Supplementary File of the TPDS Manuscript: A Hybrid Chemical Reaction Optimization Scheme for Task Scheduling on Heterogeneous Computing Systems

Yuming Xu, Kenli Li*, *Member, IEEE*, Ligang He, *Member, IEEE*, Longxin Zhang and Keqin Li, *Fellow, IEEE*

Abstract–This supplementary file contains the supporting materials of the TPDS manuscript – "A Hybrid Chemical Reaction Optimization Scheme for Task Scheduling on Heterogeneous Computing Systems." It improves the completeness of the TPDS manuscript.

1 Related Works

In this section, we discuss related works on heuristic scheduling and meta-heuristic scheduling in subsections 1.1 and 1.2, respectively.

1.1 Heuristic-based Algorithms

Heuristic methods usually provide good solutions for restricted instances of a task scheduling problem. They can find a near optimal solution in polynomial time. A heuristic method searches a path in the solution space and ignores other possible paths [1], [2]. The heuristic-based group consists of three subcategories, which are list scheduling [1], [3], cluster scheduling [4], [5], and duplication-based scheduling [6].

List scheduling is the most popular among these three when referring to scheduling DAG applications. A list scheduling algorithm is usually divided into two steps. In the first step, list scheduling maintains an ordered list of tasks within a DAG application according to some greedy heuristics. In the second step, the task is selected in a specified order (the highest priority) for mapping to the most suitable computing processor, which provides the earliest start time. List scheduling algorithms produce the most efficient schedule without deteriorating the makespan and with a reasonable time complexity. The major difference among different list scheduling algorithms is the means by which priorities are assigned and the most suitable computing node is selected.

- *The authors are with the College of Computer Science and Electronic Engineering, Hunan University, and with the National Supercomputing Center in Changsha, Hunan, Changsha, 410082, China. Corresponding author: Kenli Li, Email: lkl@hnu.edu.cn.*
- *L. He is also with the Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom.*
- *Keqin Li is also with the Department of Computer Science, State University of New York, New Paltz, NY 12561, USA*

*Manuscript received ****, 2014; revised ****, 2014.*

Clustering algorithm [4], [5] is another type of heuristic algorithm, which is mainly proposed for homogeneous systems and aims to form clusters of tasks that are assigned to processors. Clustering heuristic algorithm assumes that there are an unlimited number of computing nodes available to task executions. The clustering algorithm will use as many computing nodes as possible in order to reduce the *makespan* of a schedule. If the number of computing nodes used by a schedule is more than the number of nodes available, then the mapping process is required to merge the tasks in a candidate schedule onto those available nodes.

The duplication-based scheduling heuristic [6] attempts to reduce communication delays by executing the key tasks on a same node or several nodes with less communication cost. Duplication-based scheduling essentially aims to further improve the performance of list scheduling, which produces the shortest makespan. However, it often has two disadvantages: the first one is the higher time complexity, such as three times the number of tasks; and the second, they have lower efficiency because the main strategy is to duplicate the execution of tasks, which results in more power and resource consumption.

In this paper, we combine the CRO technique with the heuristic algorithm to try to find a balance between solution quality and time overhead of DAG task scheduling. In the proposed scheduling technique, we use CRO to search execution order of tasks and use a heuristic method to determine the most suitable computing node for a selected task.

1.2 Meta-heuristic

Contrary to heuristic-based algorithms, the meta-heuristic algorithms are used for combinatorial optimization in which an optimal solution is sought over a discrete search-space. The meta-heuristic algorithms have been proposed to solve a variety of task scheduling problems, and proven to be a kind of robust algorithm on the DAG task scheduling

problem. They incorporate a combinatorial process when searching for solutions. The meta-heuristic algorithms are basically classified into two categories by the type of search strategy [7]. One type of search strategy is an improvement on simple local search algorithms; Meta-heuristics of this type include *simulated annealing* [8], *tabu search* [9], [10], etc. The other type of search strategy has a learning component to the search process; meta-heuristics of this type include *ant colony optimization* [11], [12], [13], *evolutionary algorithm* [14], [15], *genetic algorithms* [16], [17], [18], [19], [20], [21], and *particle swarm optimization* [22], [23].

