2012 10th IEEE International Symposium on Parallel and Distributed Processing with Applications

Chemical Reaction Optimization for Heterogeneous Computing Environments

Kenli Li, Zhimin Zhang, Yuming Xu
College of Information Science and Engineering
Hunan University
Changsha, Hunan, China
Email: lkl510@263.net
121862525@qq.com
xxl1205@163.com

Bo Gao, Ligang He
Department of Computer Science
University of Warwick
United Kingdom
Email: bogao@dcs.warwick.ac.uk
liganghe@gmail.com

***Abstract*—Task scheduling has been proven to be NP-hard problem [1] and we can usually approximate the best solutions with some classical algorithm, such as Heterogeneous Earliest Finish Time(HEFT), Genetic Algorithm. However, the huge types of scheduling problems and the small number of generally acknowledged methods mean that more methods are needed. In this paper, we propose a new method to schedule the execution of a group of dependent tasks for heterogeneous computing environments. The algorithm consists of two elements: An intelligent approach to assign the execution orders of tasks by task level, and an allocation algorithm based on chemical-reaction-inspired metaheuristic called Chemical Reaction Optimization (CRO) to map processors to tasks. The experiments show that the CRO-based algorithm performs consistently better than HEFT and Critical Path On a Processor (CPOP) without incurring much computational cost. Multiple runs of the algorithm can further improve the search result.**

***Keywords*-task scheduling; chemical reaction optimization; DAG; heterogeneous computing;**

I. Introduction

Scheduling a group of dependent tasks on parallel processors is an intensively studied problem in parallel computing. By decomposing a computation into smaller tasks and executing the tasks on multiple processors, we can potentially reduce the total execution time of the computation.

Traditional scheduling problems assume a homogeneous computing environment in which all processors have the same processing abilities and they are fully connected. Recent studies have been diverted to scheduling for heterogeneous computing environments in which the execution time of a task may vary among different processors, not all processors are directly connected, and the bandwidth of communication links connecting processors may also be different. In addition, some scheduling problems allow a task to be executed on multiple processors, while other problems restrict the execution of a task on only one processor [2].

The search for an optimal solution to the problem of multi-processor scheduling has been proven to be NP-hard except for some special cases [4]. Numerous approaches have been developed to solve the problem for heterogeneous computing environments. These approaches can be mainly classified into two categories: Deterministic approaches and non-deterministic approaches.

Deterministic approaches attempt to exploit the heuristics extracted from the nature of the problem in guiding the search for a solution. (e.g. HEFT [3] [11] and [3] [11] ,etc.). They are efficient algorithms as the search is narrowed down to a very small portion of the solution space. However, the performance of these algorithms is heavily dependent on the effectiveness of the heuristics. Therefore, they are not likely to produce consistent results on a wide range of problems.

Contrary to deterministic algorithms, non-deterministic algorithms incorporate a combinatoric process in the search for solutions. They typically require sufficient sampling of candidate solutions in the search space and have shown robust performance on a variety of scheduling problems. Since Genetic Algorithms [5], [15], [12], Simulated annealing [6], [7], and Tabu [10] search have been successfully applied to task scheduling, we propose a new method based on CRO to solve task scheduling problems.

CRO is a (variable) population-based general-purpose optimization metaheuristic [1], [14], [8], [9]. It mimics the interactions of molecules driving towards the minimum state of free energy (i.e. the most stable state). The manipulated agents are molecules, each of which has a molecular structure, potential energy (PE), kinetic energy (KE), and some other optional attributes. The molecular structure and PE corresponds to a solution of a given problem and its objective function value, respectively. KE represents the tolerance of a molecule getting a worse solution than the existing one, thus allowing CRO to escape from local optimum solutions. Imagine that we have a set of molecules in a closed container. They move and collide either on the walls of the container or with each other. Each collision results in one of the four types of elementary reactions, including on-wall ineffective collision, decomposition, inter-molecular ineffective collision, and synthesis. They have different characteristics and extent of change to the solutions. With the conservation of energy, the solutions change from high to low energy states and we output the molecular structure with the lowest found PE as the best solution.

