Title	Chemical reaction optimization for task scheduling in grid computing
Author(s)	Xu, J; Lam, AYS; Li, VOK
Citation	IEEE Transactions On Parallel And Distributed Systems, 2011, v. 22 n. 10, p. 1624-1631
Issued Date	2011
URL	http://hdl.handle.net/10722/155651
Rights	©2011 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Chemical Reaction Optimization for Task Scheduling in Grid Computing

Jin Xu, *Student Member*, *IEEE*, Albert Y.S. Lam, *Member*, *IEEE*, and Victor O.K. Li, *Fellow*, *IEEE*

Abstract—Grid computing solves high performance and high-throughput computing problems through sharing resources ranging from personal computers to supercomputers distributed around the world. One of the major problems is task scheduling, i.e., allocating tasks to resources. In addition to *Makespan* and *Flowtime*, we also take reliability of resources into account, and task scheduling is formulated as an optimization problem with three objectives. This is an NP-hard problem, and thus, metaheuristic approaches are employed to find the optimal solutions. In this paper, several versions of the Chemical Reaction Optimization (CRO) algorithm are proposed for the grid scheduling problem. CRO is a population-based metaheuristic inspired by the interactions between molecules in a chemical reaction. We compare these CRO methods with four other acknowledged metaheuristics on a wide range of instances. Simulation results show that the CRO methods generally perform better than existing methods and performance improvement is especially significant in large-scale applications.

Index Terms—Grid computing, task scheduling, multicriteria scheduling, chemical reaction optimization.

1 Introduction

With the advancement of high-speed networks, grid computing (also known as computational grid) is proposed and considered as the foundation of the next-generation Internet [1]. The goal of a grid system is to solve large-scale and high-performance computing problems through sharing many geographically distributed computing resources belonging to different administrative domains. Grid technology has a wide range of applications in many fields of science and engineering, e.g., astronomy, meteorology, bioinformatics, transportation, financial modeling, drug discovery, high energy physics, data mining, and image manipulation [2], [3], [4].

A grid usually consists of five parts: clients, the Global and Local Grid Resource Brokers (GGRB and LGRB), Grid Information Server (GIS), and resource nodes [5] (see Fig. 1). Clients register their requests of processing their computational tasks at GGRB. Resource nodes register their donated resource at LGRB and process clients' tasks according to the instructions from LGRB. In practice, client and resource node can be the same computer. GIS collects the resource information from all LGRBs, and transfers it to GGRB. GGRB is responsible for scheduling. It possesses all necessary information about the tasks and resources and acts like a database of the grid [5]. A grid operates in time intervals. At the beginning of each interval, GGRB performs scheduling. During an interval, the registered resources process the tasks. Scheduling tasks for various heterogeneous nodes is an essential part of a grid system. Efficient scheduling (i.e., performing a scheduling scheme which minimizes the computational overheads and uses the resources effectively) is one of the most important problems in grid computing.

The main challenge of task scheduling in grids (different from the scheduling problems of other computer/communication system, e.g., [6], [7]) is its highly dynamic environment, where the resource nodes have their own access policies, availability, and so on. In other words, some nodes are completely dedicated to the grid; some provide service only when they are idle; and the rest falls between the two extremes [8]. Meanwhile, the resources can be of great heterogeneity, ranging from desktop PCs to supercomputers.

Task scheduling in grids (also named as grid scheduling) is an *NP*-hard optimization problem [9], and many metaheuristic algorithms have been proposed to solve it. Some of these algorithms are nature-inspired, e.g., Simulated Annealing (SA) [10], Genetic Algorithm (GA) [11], Ant Colony Optimization (ACO) [12], Particle Swarm Optimization (PSO) [13], etc. There are also non-nature-inspired metaheuristics, such as Tabu Search [14] and Threshold Accepting (TA) [15]. Chemical Reaction Optimization (CRO) is recently proposed [16], and it mimics the interactions of molecules in chemical reactions. CRO is a population-based metaheuristic, and it has already shown its power in solving problems like Quadratic Assignment Problem (QAP), Resource-Constrained Project Scheduling Problem (RCPSP), and Channel Assignment Problem (CAP) [16]. In this paper, we develop several versions of CRO to solve the grid scheduling problem, and we compare the performance of CROs with that of SA, GA, PSO, and TA.