Recently, a new meta-heuristic method, named *Chemical reaction optimization* (CRO) , has been proposed [24], [25], [26], [27]. The method encodes solutions as molecules, and mimics the interactions of molecules in chemical reactions to search the optimal solutions. In other words, in CRO, all of solutions are encoded, and then a sequence of operations are performed over these solutions by loosely simulating the molecule behaviors in reaction. Gradually, good solutions will emerge. CRO has already shown its power in solving problems like *resource-constrained project scheduling problem* (RCPSP), *channel assignment problem* (CAP), and *quadratic assignment problem* (QAP) [24], [27]. CRO has also been applied to solve task scheduling problems [25], [26]. However, to date, it has only been used to encode scheduling for independent tasks on heterogeneous computing systems.

Compared with scheduling independent tasks, the challenge of applying CRO to handle DAG scheduling is that in DAG scheduling, both task execution order and task-to-processor mapping need to be considered. In order to solve the dependent task scheduling problem, we have been investigating the problem of applying CRO to address the DAG scheduling. Our previous work presented in [28] applied the conventional CRO framework to address the DAG scheduling. In both the work in [28] and the work presented in this paper, the DAG scheduling is divided into two phases: 1) determining the execution order of the tasks in a DAG, and 2) mapping the tasks to computing nodes. In [28], we developed a double molecular-based CRO (DMSCRO), i.e., designed and applied the CRO operations to both phases. During the work, we realized that although the developed DMSCRO is able to consistently produce good scheduling solutions, and the time overhead, i.e., time spent in finding a good solution, is high. In order to reduce the time overhead, this paper integrates the heuristic approach into the CRO technique. In this paper, we design and apply the CRO operations to the first phase of the DAG scheduling, i.e., determining the execution order of the tasks, and then design and develop the heuristic approach to determine the task-to-processor mapping.

2 Background of CRO

In this section, we introduce the background knowledge of CRO [24], [25], [29]. In CRO, a molecule has a unique structure which represents a solution of an optimization problem, and has itself *kinetic energy (KE)* and *potential energy (PE)*, respectively, which are two key properties attached to the molecule structure. The former is used to control the acceptance of new solutions with worse fitness and the latter corresponds to a *fitness* value of the solution.

CRO mimics the process of a chemical reaction where molecules undergo a sequence of reactions between each other or with the environment in a closed container. For example, suppose ω and f are a molecular structure (solution) and a fitness function, respectively. Then $PE_\omega = f(\omega)$. *KE* is a non-negative number and it helps the molecule escape from local optimums. During a reaction, a molecule structure, ω, attempts to change to another molecule structure ω' if $PE_{\omega'} \leq PE_\omega$, or $PE_{\omega'} \leq PE_\omega + KE_\omega$.

A central energy buffer is also implemented in CRO algorithm. In the environment, the energy stored in the buffer can be regarded as the energy in the closed container. The energy may also flow between the molecules and the energy buffer.

During a CRO process, the following four types of elementary reactions are likely to happen.

- On-wall ineffective collision: this reaction is an uni-molecule reaction, whose reactant involves only one molecule. When a molecule ω collides onto the wall of the closed container, it is allowed to change to another molecule ω', if Eq. (1) related to their energy values holds,

$$PE_\omega + KE_\omega \geq PE_{\omega'}. \quad (1)$$

  After collision, the *KE* energy will be re-distributed. A certain portion of *KE* of the new molecule will be withdrawn to the central energy buffer (i.e., environment). The *KE* energy of the new module can be calculated in Eq. (2),

$$KE_{\omega'} = (PE_\omega - PE_{\omega'} + KE_\omega) \times a, \quad (2)$$

  where a is a number randomly selected from the range of $[KE\_LossRate, 1]$. $KE_{LossRate}$ is the loss rate of the *KE* energy, which is a system parameter set during the initialization stage of CRO.
- Decomposition: this reaction is also an uni-molecule reaction. A molecule ω can decompose into two new molecules, ω'_1 and ω'_2, if Eq. (3) holds,

$$PE_\omega + KE_\omega + buffer \geq PE_{\omega'_1} + PE_{\omega'_2}, \quad (3)$$

  where $buffer$ denotes the energy stored in the central buffer, which represents the energy interactions between molecules and the environment. Let $E_{dec} = (PE_\omega + KE_\omega) - (PE_{\omega'_1} + PE_{\omega'_2})$. Then after decomposition, the *KE* energies of ω'_1 and ω'_2 are calculated by Eqs. (4) and (5),

$$KE_{\omega'_1} \leftarrow (E_{dec} + buffer) \times \delta_1 \times \delta_2, \quad (4)$$

$$KE_{\omega'_2} \leftarrow (E_{dec} + buffer - KE_{\omega'_1}) \times \delta_3 \times \delta_4, \quad (5)$$

  where $\delta_1, \delta_2, \delta_3, \delta_4$ is a random number generated between 0 and 1.
  The energy in the buffer is updated by Eq. (6),

$$buffer \leftarrow E_{dec} + buffer - (PE_{\omega'_1} + PE_{\omega'_2}). \quad (6)$$