978-0-7695-4701-5/12 $26.00 © 2012 IEEE
DOI 10.1109/ISPA.2012.11

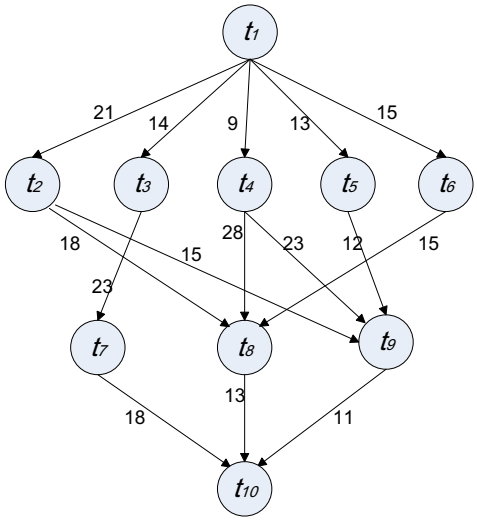

Figure 1. A DAG consists of 10 tasks.

The rest of this paper is organized as follows. We formulate the problem in Section II. In Section III, we describe the proposed CRO-based algorithm. Section IV gives the simulation results, compared with HEFT and CPOP evolutionary algorithms. We conclude this paper and suggest possible future work in Section V.

II. Problem Formulation

A. System Model

The target system used in this work consists of a set P of p heterogeneous processors/machines that are fully interconnected with the same communication links (i.e.,with the same bandwidths), but they have different processing abilities. In addition, each task can only be executed on one processor. The communication time between two dependent tasks should be taken into account if they are assigned to different processors. We also assume a static computing model in which the dependence relations and the execution times of tasks are known a prior and do not change over the course of scheduling and task execution.

B. Task Model

The scheduling problem is typically given by a group of dependent tasks along with a group of interconnected processors. The data dependency and execution precedence among tasks can be described with a directed acyclic graph (DAG). In general, a DAG can be defined as a four tuple $G = (V, E, C, W)$, which V is the set of vertex. it represents tasks partitioned from an application. An edge $e(i, j) \in E$ between task t_i and task t_j represents inter task communication. In other words, the output of task t_i has to be transmitted to task t_j in order for task t_j to start its execution. A task with no predecessors is called an entry task, t_{entry}, whereas an exit task, t_{exit}, is one that does not have any successors. The weight on an edge, denoted as $c(i, j) \in C$ represents the communication cost between two tasks, t_i and t_j. However, a communication cost is only required when two tasks are assigned to different processors. it means the communication cost can be ignored when tasks are assigned to the same processor. The weight on a task t_i is denoted as $w_i \in W$ represents the computation cost of the task. In addition, the actual start and finish times of a task t_i on a processor p_k, are denoted as AST(t_i, p_k) and AFT(t_i, p_k).

Fig. 1 shows an example DAG that contains ten tasks, t_1 to t_{10}. The arrows represent data dependencies among tasks. Two tasks are dependent if the execution of one task relies on the execution result of the other. The numbers represent the communication times needed to transfer data between two dependent tasks. Table I lists the execution times of each task on three processors, p_1, p_2 and p_3. Fig. 2 shows an execution schedule of tasks with a total execution time of 93 (also known as makespan).

C. Scheduling Model

The task scheduling problem in this study is the process of allocating a set T of t tasks to a set P of p processors without violating precedence constraints-aiming to minimize makespan as low as possible. Through the description above, we can define the makespan as $M = max\{AFT(t_{exit})\}$ after the scheduling of t tasks in a task graph G is completed. Meanwhile we can define a binary function $F(T, P) = max\{AFT(t_{exit})\}$, then we can get the objective function as follows:

$$f(t) = min(F(T_i, P_j)), \quad T_i \in T, P_j \in P \tag{1}$$

Table I
The Computation costs of ten tasks in Fig. 1 on three Processor, P_1, P_2, P_3.