The rest of the paper is organized as follows: related work is described in Section 2. In Section 3, we formulate the grid scheduling problem. Section 4 describes the

- *J. Xu is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Room 615, Chow Yei Ching Bldg., Pokfulam Road, Hong Kong, China. E-mail: xujin@eee.hku.hk.*
- *A.Y.S. Lam is with the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, Berkeley, CA 94720. E-mail: ayslam@eecs.berkeley.edu.*
- *V.O.K. Li is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Room 601D, Chow Yei Ching Bldg., Pokfulam Road, Hong Kong, China, and the Department of Computer Engineering, King Saud University, Saudi Arabia. E-mail: vli@eee.hku.hk.*

Manuscript received 19 Apr. 2010; revised 16 Sept. 2010; accepted 12 Oct. 2010; published online 18 Jan. 2011.
Recommended for acceptance by A. Zomaya.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2010-04-0228.
Digital Object Identifier no. 10.1109/TPDS.2011.35.

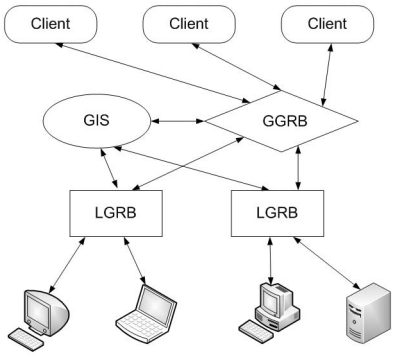

Fig. 1. The architecture of the Grid System.

framework of CRO, as well as two different solution representations with their operator mechanisms. Simulation results and analysis are given in Section 5. Finally, we present concluding remarks and suggestions for future work in Section 6.

2 Related Work

In this section, we will first review the previous work on the grid scheduling problem. Then brief description of the metaheuristic approaches considered in this paper will be given.

2.1 Grid Scheduling

Recently grid scheduling has attracted much attention. Many grid computing infrastructure have emerged, such as the e-Minerals Grid [17] in the United Kingdom, and the EGEE project [18] funded by the European Commission. At the same time, a number of scheduling systems have been developed, such as Condor-G [19], AppLes [20], and CHIMERA [21]. In [22], scheduling in grids can be classified into different types. From the system's perspective, scheduling can be centralized, hierarchical or decentralized. Also, the system can process the tasks in batches or handle a task immediately once submitted. Other policies like Deferred Assignment Scheduling (DAS) [23], [24] are adopted by some researchers as well. Moreover, tasks can also be independent or have precedence relationship. In most cases, applications with intensive use of CPUs may be split into independent parts, which can be considered as independent tasks. For those applications involving extensive data sharing among resources, we need to account for the precedence relationship between tasks, also known as the Grid Workflow. In this paper, we consider problems in batch-processed mode with independent tasks.

Heuristic and metaheuristic methods have been applied to solve different types of grid scheduling problems. In [25], the authors tried to map independent tasks onto heterogeneous distributed computing systems, and they compared eleven approaches, including heuristic methods (such as Min-Min, Max-Min, Duplex, etc.) and metaheuristic (like GA, SA, and GSA which is a hybridization of the former two.). The outcome is that GA consistently gave the best results when the benchmarking and parameter values were set according to some rules. Fidanova [26] used SA for grid scheduling, optimizing the makespan, and showed that it performed better than ACO. A novel PSO approach was proposed in [27], and it evaluated makespan and flowtime simultaneously. Threshold Accepting Algorithm [28], which was relatively simple and easy to implement, converged faster than SA and GA-LS (GA combined with a local search) when solving the workflow problem.

2.2 Metaheuristics

Since we shall compare CRO with GA, SA, PSO, and TA, we describe them below.

GA, a particular class of evolutionary algorithms, was created by Holland [11]. It follows the natural selection law, which means only the fittest can survive. GA is a population-based metaheuristic and produces the next generation with the techniques inspired by evolutionary biology, such as inheritance, mutation, crossover, and selection. Consider a solution as an organism. The better the quality of a solution, the higher the probability it will survive. Through crossover (also called recombination) and mutation, GA can escape from the local optimal to search for the global optimal.

SA, originated from annealing in metallurgy, imitates its process of decreasing the defects of a substance according to a cooling procedure. For each iteration, SA generates a new solution randomly in the neighborhood of the present solution, and it will be accepted if it is better, or accepted with a probability controlled by a temperature parameter. As the temperature gradually drops, the ability to jump out of local optima decreases and finally moves to the global optimum.