- Inter-molecular ineffective collision: this reaction is an inter-molecule reaction, whose reactants involve two molecules. When two molecules, ω_1 and ω_2,

collide into each other, they can change to two new molecules, ω_1' and ω_2', if Eq. (7) holds,

$$PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} \geq PE_{\omega_1'} + PE_{\omega_2'}. \quad (7)$$

E_{inter} denotes the spare energy after the inter-molecule collision, which can be calculated by Eq. (8),

$$\begin{aligned} E_{inter} =& (PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2}) \\ & - (PE_{\omega_1'} + PE_{\omega_2'}). \end{aligned} \quad (8)$$

The *KEs* of the new molecules will share the spare energy. Therefore, The *KEs* of ω_1' and ω_2' are calculated by Eqs. (9) and (10),

$$KE_{\omega_1'} \leftarrow E_{inter} \times \delta_1, \quad (9)$$

$$KE_{\omega_2'} \leftarrow E_{inter} \times (1 - \delta_1), \quad (10)$$

where δ_1 is a random number generated from [0, 1].

- Synthesis: this reaction is also an inter-molecule reaction. When two molecules, ω_1 and ω_2, collide into each other, they can be combined to generate a new molecule, ω', if Eq. (11) holds,

$$PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} \geq PE_{\omega'}. \quad (11)$$

The *KE* energy of ω' is calculated by Eq. (12),

$$KE_{\omega'} = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'}. \quad (12)$$

The fitness of a solution is judged by the *PE* energy of the molecule.

The typical execution flow of CRO is as follows.

- The CRO is first initialized to set some system parameters, such as the size of the populations (molecules), $KE_{LossRate}$, *InitialKE* (the initial energy associated to molecules), *buffer* (initial energy in the energy buffer), *MoleColl* (It is used later in the process to determine whether to perform a uni-molecular or an inter-molecular operation), etc.
- Then the process enters a loop. In each iteration, the process first decides whether to perform uni-molecular operations or inter-molecular operations following on a certain probability. It is decided in the following way. A random number, b, is generated in the interval [0, 1]. If b is bigger than the value of *MoleColl* that the process sets in the initialization stage, an uni-molecular operation will be performed; otherwise, an inter-molecular operation will take place. If it is an uni-molecular operation, the process randomly selects a certain number of molecules and then further decides whether to perform on-wall collision or decomposition according to Eqs. (1) and (3). Similarly, if the process decides to perform the inter-molecular operations, the process then further decides whether to perform inter-molecular collision or synthesis according to Eqs. (7) and (11). At the end of the iteration, the process checks whether a new better solution is found by checking the PE energy of each newly generated molecule.
- The iteration process repeats until the stopping criteria satisfies (e.g., the best solution does not change for a certain number of consecutive iterations).

3 The Heuristic Method to Perform Task-to-Processor Mappings and the Framework of HCRO

3.1 The Heuristic Method to Perform Task-to-Processor Mappings

The earliest finish time of the exit task will be the makespan of the DAG job. The earliest finish time of a task relates to its *earliest start time*. The earliest start and finish times of a task is calculated as follows.

$EST(T_i, P_k)$ denotes the earliest start time of task T_i on the computing node P_k, which can be calculated with Eq. (13),

$$EST(T_i, P_k) = \begin{cases} 0, & T_i = T_{entry} \\ \max\limits_{T_j \in Pred(T_i)} AFT(T_j, P_l), & P_k = P_l \\ \max\limits_{T_j \in Pred(T_i)} (AFT(T_j, P_l) + C(T_j, T_i)), & P_k \neq P_l. \end{cases} \quad (13)$$

where $EFT(T_i, P_k)$ denotes the earliest finish time of task T_j on the computing node P_m.