t_i	P_1	P_2	P_3
t_1	11	13	9
t_2	10	15	11
t_3	9	12	14
t_4	11	16	10
t_5	15	11	19
t_6	12	9	5
t_7	10	14	13
t_8	11	15	10
t_9	14	16	9
t_{10}	13	19	18

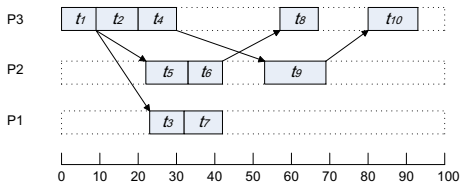

Figure 2. A schedule for Fig. 1, and the makespan is 93.

III. Algorithm Design

For a given task sequence $T_i \in T$, once the processor of every element in T_i is decided, then we can get the makespan. In this paper, we use an intelligent approach to get a set T of the task topologies in a DAG without violating precedence constraints. For each task topology sequence $T_i \in T$, CRO is implemented for searching a better processors allocation. Meanwhile, we record the smaller makespan with the corresponding task topology sequence and processors mapping.

A. An Intelligent Algorithm for Task Topology Sequences

The implement of the tasks in a DAG should be met by the predecessor constraints, that means a task is ready if it has no predecessor or all its predecessor tasks are already scheduled. While a task priority can be also expressed by its level. We simply think that a task owns a higher priority if it has more descendants, so the task level is used. The task level can be calculated recursively with the following equation:

$$IL(t_i) = \begin{cases} 0, & t_i = t_{entry} \\ max(IL(t_j)) + 1, & t_j \in pred(t_i) \end{cases} \quad (2)$$

A task that does not have any predecessor receives an level of zero, while any other task level is calculated by the maximum value of its predecessors' plus one. The level of a task is independent of the processors to which the task and all its predecessor tasks are assigned. Therefore, the level can be calculated before processors mapping. Table II shows the task levels in Fig. 1.

As shown above, the lower a task level is, the higher priorities it owns. So we can get the set T with some topologies of tasks ordered by $IL(t_i)$ up, and these topologies can be easily proved to meet the predecessor constraints. Table II is also a reasonable topology sequence. Note that the tasks with the same level, we consider them as the same priority and they can be exchanged randomly to get new topology structure. Fig. 3 is an example for ten tasks in Fig. 1 to get a new topology sequence.

B. CRO for Processors Mapping and Solution Representation

After getting the set T, for each $T_i \in T$, we use CRO-based algorithm to perform processors mapping, assign one of the available processors to the execution of each task. so we can model a one-dimensional vector $w = \{p_{w0}, p_{w1}, \ldots, p_{wn}\}$ as a solution, and the solution

Table II
The Task Level For Fig. 1.

t_i	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
IL	0	1	1	1	1	1	2	2	2	3

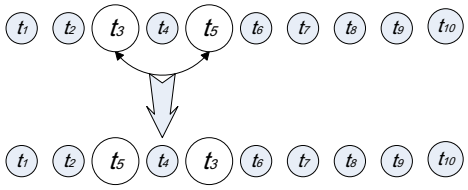

Figure 3. An example of getting new topology through task level.

Table III
The encoding of solution for Fig. 1. Each integer represents the processor that a task is assigned to.

3	3	1	3	2	2	1	3	2	3

of the CRO is encoded with a linear list of integers, with each integer representing the processor to which a task is assigned. Suppose there are t topologies, n tasks and m available processors. The value of each integer in the solution ranges from 1 to m, and there are n integers in each solution. The search space of a CRO, therefore, is $t \times m^n$. Table. III shows the corresponding CRO individual to the schedule in Fig. 2.

C. CRO Elementary Reaction Operators

We are going to describe the operators corresponding to the four elementary reactions of CRO. They all operate on the vector representation of solutions only. In the following, we denote a solution in vector form with w.