PSO, introduced by Kennedy and Eberhart [13], is a population-based direct search method. It is modeled on swarm intelligence, like bird flocking and fish schooling. Particles, similar to individuals, not only remember their own local best positions (solutions), but also communicate with each other and record the globally best position. These two parameters are used in the velocity function to balance exploration and exploitation.

TA, presented by Dueck and Scheuer [15], is similar to SA but with a different acceptance rule. Every new solution would be accepted as long as the difference is smaller than a threshold. Like the temperature in SA, the threshold also decreases gradually with the iterations. TA is simple and does not require probability calculations and random decision making, yielding computational results superior to SA in many cases [15].

3 Problem Formulation

We formulate the problem based on the "Expected Time to Compute" (ETC) Model [29]. In a particular time interval, n independent tasks $J_1, J_2, \ldots, J_n$ (expressed in millions of instructions) are submitted to GGRB for scheduling, and at the same time, GIS locates m (usually $n \gg m$) grid nodes $R_1, R_2, \ldots, R_m$, donating resources. As the nature of a grid node's resources (e.g., CPU speed, memory, storage, etc.) varies greatly, we simply measure the combined processing power of a grid node, in terms of "millions of instructions per second." We use the terms "node" and "resource" interchangeably.

Based on the specifications of the resources and tasks, GGRB computes an $n \times m$ matrix *ETC*, where entity ETC_{ij} represents the expected time for resource j to process task i.

If resource j cannot handle task i, ETC_{ij} is set to infinity. In addition, the grid nodes may have unfinished workload accumulated from the previous intervals. We use $W_1, W_2, \ldots, W_m$ to denote the time required for processing the remaining tasks for the m resources. We also assume:

1. A task can only be executed on one grid node in each interval.
2. No preemptive process is allowed within tasks or resources.
3. When a node fails, its tasks will be reallocated to other node(s) in the next interval.
4. When a node processes its tasks, there is no priority distinctions between the tasks assigned in the previous intervals and those assigned in the current interval.
5. A node cannot remain idle when tasks have been assigned to it.

In this paper, we employ both the permutation- and vector-based representations for the schedules, i.e., the solutions. For the permutation-based representation, two vectors are usually employed. The first one describes the number of tasks allocated to each resource. So the position of the element is the resource's ID, and the sum of values of all the elements is equal to n, where n is the total number of tasks. The second vector is a permutation of $\{1, 2, \ldots, n\}$. For the vector-based representation, the number of elements in the vector is equal to the number of tasks. The position of the element in the vector denotes the task's ID. All the elements are integers in the range of $[1, m]$, where m is the total number of resources. The element's value indicates which resource is chosen to process the corresponding task, and these values can be repetitive. For example, let ω be a vector denoting a solution, and suppose six tasks are scheduled with four resources, and the first and third tasks are allocated to Resource 1, the second task runs on Resource 3, the fourth task on Resource 4, and the fifth and sixth on Resource 2. When the solution employs the permutation-based representation, let the first solution vector ω^1 be $[2, 2, 1]$. It means that both Resources 1 and 2 have two tasks and Resources 3 and 4 each own one task. Let the second vector ω^2 be $[\underline{1, 3}, \underline{5, 6}, \underline{2}, \underline{4}]$ and it means that first group of elements underlined are given to Resource 1, the second group allocated to Resource 2, and so on. When it comes to the vector-based one, the same solution can be represented by $\omega = [1, 3, 1, 4, 2, 2]$.

Makespan and *Flowtime* are two commonly used metrics for measuring the quality of a schedule in grid computing [22], [25], [27]. *Makespan* is the completion time of the last finished task, while *Flowtime* is the total time consumed by all tasks. We define C_J as the time required for Node j to complete all its assigned tasks. Mathematically, we have

$$C_j = \sum_{\{i|\omega_i = j\}} ETC_{ij} + W_j, \tag{1}$$

$$Makespan = max\{C_j\}, \tag{2}$$

and

$$Flowtime = \sum_{j=1}^{m} C_j, \tag{3}$$

where $j = 1, 2, \ldots, m$, and $\{i|\omega_i = J\}$ represents the tasks assigned to Node j.