The *Earliest Finish Time* (EFT) of node T_i on processor P_k is represented as $EFT(T_i, P_k)$, shown in Eq. (14),

$$EFT(T_i, P_k) = AST(T_i, P_k) + W(T_i, P_k). \quad (14)$$

The *Actual Start Time* (AST) of node T_i on processor P_k is represented as $AST(T_i, P_k)$, shown in Eq. (15),

$$AST(T_i, P_k) = \max(EST(T_i, P_k), Avail(P_k)), \quad (15)$$

where $Avail(P_k)$ is defined as the earliest time at which the processor P_k is ready for the task execution.

The *Actual Finish Time* (AFT) of a subtask T_i over all processors is represented as $AFT(T_i)$, shown in Eq. (16),

$$AFT(T_i) = \min_{1 \leq k \leq m} EFT(T_i, P_k). \quad (16)$$

Using Eq. (16), we can calculate the earliest finish time of the exit task of the DAG job, which is the makespan of the DAG job for the given execution order of the tasks.

Since meta-heuristic scheduling works by searching a good solution in the solution space, the size of solution space will largely determine the time spent by a meta-heuristic scheduling method to find a desirable solution. Theorem 1 below analyze the size of the solution space under the heuristic CRO method proposed in this paper and the pure CRO process, respectively.

Theorem 1. *When a pure CRO randomly maps n tasks without dependency on m heterogeneous computing nodes, the solution space is $n! \times m^n$.*

Proof. In the pure CRO scheduling, a solution includes both task execution order and randomly task-to-processor mapping. For any task in an execution order, there is m computing node mapping possibilities. Therefore, for any execution of n tasks, there is total m^n mapping possibilities.

Since there are $n!$ execution orders, the total number of solutions is $n! \times m^n$. □

For task-to-processor mapping, the heuristic-based *Heterogeneous Earliest Finish Time* (HEFT) is proposed to search for a solution in order to minimize $makespan$ without violating precedence constraints. The HEFT algorithm selects the subtask with the highest upward rank value at each step and assigns the selected subtask to the processor, which minimizes its *earliest finish time* with an insertion-based approach [3]. HEFT algorithm is a performance-effective and low-complexity task scheduling on heterogeneous computing systems. The time complexity of HEFT algorithm is $O(e \times m)$ for *e* edges and *m* processors. For a dense graph when the number of edges is proportional to $O(n^2)$ (*n* is the number of subtasks), the time complexity is on the order of $O(n^2 \times m)$ [3].

In this paper, in order to accelerate convergence speed of HCRO algorithm for DAG job scheduling, HEFT algorithm is adopted to realize the performance-effective and low-complexity approach which avoids less effective task-to-processor mapping. A detailed description about the task-to-processor mapping is given in Algorithm 2.

Algorithm 1 Task-to-Processor Heuristic Mapping

Input:
The task priority queue.
Output:
The schedule scheme and the makespan.
1: Add all subtasks (atoms) T_i to the *priority queue* according to their priority;
2: **while** $PriorityQueue \neq \emptyset$ **do**
3: Select the first subtask T_i from the priority queue for scheduling;
4: **for** each processor P_k in the processor set **do**
5: Compute $AFT(T_i, P_k)$ value using the insertion based HEFT scheduling policy;
6: Assign subtask T_i to the processor P_k that minimizes $EFT(T_i)$;
7: **end for**
8: Remove T_i from priority queue;
9: **end while**
10: **return** *makespan*= $AFT(T_{exit})$.

As can be seen from Theorem 1, the size of search space of the heuristic CRO method proposed in this paper is much smaller than that of a pure CRO method. Therefore, it is expected that the heuristic CRO method spends less time than the pure CRO method to find a desirable solution.

3.2 The Framework of Combining CRO and the Heuristic Method

After execution orders are generated according to the CRO method, the heuristic method presented in Subsection 3.1 is applied to calculate the makespan related to a given execution order. The makespan is used as the *PE* energy of the solution, that is, the makespan is used as the fitness function of execution order. The entire algorithm for scheduling a DAG job is outlined in Algorithm 2. The first step of the algorithm initializes the CRO process. Then the process enters a loop. In each iteration, the CRO process is applied to generate new execution orders (Steps 3-18). Then the heuristic algorithm is used to perform the task-to-processor mapping for each newly generate execution order (Steps 20-27) and consequently the corresponding makespan is obtained (Step 28). The procedure repeats until the stopping criteria is satisfied.