1) On-wall Ineffective Collision: In this elementary reaction, a molecule hits the wall of the container. There is little perturbation to the molecule, and thus, a mechanism with a small change to the solution (corresponding to the molecule) can be adopted. In this work, we get a new solution $w^{'}$ from an existing one w by changing two items of w randomly. Fig. 4 is a simple example,

2) Decomposition: One molecule w tries to split into two, $w_1^{'}$ and $w_2^{'}$. The resultant molecules have great perturbations from the original one. Therefore, $w_1^{'}$ and $w_2^{'}$ are quite different from w. To do this, we select a item p_{wi} from w randomly as the decomposition-point (DP). the left part of DP is used for the left of $w_1^{'}$ and we generate the right of $w_1^{'}$ randomly. At this point, $w_1^{'}$ can be obtained. Also,

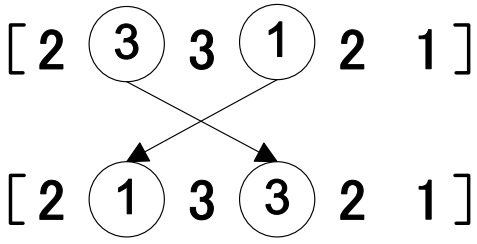

Figure 4. An example for on-wall ineffective collision reaction.

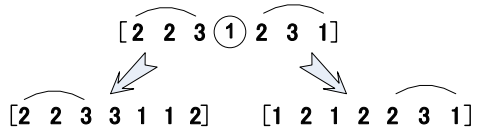

Figure 5. An example for one molecule decomposition.

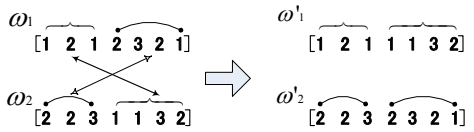

Figure 6. An example for two molecule synthesis.

at the same way, we generate the left of $w_2^{'}$ randomly, and combine the right part of DP as the right of $w_2^{'}$. This seems to place search "seeds" in two new and different regions of the solution space and thus increases the exploration ability of CRO. Fig. 5 is a simple example,

3) Inter-molecular Ineffective Collision: Two molecules, w_1 and w_2, collide with each other. Two new solutions, $w_1^{'}$ and $w_2^{'}$, are produced by adding small perturbations to w_1 and w_2, respectively. To do this, we apply the mechanism used for the on-wall ineffective collision to both w_1 and w_2 separately.

4) Synthesis: This tries to combine two molecule w_1 and w_2 into a new one $w^{'}$. $w^{'}$ should be quite different from w_1 and w_2 when compared with the ineffective collisions. To do this, we just take a simple way to avoid more computation. Firstly, a random integer is generated and is used as position in w_1 and w_2. Then combine left of w_1 with right of w_2 to get new molecule $w_1^{'}$, at the same time, w_1's right and w_2's left are connected to obtain new molecule $w_2^{'}$. At last, the solution with less makespan between $w_1^{'}$ and $w_2^{'}$ is selected as $w^{'}$. An example as Fig. 6,

D. Algorithm Outline

We basically follow the design framework described in [1] to develop a CRO-based algorithm to solve heterogeneous computing environments. The whole process consists of two elements: An intelligent approach to search some task topologies through task level, and implementing CRO metaheuristic method to search better processors mapping for each task topology. Firstly, we compute the level of each task, and order them by level up. Then two or more tasks with the same level are exchanged randomly to get new topology sequence while the loop number doesn't reach preset value, $TopoSize$. So there is $TopoSize$ task topologies in all. Here, $TopoSize$ is related to the problem scale, it means the value should get bigger with task number increased. Next is the core of the algorithm, implementing CRO searching of processors mapping for each topology. The detail of CRO is as following:

In the initialization, we create the initial set of molecules with size equal to $PopSize$ and their molecular structures are solutions in the way of one dimensional vector with every item generated randomly, then the objective function is evaluated and the corresponding values are the PE of the molecules. The initial KE of every molecule is set to the value of $InitialKE$. In each iteration, we decide whether a uni-molecular or an inter-molecular reaction is carried out in the iteration by comparing a random number $h \in [0, 1]$ with $MoleColl$. We select an appropriate subset of molecules to undergo an elementary reaction determined by the decomposition criterion or the synthesis criterion (depending on whether the elementary reaction is uni-molecular or inter-molecular). The iteration process continues until the stopping criterion is satisfied. We output the best-so-far solution in the final stage. For more information about CRO algorithm, interested readers may refer to [1]. The pseudo code of our method is as follows:

Algorithm 1 CRO for Scheduling.