As mentioned before, system performance is vulnerable to the volatile donated grid resources. Grid nodes may become unavailable due to disconnection of power or communication, unwillingness or policies of the resource owners, etc., [8] describes a failure predictor, which utilizes historical data to forecast the availability of grid nodes. Thus, besides *Makespan* and *Flowtime*, it is important to take the reliability into account. We define P_J as the reliability probability of resource j, where $j \in 1, 2, \ldots, m$. This allows GGRB to select a resource with higher P_J when more than one resource is available. We evaluate the impact of reliability with *T_aborted*, where

$$T\_aborted = \sum_{j=1}^{m} \sum_{\{i|\omega_i = j\}} [(1 - P_j) \times ETC_{ij}]. \tag{4}$$

T_aborted, in fact, represents the total wasted time caused by the aborted tasks in terms of reliability. The higher the P_J, the smaller *T_aborted* is.

In this paper, we try to minimize *Makespan*, *Flowtime*, and *T_aborted* simultaneously, and thus, the grid scheduling problem is a multiobjective optimization problem. However, as discussed in [30], minimizing *Makespan* requires the most demanding tasks to be assigned to the fastest resource, at the expense of increasing the finish time of other tasks, and hence increasing *Flowtime*. On the other hand, optimizing *Flowtime* requires all tasks to finish quickly on the average, at the expense of having the most demanding tasks taking a longer completion time, thus increasing *Makespan*. Moreover, to minimize *T_aborted*, the tasks tend to be allocated to the resources with high reliability. Therefore, these three criteria lead to contradictory decisions. Let f denote the fitness function. We formulate the grid scheduling problem as

$$\begin{aligned} min \qquad f(\omega) = a \times Makespan + b \times (Flowtime/m) \\ + c \times (T\_aborted/m), \end{aligned} \tag{5}$$

where $a + b + c = 1$ and $0 \leq a, b, c \leq 1$.

The weights put on the three objectives will be further discussed in [31]. *Flowtime* and *T_aborted* have higher order of magnitude over *Makespan*, and thus, we normalize them by m.

4 Algorithm Design

4.1 Modified Framework of CRO

The basic concept of CRO and its main executing process are illustrated in [31]. The canonical framework of CRO [16] has been employed to solve the grid scheduling problem in [32]. Based on that, we develop a modified framework of CRO for the problem and its flowchart is shown in [31, Fig. S3].

In the modified framework, only one molecule is generated in the initialization stage. Besides, before reaction happens, we put some energy in the container which means that the central energy buffer is set to a certain positive value, instead of zero in the buffer of the canonical framework. The initial quantity of energy in the central buffer will influence the number of molecules in the system.

Then we divide the iteration stage into two parts. In the first part, only unimolecular collisions (on-wall ineffective collision and decomposition)are allowed. As the decomposition is executed frequently, more and more molecules will be generated. The purpose is to explore the solution space as much as possible. After the predefined number of iterations, the second part starts. This part is similar to the whole iteration stage of the canonical framework, except that it excludes the decomposition reaction. This part aims to enhance the convergence of solutions.

4.2 Operators

As mentioned before, both permutation-based and vector-based representations of solutions are used in CRO comparisons. However, in the implementation, operators are totally different for these two representations when they are employed to generate solutions. In the following two sections, we will describe the operators for these two representations.

4.2.1 Permutation-Based Representation

As described in Section 3, in the permutation-based form, usually we need to use two vectors to represent the solutions. In this paper, we transform these two vectors into one. We extend the vector with length n to length $n+m-1$, where the added $m-1$ elements are the delimiters to distinguish which tasks are assigned to the nodes. In other words, the tasks between the adjacent delimiters are distributed to the specified resource. For simplicity, we use $n+1,\ n+2,\ldots,n+m-1$ as the delimiters. For example, originally, we use $[1,3,5,6,2,4]$ and $[2,2,1,1]$ to denote one solution; now, it is instead expressed by $[1,3,\underline{7},5,6,\underline{8},2,\underline{9},4]$, where 7, 8, 9 are the delimiters. The elements before 7, i.e., Tasks 1 and 3 are distributed to Resource 1; Tasks 5 and 6 which are between the delimiters 7 and 8 are allocated to Resource 2; Task 2 between 8 and 9 is assigned to Resource 3, and Task 4 is given to Resource 4. If there is no task between any two neighboring delimiters, the relative resource will be idle. With this transformation, only one vector is needed to represent the solutions of the grid scheduling problem. Following are the operators used in CRO:

