterogeneity* – V_{task} represents the amount of variance among the execution times of tasks for a given machine
- *machine heterogeneity* – $V_{machine}$ represents the variation among the execution times for a given task across all the machines
- *consistency* – 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 machine M_k
- *inconsistency* – machine M_j may be faster than machine M_k for some tasks and slower for others

The DE algorithm was used with the parameters summarized in the Table 1. The parameters were set after brief initial tuning. The factor λ was set to 0.5 to have equal contribution of makespan and mean flowtime to the fitness value.

The experiments were conducted with two different settings for initial population. In the first case, whole initial population was generated randomly. In the second case, the initial population contained some vectors obtained by scheduling heuristics.

Table 1 A summary of DE parameters

Parameter	Value
Population size	20
Terminating generation	100000
Probability of crossover	$C = 0.9$
Scaling factor	$F = 0.1$
Makespan / flowtime ratio	$\lambda = 0.5$

5.1 Experiments with Random Initial Population

Each ETC matrix was named using the pattern $x - y - z$, where x describes task heterogeneity (*h*igh or *l*ow), y describes machine heterogeneity (*h*igh or *l*ow) and z describes the type of consistency (*i*ncosnsistent, *c*onsistent or *s*emiconsistent).

The Table 2 and the Table 3 show makespan and flowtime obtained by max-min heuristic, sufferage heuristic, min-min heuristic, and min-max heuristic. It illustrates that the min-max is the best heuristics to obtain optimal makespan for majority of the ETC matrices and min-min is the best heuristics to obtain optimal flowtime for the majority of the ETC matrices.

The Table 4a and the Table 4b show makespan and flowtime of schedules obtained by the differential evolution algorithm with random initial population.

As apparent from the tables, DE with random initial population cannot compete with domain specific heuristics when optimizing makespan. It ranks fourth and its results are usually better than max-min heuristics, but worse than sufferage heuristics, min-min heuristics and min-max heuristics.

Table 2 Makespan obtained by heuristic algorithms

ETC	max-min	sufferage	min-min	min-max
l-l-c	6753	5461	5468	**5310**
l-l-s	5947	3443	3599	**3327**
l-l-i	4998	2577	2734	**2523**
l-h-c	400222	333413	279651	**273467**
l-h-s	314048	163846	157307	**146953**
l-h-i	232419	121738	113944	**102543**
h-l-c	203684	170663	164490	**164134**
h-l-s	169782	105661	106322	**103321**
h-l-i	153992	**77753**	82936	77873
h-h-c	11637786	9228550	8145395	**7878374**
h-h-s	9097358	4922677	4701249	**4368071**
h-h-i	7016532	3366693	3573987	**2989993**

Table 3 Flowtime obtained by heuristic algorithms

ETC	max-min	sufferage	min-min	min-max
l-l-c	108014	86643	**80354**	84717
l-l-s	95091	54075	**51399**	52935
l-l-i	79882	40235	**39605**	39679
l-h-c	6400684	5271246	**3918515**	4357089
l-h-s	5017831	2568300	**2118116**	2323396
l-h-i	3710963	1641220	**1577886**	1589574
h-l-c	3257403	2693264	**2480404**	2613333
h-l-s	2714227	1657537	**1565877**	1640408
h-l-i	2462485	1230495	1214038	**1205625**
h-h-c	185988129	145482572	**115162284**	125659590
h-h-s	145337260	76238739	**63516912**	69472441
h-h-i	112145666	47237165	**45696141**	46118709

Table 4 Makespan and flowtime obtained by DE with random initial population

(a) Makespan obtained by DE with random initial population.

ETC	DE_{best}	DE_{avg}
l-l-c	7151	7303.2
l-l-s	4479	4582.2
l-l-i	3127	3203
l-h-c	451815	457741
l-h-s	212828	220334
l-h-i	141635	152186
h-l-c	212175	220142.2
h-l-s	141176	142405.2
h-l-i	99413	100307
h-h-c	13325802	13595908
h-h-s	6138124	6545734
h-h-i	4418167	454678

(b) Flowtime obtained by DE with random initial population.