Assign parameter values to $PopSize$, $KELossRate$, $MoleColl$, $InitialKE$, α, β, $TopoSize$, $minSolution$
For each task t_i do
 Calculate its level $IL(t_i)$
end For
Order tasks by task level up
set $Iteration$=0, $minSolution$=DBL_MAX
while $Iteration \leq TopoSize$ **do**
  Exchange two tasks with the same level randomly to get new task topology sequence T_i
  Implement CRO processors mapping for T_i, then get the result $croValue$
  //Check for any new minimum solution
  if $croValue \leq minSolution$ **then**
    set $minSolution$ = $croValue$, record T_i
  end if
  $Iteration$ plus one
end while
Output the overall minimum solution, its function value and corresponding topology sequence

Algorithm 2 Framework of CRO.

The detail about CRO, we follow the same framework to the flow of CRO mentioned in [1].

IV. Simulation and Results

In this section, we will compare the performance of CRO with HEFT and CPOP on random task graphs. According to our extensive comparative evaluation study, the results show that our algorithm performs better.

Table IV
THE CRO PARAMETER SETTING FOR VARYING CCR AND λ.

Parameter	Value
$PopSize$	25
$KELossRate$	0.4
$InitialKE$	1000
$MoleColl$	0.4
α	40
β	10

A. Comparison Metrics and Experimental Design

The communication to computation ratio (CCR) is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system [2].

Parallelism factor, λ [13]: The number of levels of the application DAG is generated randomly, using a uniform distribution with a mean value of $\frac{\sqrt{n}}{\lambda}$ (n equal the number of tasks), and then rounding it up to the nearest integer. The width is generated using a uniform distribution with a mean value of $\lambda\sqrt{n}$ and then rounding it up to the nearest integer [3]. A low λ leads to a DAG with a low parallelism degree.

In our experiment, we test 11 groups of task graphs with different CCRs and parallel factors respectively. Both the communication to computation ratio and parallel factor have baseline settings of 1.0 and range between 0.4 and 2.4. To have fair comparisons of performance over various optimization strategies, for each group, we generate 30 task graphs randomly and some necessary information, such as the implementing time of each task on processors and the communication time between two tasks, then we calculate its average value as the reference. Each graph is based on 4 processors and 20 tasks. For each run, we calculate the speedup of the solution using the following equation:

$$Speedup = \frac{serial\_execution\_time}{makespan} \tag{3}$$

which serial execution time is the sum of the average computation times of all tasks. We use serial execution time to approximately calculate the makespan of a schedule if all tasks are serially assigned to the same processor. The bigger the speedup is, the more effective the distribution of task execution on parallel processors.

For each task graph, we run CRO 30 times, and calculate both the average speedup of solutions and the speedup of the best solution found in 30 runs.

The parameter values of CRO are given as Table. IV:

At the same time, we evaluate the performance of the other references on the same 11 groups of task graphs and calculate the average speedup of solutions for each group. HEFT is a list scheduling algorithm and the priority of tasks is based on their upward ranks. As a deterministic algorithm, HEFT is run only once for each task graph. So is CPOP. Finally, All the experiment are performed on a 2.81GHz AMD dual-core Processor with 4.00GB of RAM.