1. Insertion operator (for on-wall ineffective collision). Two numbers are randomly selected from the vector, then the second number is inserted before the first number. As shown in the example below, the numbers 7 and 2 are selected, and we put the number 2 ahead of 7 in the insertion operator

$$[1,3,\mathbf{7},5,6,8,\mathbf{2},9,4] \rightarrow [1,3,\mathbf{2},\mathbf{7},5,6,8,9,4].$$

2. Position-based operator (for synthesis). It involves two molecules and combines two solutions ω_1 and ω_2 into a new one ω'. Each number in ω_1 has 50 percent probability to be chosen and passed to ω' at the same position. Then, the numbers in the unfilled positions of ω' are picked from ω_2. For example, let ω_1 be [1, 3, 7, 5, 6, 8, 2, 9, 4]. We randomly choose the numbers 7, 6, and 2, then pass them to the same positions in ω'. The remaining numbers, 1, 3, 4, 5, 8 and 9 in ω' are put in the vacant positions according to their orders in ω_2, which is 4, 5, 3, 1, 8, 9. Thus, we have

Strings before synthesis

$$\omega_1 : [1,3,\mathbf{7},5,\mathbf{6},8,\mathbf{2},9,4]$$
$$\omega_2 : [7,\underline{4},\underline{5},\underline{3},\underline{1},2,\underline{8},6,\underline{9}]$$

String after synthesis

$$\omega' : [4,5,7,3,6,1,2,8,9].$$

3. Two-exchange neighborhood operator (for the intermolecular ineffective collision). We select two numbers randomly from the solution, then exchange their positions. In the following example, the positions of 7 and 2 are exchanged

$$[1,3,\mathbf{7},5,6,8,\mathbf{2},9,4] \rightarrow [1,3,\mathbf{2},5,6,8,\mathbf{7},9,4].$$

4. For the decomposition reaction in the CRO, we randomly generate two totally new solutions.

4.2.2 Vector-Based Representation

We also introduce four operators for this representation as follows:

1. One-resource change operator (for on-wall ineffective collision). One task is randomly selected and then assigned to another resource. As shown below, we assign the fifth task from Resource 2 to 3

$$[1,3,1,4,\mathbf{2},2] \rightarrow [1,3,1,4,\mathbf{3},2].$$

2. Pairwise exchange operator (for on-wall ineffective collision and intermolecular ineffective collision). We randomly pick two tasks and exchange their respective assigned resources. In the following example, originally the second task and the fifth task are assigned with Resources 3 and 2, respectively. After the change, the second task is now assigned to Resource 2, and the fifth to Resource 3

$$[1,\mathbf{3},1,4,\mathbf{2},2] \rightarrow [1,\mathbf{2},1,4,\mathbf{3},2].$$

3. One-position exchange operator (for synthesis). We combine two solutions ω_1 and ω_2 into a new one ω'. An integer value k is randomly generated in the range of $[1,n]$, where n is the total number of tasks. Then ω' is generated by picking the first k values from ω_1 and the rest of the $(n-k)$ values from ω_2. Below, ω' ($[1,3,2,3,4,1]$) is formed by combining the first two numbers (1 and 3) from ω_1 and the last four (2, 3, 4, and 1) from ω_2. Thus, we have

Strings before synthesis

$$\omega_1 : [\underline{1,3,}1,4,2,2]$$
$$\omega_2 : [3,2,\underline{2,3,4,1}]$$