ETC	DE_{best}	DE_{avg}
l-l-c	85422	891272.4
l-l-s	53675	53964.4
l-l-i	43941	44846.2
l-h-c	3783520	3788428
l-h-s	2277816	2383501
l-h-i	1890529	1935355.4
h-l-c	2699241	2765402.2
h-l-s	1597594	1625219.6
h-l-i	1359241	1380342
h-h-c	100921177	104753227
h-h-s	67874790	70281581
h-h-i	57808847	58216428

The Table 3 and the Table 4b show the flowtime of optimized schedules. In this case, the DE reached the best value for two of experimental matrices (l-h-c and h-h-c). Also in the other cases, DE delivered quite competitive results. Obviously, used setting of scheduling DE suited better to the optimization of flowtime.

5.2 Experiments with Optimized Initial Population

In the second set of experiments, the initial population of the DE was upgraded with vectors obtained by scheduling heuristics. Max-min heuristic, sufferage heuristic, min-min heuristic, and min-max heuristic were used to generate four vectors that were included in the initial population. Those vectors were superior to the remaining members of the initial population in terms of makespan and flowtime. The factor *lambda* was set to 0.9 in order to preserve the suboptimal makespan from the initial population because initial experiment showed the tendency to improve flowtime at the expense of good initial makespan. The results of the second round of the DE optimization experiments are summarized in the Table 5b and in the Table 5a respectively.

Table 5 Makespan and flowtime obtained by DE with upgraded initial population

(a) Makespan obtained by DE with upgraded initial population.

ETC	DE_{best}	DE_{avg}
l-l-c	5250	5271
l-l-s	3326	3326.8
l-l-i	2498	2502.2
l-h-c	267773	270912.4
l-h-s	146125	146759.4
l-h-i	100904	101254.6
h-l-c	159770	161262.2
h-l-s	101824	102440.2
h-l-i	76096	76297.4
h-h-c	7775829	7856042.8
h-h-s	4368071	4372414.6
h-h-i	2922633	2953782.6

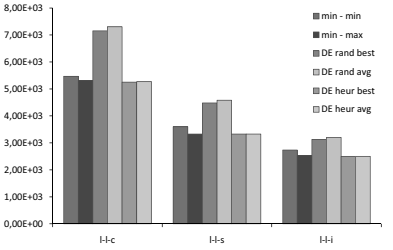
(b) Flowtime obtained by DE with upgraded initial population.

ETC	DE_{best}	DE_{avg}
l-l-c	79580	80785.4
l-l-s	52729	52754.8
l-l-i	39674	39724.6
l-h-c	3829129	3983780.4
l-h-s	2280929	2288328.2
l-h-i	1586502	1589414.8
h-l-c	2468081	2496781.6
h-l-s	1573431	1580786.8
h-l-i	1204845	1206638.4
h-h-c	114841390	118413991.8
h-h-s	64502140	67964923.8
h-h-i	45446258	45954812.2

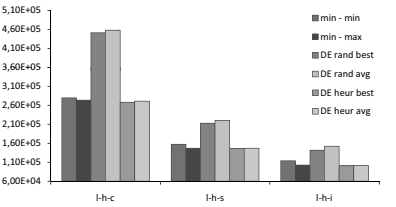
The best schedules obtained by DE with upgraded initial population were superior in terms of makespan in all cases. For all ETC matrices, except of h-h-s, also the average DE makespan outperformed makespans obtained by scheduling heuristics.

The flowtime obtained by the DE with the upgraded initial population was not the best in all cases. However, the differential evolution managed to optimize makespan and flowtime at once whereas the heuristic algorithms were not able to do that. Also, the value of *lambda* used during the experiment prioritized makespan.

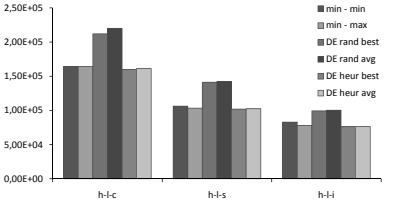
The makespan of optimized schedules is illustrated in Figure 2 and the flowtime of optimized schedules is shown in Figure 3. The series denoted ”DE rand” illustrates the results obtained by DE with random initial population. The series labeled ”DE heur” illustrates in the figures the values of makespan and flowtime obtained



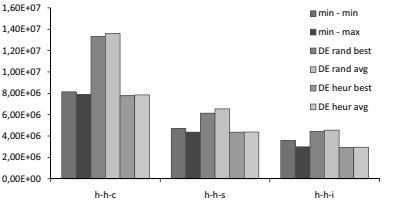
(a) Comparison of makespan for l-l-x matrices.



(b) Comparison of makespan for l-h-x matrices.