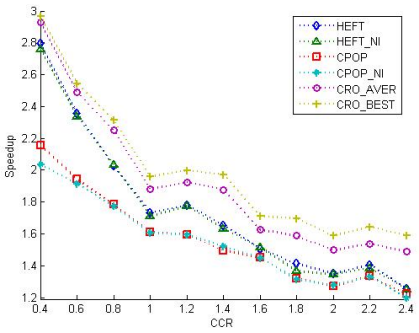

Figure 7. The comparisons of performance between CRO and HEFT, HEFT_NI(HEFT with no insertion), CPOP, CPOP_NI(CPOP with no insertion) on task graphs with varying CCRs.

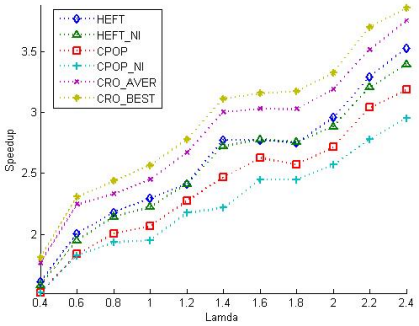

Figure 8. The comparisons of performance between CRO and HEFT, HEFT_NI, CPOP, CPOP_NI on task graphs with varying λs.

B. Experimental Results and Analysis

Fig. 7 shows the comparison between the CRO-based algorithm and the other references(HEFT, HEFT_NI. CPOP, CPOP_NI) on task graphs with varying CCRs. The parallel factor λ is fixed at 0.5. For CRO runs, we show the average speedup of both the best solutions and the average solutions in each test case. The results indicate that the speedup of schedules decreases quickly as the CCR increases. The CRO performs consistently better than the other algorithms in all CCR cases. The gaps on the performance of these algorithms are more noticeable in test cases with higher CCRs (e.g., with a ratio of 2.4). To schedule a task graph with a CCR, proper assignment of dependent tasks on processors is essential to avoid or reduce high communication costs. The use of the CRO for processor mapping enables the algorithm to search for a larger solution space than the others, so it is more likely to find better mapping for tasks. Fig. 7 also indicates that a better result can be found if we run the algorithm sufficient number of times.

Table V
INFORMATION FOR VARIED TASK SIZE, THE PROCESSOR SIZE IS FIXED AT 8

Task size	Computation	Topology Number	Rate(K)
20	8000	15	1.0408e-013
25	13000	20	6.8821e-018
30	18000	20	2.9081e-022
35	20000	25	1.2326e-026
40	35000	35	9.2159e-031
45	60000	60	8.2652e-035
50	90000	60	3.7835e-039
55	110000	80	1.8816e-044

Fig. 8 shows the comparison between the CRO algorithm and the others on task graphs with a varying λ. The CCR is fixed at 1.0. For CRO runs, we also show the average speedup of both the best solutions and the average solutions in each test case. The results indicate that the speedup of schedules increases quickly as λ increases. The CRO algorithm performs consistently better than the other algorithms in all test cases. To schedule a task graph with a high λ , proper assignment of dependent tasks on different processors is essential to avoid or reduce high waiting costs. Again, running the algorithm multiple times allows us to find better solutions than a single run.

In addition, we also perform the experiments to evaluate the effectiveness when problem scale increases. For comparison, we introduce a new comparing item. We define the ratio of other algorithm and CRO makespan as argument INC:

$$INC = \frac{makespan\ of\ others}{makespan\ of\ CRO} \tag{4}$$

which others means HEFT, HEFT_NI, CPOP and CPOP_NI. INC indicates that CRO is how much better than the other algorithms. If the INC is bigger than one, it illustrates the makespan of CRO is smaller, and CRO is better. At the same time, the smaller the makespan of CRO is, the bigger INC is, and CRO is more suitable.

The CRO-based algorithm outperforms the other algorithms in all test cases for varied problem scale. Fig. 9 give the results respectively. The two parameters CCR and λ are fixed at value 1.0 and 1.0 respectively. For CRO run, we also show the average INC of both the best solutions and the average solutions in each test case. The results indicate the INC of scheduling is greater than value one in each task size. CRO performs consistently better than all the other algorithms when the problem scale getting bigger. Further, we can find that the curves are incremental when the task number increases, that means CRO is more superior for problems with large scale.