String after synthesis

$$\omega' : [1,3,2,3,4,1].$$

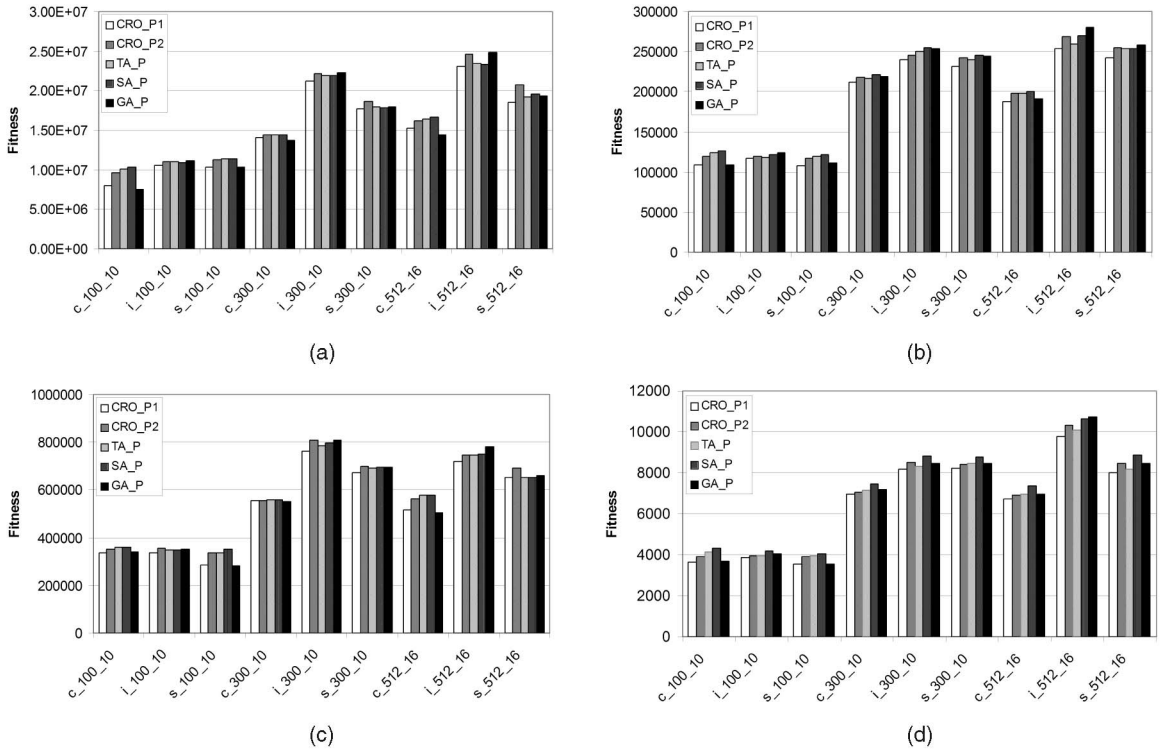

Fig. 2. Comparison of fitness values for the 36 simulation cases with configurations of (100,10), (300,10), and (512,16) in permutation-based representations. (a) High task heterogeneity and high resource heterogeneity. (b) High task heterogeneity and low resource heterogeneity. (c) Low task heterogeneity and high resource heterogeneity. (d) Low task heterogeneity and low resource heterogeneity.

4. Half-random operator (for decomposition). We produce two new solutions ω'_1 and ω'_2 from one solution ω'. We divide the values of the solution into two sets, depending on whether they are in the odd- or even-numbered positions. The odd-position set of ω is assigned to the odd positions of ω'_1, while the even-position set of ω is assigned to the even position of ω'_2. The unassigned positions in these two new solutions are filled with random numbers. As the example shown below, the numbers 1, 1, and 2 are given to ω'_1 in the same odd places, while 3, 4, and 2 are allocated to ω'_2 in the even places. Then we generate random numbers (2, 3, and 1) to fill in the vacant positions (boxed) of these two new solutions. Thus, we have

   Strings before decomposition

$$\omega : [\underline{1}, \mathbf{3}, \underline{1}, \mathbf{4}, \underline{2}, \mathbf{2}]$$

   String after decomposition

$$\omega'_1 : [\underline{1}, \boxed{2}, \underline{1}, \boxed{3}, \underline{2}, \boxed{1}]$$

$$\omega'_2 : [\boxed{2}, \mathbf{3}, \boxed{3}, \mathbf{4}, \boxed{1}, \mathbf{2}]$$

4.3 Summary

Up till now, we have two frameworks of the CRO and two representations for the solutions. Thus totally, we have four versions of CRO, named as CRO_P1, CRO_P2, CRO_V1, and CRO_V2, respectively ("P" means permutation-based representation, while "V" is vector-based one; "1" stands for the canonical framework of CRO, while "2" is the modified one). We will compare the performance of these CRO versions with other algorithms below.

5 Simulation Results

In this section, we first show simulation results with permutation-based and vector-based representations, respectively. Second, the outcomes of employing these two different representations are compared. Details on the simulation environment, parameter analysis for CRO and weights selection for the fitness functions are given in [31].