Algorithm 2 Heuristic CRO Scheduling

Input:
A DAG job.
Output:
A scheduling solution with minimal makespan.
1: Initializing the CRO process;
2: **while** The stopping criteria is not satisfied **do**
3: Generate $b \in [0, 1]$
4: **if** $b > MoleColl$ **then**
5: Randomly select one molecule ω;
6: **if** Eq. (3) holds **then**
7: Perform the decomposition operation on ω to generate two new molecules;
8: **else**
9: Perform the Onwall Ineffective Collision on ω to generate a new molecule;
10: **end if**
11: **else**
12: Randomly select two molecules ω_1 and ω_2;
13: **if** Eq. (11) holds **then**
14: Perform the synthesis operation on ω_1 and ω_2 to generate a new molecule;
15: **else**
16: Perform Intermolecular Ineffective Collision on ω_1 and ω_2 to generate two new molecules;
17: **end if**
18: **end if**
19: **for** each newly generated molecule ω_i' **do**
20: **while** The queue of ω_i' is not empty **do**
21: Select the first task T_i from the queue of ω_i';
22: **for** each computing node P_k in the computing system **do**
23: Call Eq.(14) to compute $EFT(T_i, P_k)$;
24: **end for**
25: Call Eq.(16) to map task T_i to computing node P_k that provides minimal $AFT(T_i)$;
26: Remove the scheduled task from ω_i';
27: **end while**
28: *makespan*= $AFT(T_{exit})$;
29: **end for**
30: Record ω_i' which generates the minimal *makespan* and the corresponding task-to-processor mapping;
31: **end while**
32: **Return** the scheduling solution with the minimal makespan.

4 Size of Molecular Population and A Good Uniform Coverage

In this section, we elaborate how to generate a good "seeding", good uniform coverage, and molecular diversity in our approach in subsection 4.1, and we discuss the problem of size of initial molecular population in subsection 4.2, respectively.

4.1 A Good Uniform Coverage

A desirable property for an initial molecular population is a *good uniform coverage*. A good uniform coverage is desired, because information is obtained throughout the whole feasible solution space. Therefore, by a good uniform coverage, the molecules are well spread out to cover the whole feasible solution space. The molecules have a good

uniform coverage if they do not form clusters or leave relatively large areas of the feasible solution space unexplored.

In this paper, in order to achieve a good uniform coverage, initially, for the task scheduling problem, the good "seeding" molecules are generated by three heuristic rank policies (upward rank, downward rank, and a combination of level and upward-downward rank), which are the mostly used by traditional list scheduling approaches to estimating the priority of each subtask, respectively, as shown in Table 1.

TABLE 1
Task Priorities of DAG Application in Fig.1 of The Main Paper

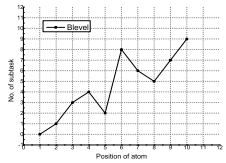
T_i	$Rank_b$	$Rank_t$	$Rank_{b+t}$	$Level$
0	74.3	0.0	74.3	0
1	57.3	12.67	70.0	1
2	49.7	24.67	74.3	1
3	51.0	18.67	69.7	1
4	51.0	11.67	62.7	1
5	28.3	24.7	53.0	1
6	29.3	40.7	70.0	2
7	20.3	32.7	53.0	2
8	34.3	40.0	74.3	2
9	14.3	60.0	74.3	3

Then, the priority queues (molecules) are generated by selecting an atom in the priority queues for these molecules from left to right, where an atom is selected and inserted into a right position without violating the precedence constraints in order to make the largest Hamming distance of two molecules. In other words, this approach uses the criterion of maximizing the Hamming distance between the solution (new molecule) under consideration and the solution (seeding molecule) already generated before. Molecular diversity of the initial population can help the CRO be able to reach part of the feasible solution space as large as possible. Genetically more diversified initial populations are preferable. Finally, the rest priority queues are generated by selecting randomly an atom in the priority queues for these molecules, where an atom is randomly selected in the selected molecule and inserted into a right position without violating the precedence constraints in order to make the largest hamming distance of two molecules. The new generated molecule is added into the molecular population in order to effectively improve molecular diversity of the population. Illustration of the initial population is in shown in Fig. 1.

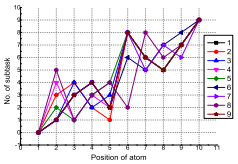
4.2 Size of Molecular Population

The molecular population consists of $PopSize$ molecules. The size of the initial population has been investigated that the underlying idea is always of a trade-off between efficiency and effectiveness. Intuitively, it would seem that there should be some "optimal" value for a given size, on the grounds that a too small molecular population would not allow sufficient room for exploring the search space effectively, while a too large molecular population would so impair the efficiency of the method that no solution could be expected in a reasonable amount of time.