Especially, we increase the number of function evaluations with problem scale getting larger. while in fact, increasing the amount of CRO evaluations by specified rate is not necessary. That means the mount of evaluations doesn't need to increase same rate when the solution space increases. An example is shown from Table V, when the task number is 35, the solution space is 8^{35} , and the mount of CRO computation is 500000 (20000*25), so the computation rate $K_{35} = 1.2326 \times 10^{-26}(500000/8^{35})$. However, when the task number is 40, the rate is $K_{40} = 9.2159 \times 10^{-31}$, much less than K_{35}. Also we can see that the rate decrease quickly when the problem scale increase, that futher proves our method is more suitable for large scale problem. Again, running CRO more times allows us to find better solutions.

V. CONCLUSION

We design a CRO-based algorithm for scheduling tasks on heterogeneous processors. This algorithm incorporates a CRO search to map processors to tasks while using an intelligent approach to assign the execution orders of tasks by task level. It increase search space effectively. So we can usually obtain better solution without much computational cost. CRO is a chemical-reaction-inspired meta-heuristic for general optimization. With the framework of CRO, we develop several operators so as to make CRO capable of generating good solutions which satisfy the problem requirements and constraints of task scheduling. The experiments show that this algorithm outperforms HEFT and CPOP, a widely used non-deterministic algorithm for heterogeneous computing systems, with a higher speed up and lower makespan on task execution. The advantage of this algorithm is more noticeable if proper assignment of tasks on processors is critical to locate high quality solutions. In the future, we will combine some classic heuristic method to search better task topologies(i.e. ANT, GA), at the same time, improve the CRO elementary reaction operators. This modification may result in better makespan and may further improve the quality of solutions.

ACKNOWLEDGMENT

We would like to thank the anonymous referees for their valuable comments on improving the quality of the paper.

REFERENCES

[1] A. Lam and V. Li. Chemical-reaction-inspired metaheuristic for optimization. *Evolutionary Computation, IEEE Transactions on*, 14(3):381 –399, june 2010.

[2] H. Yu. A hybrid ga-based scheduling algorithm for heterogeneous computing environments. In *Computational Intelligence in Scheduling, 2007. SCIS '07. IEEE Symposium on*, pages 87 –92, april 2007.

[3] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, IEEE Transactions on*, 13(3):260 –274, mar 2002.