5.1 Permutation-Based Results

In this section, we use the permutation type as the solution representations. Operator mechanisms adapted to this type are also employed. As described in Section 2, PSO evolves according to the velocity and position of the particles. In essence, PSO is updated (generating new solutions) automatically, and does not utilize any operator mechanism to get new solution. The only thing that influences the performance of this algorithm is the parameter settings. Hence, it will perform the same in both permutation- and vector-based representations. So we compare PSO in the vector-based section, and this part includes TA_P, SA_P, GA_P, CRO_P1, and CRO_P2 only.

Fig. 2 [31, Fig. S4] show the performance of the five algorithms in terms of fitness value in a wide range of cases. We adopt the notation "x_y_z" for the test cases, where "x" can be consistent "c," inconsistent "i," or semiconsistent "s," while "y" and "z" indicate the number of jobs and resources, respectively. For example, "c_100_10" refers to the case which is a consistent system with 100 jobs and 10 resources.

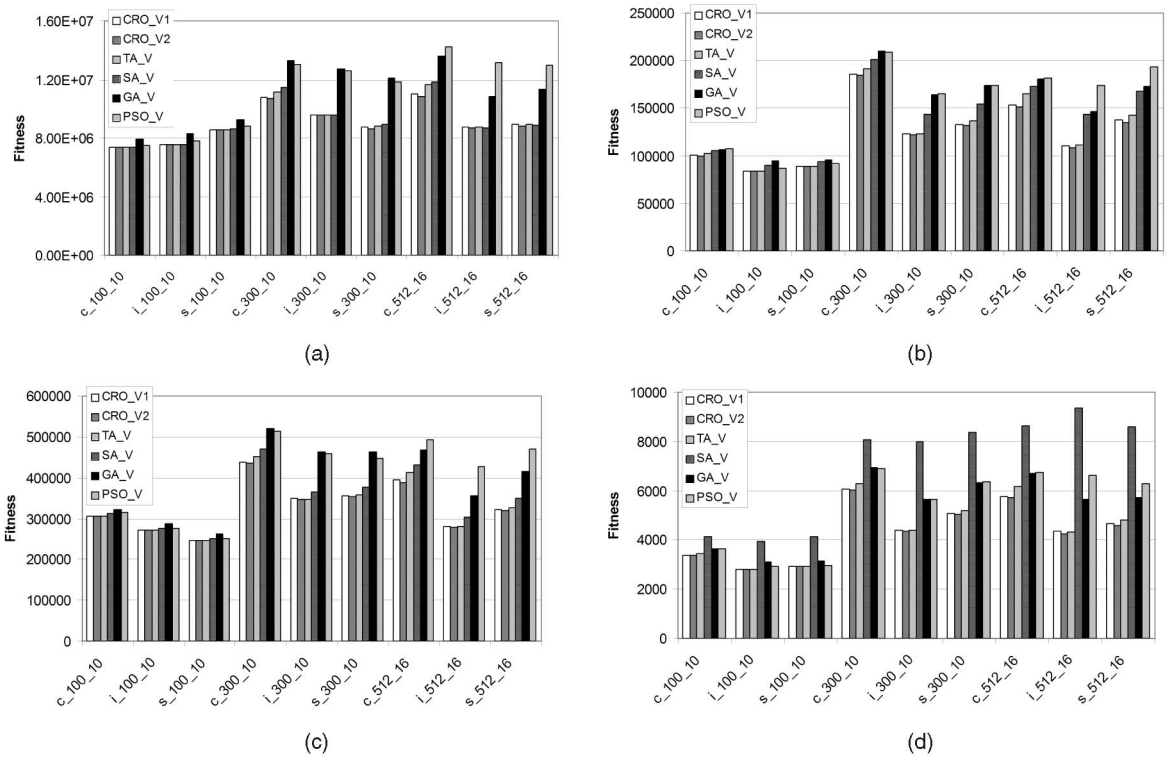

Fig. 3. Comparison of fitness values for the 36 simulation cases with configurations of (100,10), (300,10) and (512,16) in vector-based representations. (a) High task heterogeneity and high resource heterogeneity. (b) High task heterogeneity and low resource heterogeneity. (c) Low task heterogeneity and high resource heterogeneity. (d) Low task heterogeneity and low resource heterogeneity.