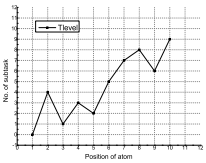
Initial molecular population size, a slightly different question that we could ask is regarding a minimum population size for a meaningful search to take place. In Colin R. Reeves paper [30], the initial principle was adopted that, at the very least, every point in the search space should be reachable from the initial population by crossover only. This requirement can only be satisfied if there is at least one instance of every allele at each locus in the whole population of strings. This helps to prevent premature convergence.



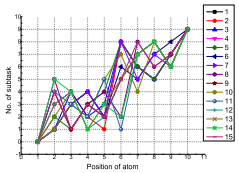
(a) *The good 'seeding' being obtained by the heuristic Blevel approach. The task priority queue is (0, 1, 3, 4, 2, 8, 6, 5, 7, 9). Sort $Rank_B$ in descending order.*



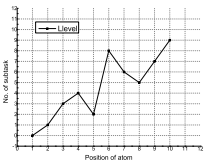
(b) *A good uniform coverage being generated by selecting an atom in the priority queues for these molecules from left to right according to the largest Hamming distance between the new molecule and the good original seed in Fig. 1.(a)*



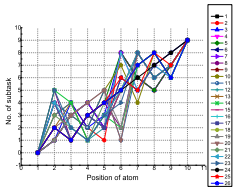
(c) *The good 'seeding' being obtained by the heuristic Tlevel approach. The task priority queue is (0, 4, 1, 3, 2, 5, 7, 8, 6, 9). Sort $Rank_T$ in ascending order.*



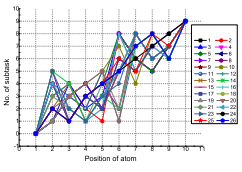
(d) *A good uniform coverage being generated by selecting an atom in the priority queues for these molecules from left to right according to the largest Hamming distance between the new molecule and the good original seed in Fig. 1.(c)*



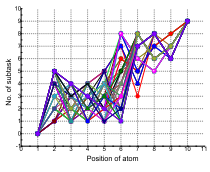
(e) *The good 'seeding' being obtained by the heuristic Llevel&(Blevel+Tlevel) approach. The task priority queue is (0, 2, 1, 3, 4, 5, 8, 6, 7, 9). Sort elements in descending order when they are the same level.*



(f) *A good uniform coverage being generated by selecting an atom in the priority queues for these molecules from left to right according to the largest Hamming distance between the new molecule and the good original seed in Fig. 1.(e)*



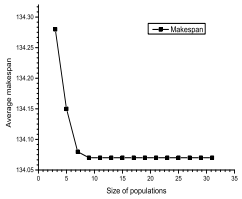
(g) *The sum of individuals with Blevel, Tlevel and Llevel&(Blevel+Tlevel).*



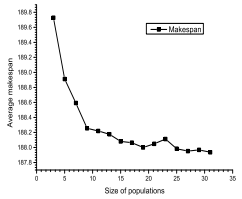
(h) *The initial population with a good uniform coverage and molecular diversity. The population of size is ten times the number of nodes of DAG application.*

Fig. 1. Illustration of the initial population for a simple DAG graph of size 10 (10 tasks).

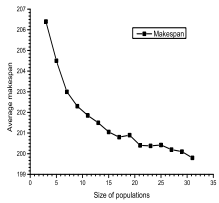
In our experiments, we maintain those parameters unchanged, such as *InitialKE* = 10000, *buffer*=2000, *MoleColl* = 0.2, $KE_{LossRate}$ = 0.5, and a stopping criterion. By increasing the size of the population, we obtain the average makespan for different sizes of DAG applications (10, 20, 50), each with 30 independent runs. The results are shown in Fig. 2.



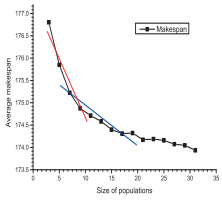
(a) *The DAG graph of size 10+2 (10 subtasks, 2 dummy nodes), 30 independent runs, 100 DAG applications with different characteristics, processors=3.*



(b) *The DAG graph of size 20+2 (20 subtasks, 2 dummy nodes), 30 independent runs, 100 DAG applications with different characteristics, processors=8.*



(c) *The DAG graph of size 50+2 (50 subtasks, 2 dummy nodes), 30 independent runs, 100 DAG applications with different characteristics, processors=16.*



(d) *The average makespan of the three classes DAG graphs.*

Fig. 2. Illustration of the average makespan vs. the size of initial population.