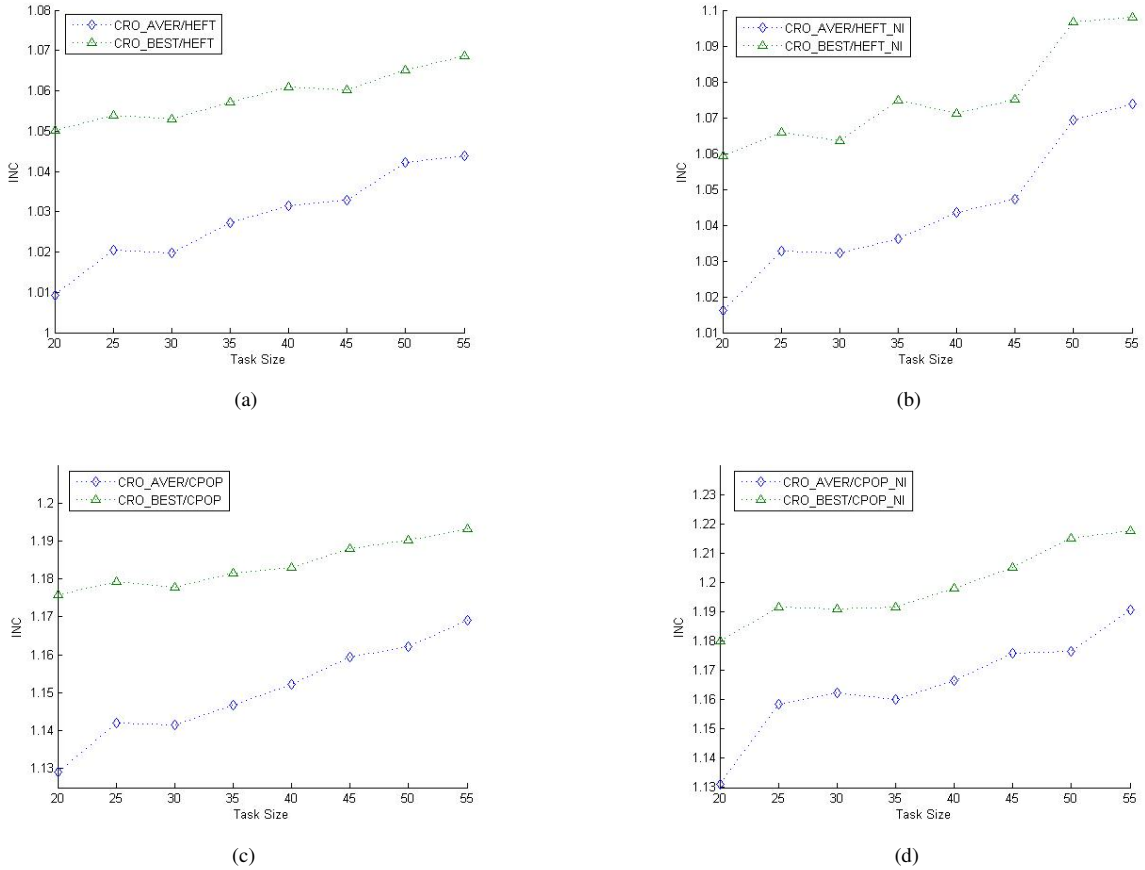

Figure 9. INCs for different task number, (a) INC for HEFT. (b) INC for HEFT_NI. (c) INC for CPOP. (d) INC for CPOP_NI.

[4] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[5] E. Hou, N. Ansari, and H. Ren. A genetic algorithm for multiprocessor scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 5(2):113 –120, feb 1994.

[6] K. Hwang and J. Xu. Mapping partitioned program modules onto multicomputer nodes using simulated annealing. In *ICPP (2)'90*, pages 292–293, 1990.

[7] A. Nanda, D. DeGroot, and D. Stenger. Scheduling directed task graphs on multiprocessors using simulated annealing. In *Distributed Computing Systems, 1992., Proceedings of the 12th International Conference on*, pages 20 –27, jun 1992.

[8] A. Lam and V. Li. Chemical reaction optimization for cognitive radio spectrum allocation. In *GLOBECOM 2010, 2010 IEEE Global Telecommunications Conference*, pages 1 –5, dec. 2010.

[9] J. Sun, Y. Wang, J. Li, and K. Gao. Hybrid algorithm based on chemical reaction optimization and lin-kernighan local search for the traveling salesman problem. In *Natural Computation (ICNC), 2011 Seventh International Conference on*, volume 3, pages 1518 –1521, july 2011.

[10] S. C. S. Porto and C. C. Ribeiro. A tabu search approach to task scheduling on heterogeneous processors under precedence constraints. *International Journal Of High Speed Computing*, 7(1):45 –71, 1995.

[11] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Heterogeneous Computing Workshop, 1999. (HCW '99) Proceedings. Eighth*, pages 3 –14, 1999.

[12] M. Daoud and N. Kharma. An efficient genetic algorithm for task scheduling in heterogeneous distributed computing systems. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 3258 –3265, 0-0 2006.

[13] X. Tang, K. Li, G. Liao, and R. Li. List scheduling with duplication for heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 70(4):323 – 329, 2010.

[14] J. Xu, A. Lam, and V. Li. Chemical reaction optimization for task scheduling in grid computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(10):1624 –1631, oct. 2011.

[15] T. Tsuchiya, T. Osada, and T. Kikuno. Genetic-based multiprocessor scheduling using task duplication. *Microprocessors and Microsystems*, 22:197 –207, 1988.