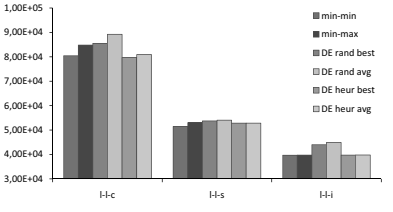


(c) Comparison of makespan for h-l-x matrices.

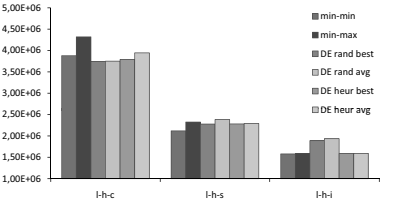


(d) Comparison of makespan for h-h-x matrices.

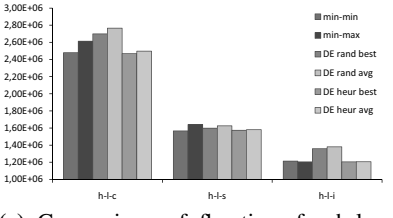
Fig. 2 Comparison of obtained makespan



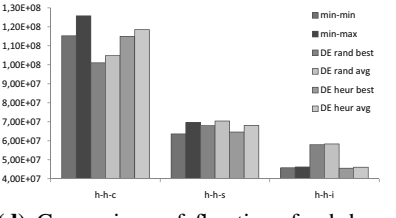
(a) Comparison of flowtime for l-l-x matrices.



(b) Comparison of flowtime for l-h-x matrices.



(c) Comparison of flowtime for h-l-x matrices.



(d) Comparison of flowtime for h-h-x matrices.

Fig. 3 Comparison of obtained flowtime

by DE with upgraded initial population. The results are compared to the schedules obtained by min-min and min-max heuristics.

5.3 Comparison with GA

The comparison of the schedules obtained by the DE with schedules obtained by the GA as presented in [10] is shown in the Table 6. We note that the comparison is rather illustrative and indirect since the DE implementetion was not adjusted to perform similar amount of work as the GA from [10] and the weight λ used in the fitness function was also different. Also, the heuristic algorithm with the best makespan is included in the comparison.

Table 6 The comparison of makespan and flowtime obtained by meta-heuristic algorithms DE and GA

(a) Makespan obtained by DE and GA.

ETC	min-max	DE_{avg}	GA_{hier}	GA_{simult}
l-l-c	5310	5271	5310.615	**5226.972**
l-l-s	3327	**3326.8**	3650.316	3515.526
l-l-i	2523	**2502.2**	2676.207	2613.11
l-h-c	273467	270912.4	252907.58	**248466.775**
l-h-s	146953	146759.4	137785.504	**127641.889**
l-h-i	102543	**101254.6**	108047.402	106069.101
h-l-c	164134	161262.2	157255.844	**155251.196**
h-l-s	103321	102440.2	101817.924	**98334.64**
h-l-i	77873	76297.4	77455.085	**75924.023**
h-h-c	7878374	7856042.8	7730973.882	**7610176.437**
h-h-s	4368071	4372414.6	4745284.938	**4359312.628**
h-h-i	2989993	**2953782.6**	3141395.822	3077705.818

(b) Flowtime obtained by DE and GA.

ETC	min-max	DE_{avg}	GA_{rib}	GA_{ss}
l-l-c	84717	**80785.4**	942076.61	920475.17
l-l-s	52935	**52754.8**	616542.78	605375.38
l-l-i	**39679**	39724.6	453399.32	446695.83
l-h-c	4357089	**3983780.4**	35677170.8	34767197.1
l-h-s	2323396	**2288328.2**	15992229.8	15644101.3
l-h-i	1589574	**1589414.8**	13108727.9	13444708.3
h-l-c	2613333	**2496781.6**	28314677.9	27687019.4
h-l-s	1640408	**1580786.8**	17007775.2	16598635.5
h-l-i	**1205625**	1206638.4	12902561.3	12775104.7
h-h-c	125659590	**118413991.8**	1073774996	1048333229
h-h-s	69472441	**67964923.8**	541570911	526866515
h-h-i	46118709	**45954812.2**	376800875	378010732

In the Table 6a, the column GA_{hier} shows the makespan obtained by the GA with the hierarchical fitness function while the column GA_{simult} shows the makespan obtained by the GA with the fitness function with simultaneous optimization of flowtime and makespan.

In the Table 6b, both GA variants used the fitness function with simultaneous optimization of both objectives that perfromed better for the makespan optimization. Two generation replacement startegies were considered in [10] for the GA. The GA with the replace only if better (GA_{rib}) strategy forms the new generation from the best individuals from the union of the set of parent and offspring chromosomes.