In Fig. 2(d), we can observe that the makespan decreases as the size of population increases. When the size of population is twenty times more than the number of nodes in a DAG application, the makespan remains almost the same (< 0.5). When the size of population is more than the number of nodes in a DAG application by between ten and twenty times, the makespan decreases a little (< 1.0). Therefore, considering the trade-off between efficiency and effectiveness, the size of initial population is set as ten times the number of nodes in a DAG application.

APPENDIX A
DEFINITIONS OF NOTATIONS

In this section, a list of notations and their definitions used in the paper is provided.

TABLE 2
Definitions of Notations

Notation	Definition
$AFT(T_i)$	The actual finish time of task T_i
$AST(T_i)$	The actual start time of task T_i
$Avail(P_k)$	The earliest time when processor P_k is ready for task execution
B	The two-dimensional matrix of communication bandwidths between processors
$B(P_k, P_l)$	The communication bandwidth between processors P_k to P_l
CCR	The communication to computation ratio, i.e., the ratio of the average communication cost to the average computation cost
$C_d(T_i, T_j)$	The amount of communication between task T_i and task T_j
$C_s(P_k)$	The communication startup cost of processor P_k
$C(T_i, T_j)$	The communication cost from task T_i (scheduled on P_k) to task T_j (scheduled on P_l)
$\overline{C(T_i, T_j)}$	The average communication cost of the $edge(T_i, T_j)$
E	A set of e edges
$EFT(T_i, P_k)$	The earliest finish time of task T_i on processor P_k
$EST(T_i, P_k)$	The earliest start time of task T_i on processor P_k
P	A set of m heterogeneous processors
P_k	The *k*-th processor in the system
$Pred(T_i)$	A set of the immediate predecessors of the task T_i
$Rank_b(T_i)$	The upward rank of task T_i
$Rank_t(T_i)$	The downward rank of task T_i
$S(T_i, P_k)$	The estimated execution speed of task T_i on processor P_k
$Succ(T_i)$	A set of the immediate successors of the task T_i
T	A set of n weighted tasks in an application
T_{entry}	The starting task without any predecessors
T_{exit}	The final task with no successors
T_i	The *i*-th task in the application
$W_d(T_i)$	The computational data of task T_i
$W(T_i, P_k)$	The computational cost of task T_i on the processor P_k
$\overline{W(T_i)}$	The average computational cost of task T_i

REFERENCES

[1] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, Dec. 1999. [Online]. Available: http://doi.acm.org/10.1145/344588.344618

[2] A. Zomaya, C. Ward, and B. Macey, "Genetic scheduling for parallel processor systems: comparative studies and performance issues," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, no. 8, pp. 795–812, aug 1999.

[3] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performanceeffective and lowcomplexity task scheduling for heterogeneous computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 3, pp. 260–274, mar 2002.

[4] A. Amini, T. Y. Wah, M. Saybani, and S. Yazdi, "A study of density-grid based clustering algorithms on data streams," in *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, vol. 3, july 2011, pp. 1652–1656.

[5] H. Cheng, "A high efficient task scheduling algorithm based on heterogeneous multi-core processor," in *Database Technology and Applications (DBTA), 2010 2nd International Workshop on*, nov. 2010, pp. 1–4.

[6] T. Tsuchiya, T. Osada, and T. Kikuno, "A new heuristic algorithm based on gas for multiprocessor scheduling with task duplication," in *Algorithms and Architectures for Parallel Processing, 1997. ICAPP 97., 1997 3rd International Conference on*, dec 1997, pp. 295–308.

[7] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, sep 2003.

[8] M. Kashani and M. Jahanshahi, "Using simulated annealing for task scheduling in distributed systems," in *Computational Intelligence, Modelling and Simulation, 2009. CSSim '09. International Conference on*, sept. 2009, pp. 265–269.

[9] R. Shanmugapriya, S. Padmavathi, and S. Shalinie, "Contention awareness in task scheduling using tabu search," in *Advance Computing Conference, 2009. IACC 2009. IEEE International*, march 2009, pp. 272–277.

[10] Y. W. Wong, R. Goh, S.-H. Kuo, and M. Low, "A tabu search for the heterogeneous dag scheduling problem," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, dec. 2009, pp. 663–670.