The GA with the steady state generation (GA_{ss}) replacement strategy replaces only the weakest individuals in the parent population by offspring chromosomes if they have better fitness.

We can see that the GA_{simult} evolved the schedule with the best makespan 8 times while the DE obtained the schedule with the best makespan 4 times. However, the DE obtained flowtime better than the GA in all cases and it obtained the best flowtime for 10 out of 12 ETC matrices.

6 Conclusions

This paper presents an algorithm for scheduling independent tasks on heterogeneous distributed environments based on the differential evolution. The algorithm was implemented and the experimental results suggest that it can deliver competitive results. With the random initial population, the algorithm managed to optimize schedules for few ETC matrices so that the flowtime was best.

Much better results were obtained when we upgraded the initial population with candidate solutions obtained by the heuristic algorithms. In such case, the algorithm managed to exploit the different sub-optimal solutions provided at the beginning and converged to better schedules.

In comparison to a recent genetic algorithm for the job scheduling problem, the DE has found schedule with better makespan several times but it was outperformed by the GA for some other ETC matrices. However, the schedules obtained by the GA featured better flowtime.

Presented algorithm has a number of parameters including C, F and λ. Fine tuning of DE parameters is subject of our future work.

Acknowledgements. This work was supported by the Czech Science Foundation under the grant no. 102/09/1494.

References

1. Ali, S., Braun, T., Siegel, H., Maciejewski, A.: Heterogeneous computing (2002)
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., Hensgen, D., Freund, R.F.: A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems (2001)

3. Tracy, M.M., Braun, T.D., Siegel, H.J.: High-performance mixed-machine heterogeneous computing. In: 6th Euromicro Workshop on Parallel and Distributed Processing, pp. 3–9 (1998)
4. Fernandez-Baca, D.: Allocating modules to processors in a distributed system. IEEE Trans. Softw. Eng. 15(11), 1427–1436 (1989)
5. Munir, E.U., Li, J.-Z., Shi, S.-F., Rasool, Q.: Performance analysis of task scheduling heuristics in grid. In: 2007 International Conference on Machine Learning and Cybernetics, vol. 6, pp. 3093–3098 (August 2007)
6. Izakian, H., Abraham, A., Snasel, V.: Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments. In: International Joint Conference on Computational Sciences and Optimization, CSO 2009, vol. 1, pp. 8–12 (April 2009)
7. Ritchie, G., Levine, J.: A hybrid ant algorithm for scheduling independent jobs in heterogeneous computing environments. In: Proceedings of the 23rd Workshop of the UK Planning and Scheduling Special Interest Group (December 2004)
8. YarKhan, A., Dongarra, J.: Experiments with scheduling using simulated annealing in a grid environment. In: Parashar, M. (ed.) GRID 2002. LNCS, vol. 2536, pp. 232–242. Springer, Heidelberg (2002)
9. Page, A.J., Naughton, T.J.: Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. Artificial Intelligence Review 24, 137–146 (2004)
10. Carretero, J., Xhafa, F., Abraham, A.: Genetic algorithm based schedulers for grid computing systems. International Journal of Innovative Computing, Information and Control 3(7) (2007)
11. Abraham, A., Liu, H., Grosan, C., Xhafa, F.: Nature Inspired Meta-heuristics for Grid Scheduling: Single and Multi-objective Optimization Approaches. Studies in Computational Intelligence, vol. 146, pp. 247–272. Springer, Heidelberg (2008)
12. Munir, E.U., Li, J., Shi, S., Zou, Z., Rasool, Q.: A performance study of task scheduling heuristics in hc environment. In: An, L.T.H., Bouvry, P., Tao, P.D. (eds.) MCO. Communications in Computer and Information Science, vol. 14, pp. 214–223. Springer, Heidelberg (2008)
13. 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., Siegel, H.J.: Scheduling resources in multi-user, heterogeneous, computing environments with smartnet. In: Heterogeneous Computing Workshop, vol. 0, p. 3 (1998)
14. Shoukat, M.M., Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. Journal of Parallel and Distributed Computing 59, 107–131 (1999)
15. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential Evolution A Practical Approach to Global Optimization. Natural Computing Series. Springer, Berlin (2005)
16. Jongen, H.T., Meer, K., Triesch, E.: Optimization Theory. Kluwer Academic Publishers, Dordrecht (2004)