[11] A. Tumeo, C. Pilato, F. Ferrandi, D. Sciuto, and P. Lanzi, "Ant colony optimization for mapping and scheduling in heterogeneous multiprocessor systems," in *Embedded Computer Systems: Architectures, Modeling, and Simulation, 2008. SAMOS 2008. International Conference on*, july 2008, pp. 142–149.

[12] F. Ferrandi, P. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 29, no. 6, pp. 911–924, june 2010.

[13] B. Cheng, Q. Wang, S. Yang, and X. Hu, "An improved ant colony optimization for scheduling identical parallel batching machines with arbitrary job sizes," *Applied Soft Computing*, vol. 13, no. 2, pp. 765–772, 2013.

[14] D. Hadka and P. Reed, "Diagnostic assessment of search controls and failure modes in many-objective evolutionary optimization," *Evolutionary Computation*, vol. 20, no. 3, pp. 423–452, Sept 2012.

[15] J. Secretan, N. Beato, D. D'Ambrosio, A. Rodriguez, A. Campbell, J. Folsom-Kovarik, and K. Stanley, "Picbreeder: A case study in collaborative evolutionary exploration of design space," *Evolutionary Computation*, vol. 19, no. 3, pp. 373–403, Sept 2011.

[16] E. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 5, no. 2, pp. 113–120, feb 1994.

[17] S. Song, K. Hwang, and Y.-K. Kwok, "Risk-resilient heuristics and genetic algorithms for security-assured grid job scheduling," *Computers, IEEE Transactions on*, vol. 55, no. 6, pp. 703–719, june 2006.

[18] J. Wang, Q. Duan, Y. Jiang, and X. Zhu, "A new algorithm for grid independent task schedule: Genetic simulated annealing," in *World Automation Congress (WAC), 2010*, sept. 2010, pp. 165–171.

[19] K. Dahal, A. Hossain, B. Varghese, A. Abraham, F. Xhafa, and A. Daradoumis, "Scheduling in multiprocessor system using genetic algorithms," in *Computer Information Systems and Industrial Management Applications, 2008. CISIM '08. 7th*, june 2008, pp. 281–286.

[20] M. Mehrjoo, N. Khaji, and M. Ghafory-Ashtiany, "Application of genetic algorithm in crack detection of beam-like structures using a new cracked eulercbernoulli beam element," *Applied Soft Computing*, vol. 13, no. 2, pp. 867–880, 2013.

[21] T. Lu and J. Zhu, "A genetic algorithm for finding a path subject to two constraints," *Applied Soft Computing*, vol. 13, no. 2, pp. 891–898, 2013.

[22] T. Chen, B. Zhang, X. Hao, and Y. Dai, "Task scheduling in grid based on particle swarm optimization," in *Parallel and Distributed Computing, 2006. ISPDC '06. The Fifth International Symposium on*, july 2006, pp. 238–245.

[23] H. Li, L. Wang, and J. Liu, "Task scheduling of computational grid based on particle swarm algorithm," in *Computational Science and Optimization (CSO), 2010 Third International Joint Conference on*, vol. 2, may 2010, pp. 332–336.

[24] A. Lam and V. Li, "Chemical-reaction-inspired metaheuristic for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 14, no. 3, pp. 381–399, june 2010.

[25] J. Xu, A. Lam, and V. Li, "Chemical reaction optimization for the grid scheduling problem," in *Communications (ICC), 2010 IEEE International Conference on*, may 2010, pp. 1–5.

[26] ——, "Chemical reaction optimization for task scheduling in grid computing," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 10, pp. 1624–1631, oct. 2011.

[27] A. Y. S. Lam, V. O. K. Li, and J. J. Q. Yu, "Real-coded chemical reaction optimization," *Evolutionary Computation, IEEE Transactions on*, vol. PP, no. 99, p. 1, 2011.

[28] Y. Xu, K. Li, L. He, and T. K. Truong, "A DAG scheduling scheme on heterogeneous computing systems using double molecular structure-based chemical reaction optimization," *Journal of Parallel and Distributed Computing*, vol. 73, no. 9, pp. 1306–1322, 2013.

[29] T. K. Truong, K. Li, and Y. Xu, "Chemical reaction optimization with greedy strategy for the 01 knapsack problem," *Applied Soft Computing*, vol. 14, no. 1, pp. 101–107, 2013.

[30] C. R. Reeves, "Using genetic algorithms with small populations," in *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1993, pp. 92–99.