We conclude that: 1) Generally, CRO_P1 performs best in the inconsistent system, while GA_P does best in the consistent and semiconsistent systems. 2) CRO_P2, TA_P, and SA_P obtain similar results. More specifically, among these three algorithms, CRO_P2 gets the best results in more than half of the cases. This reveals that it is better than TA_P and SA_P. 3) From the consistent system to the inconsistent one, the performance of GA_P deteriorates. In the inconsistent cases, GA_P is the worst. Therefore, in terms of effectiveness and practicability, CRO_P1 is the best choice if we use the permutation-based representations.

5.2 Vector-Based Results

We add PSO when using the vector-based representations. Fig. 3 gives the fitness values for the 36 test cases with configurations of (100,10), (300,10), and (512,16). In general, CRO_V1 and CRO_V2 outperform the other four metaheuristics in most of the cases. For those cases, in which CRO_V1 and CRO_V2 are not the best, they can achieve results very close to the best ones. We also find that CRO_V2 performs a little better. The differences of performance among these six algorithms become more pronounced as the scale increases, and the difference is the greatest in the largest configuration (204,864), shown in [31, Fig. S5]. (Below, we will show why PSO_V is not included in this figure.) Fig. S5 also shows that CRO_V2 outperforms all of the other five algorithms in all 12 different cases.

We compute the improvement of an algorithm over another one as follows:

$$Improvement(\%) = \frac{\delta_1 - \delta_2}{\delta_2} \times 100\%, \qquad (6)$$

where δ_1 and δ_2 are the fitness values of two different algorithms. On average, CRO_V1 performs 5.92, 55.72, and 5.50 percent better than TA_V, SA_V, and GA_V in fitness, respectively, while CRO_V2 achieves 9.98, 62.63, and 9.52 percent, respectively.

Moreover, we can also observe that: 1) the lower the task and resource heterogeneity, the worse SA_V performs. For the cases with low task and resource heterogeneity, SA_V turns out to be the worst algorithm. 2) For cases with few tasks and resources, PSO_V and GA_V can produce solutions as good as CRO_V1 and CRO_V2 (albeit not better). However, as the numbers of tasks and resources increase, the performance of CRO_V1 and CRO_V2 improves significantly.

Meanwhile, CRO_V1 and CRO_V2 take similar CPU time as TA_V, SA_V, and GA_V, but PSO_V requires much longer time than the rest. The reason is that the velocity updating in PSO_V consumes much time in each iteration. In the largest configuration, the time PSO_V consumes is much higher than what we can accept (about millions of seconds for a case). Besides, as mentioned before, the performance of PSO_V deteriorates with the adding of tasks and resources. Based on the above two considerations, we exclude PSO_V in the largest scale cases.

In [31, Fig. S6] illustrates the convergence of the algorithms. We choose four cases with a configuration of 512 tasks and 16 resources. An advantage of PSO is its convergence speed. From Fig. S6, it is clear that PSO_V converges faster than the other metaheuristics and its convergence rate is almost double than those of TA_V, SA_V, and GA_V. CRO_V1 and CRO_V2 are the second best in terms of convergence speed.

5.3 Comparison between Permutation and Vector-Based Representations

We compare the results of using permutation-based and vector-based representations in [31, Table S4]. For simplicity, we only show the results of the configurations with high task heterogeneity and high resource heterogeneity. The results of other configurations are similar. The bold numbers are the better results when comparing between permutation-based and vector-based representations for the same algorithms, while the bracketed ones are the best results among all the algorithms. From this table, it is clear that vector-based representation is better than permutation-based one in each algorithm (except for one instance with GA, namely the case of chh_100_10). As described in Section 5.1, GA_P gets the best fitness value, but, it is worse than any algorithm using the corresponding vector-based representations. Furthermore, we also find that the standard deviations of fitness values with permutation-based representation are greater than 7 percent, while the standard deviations of those with vector-based are less than 1 percent. Hence, vector-based representation is superior to the permutation-based one when solving the grid scheduling problem. This is because in our model, we do not consider the workflow (precedence relationship) between tasks, while permutation-based representation mainly handles the problem with sequential solution, like TSP. Therefore, solutions with vector-based representation can be transformed to others with better quality more directly and efficiently than the permutation-based ones.

Both SA and TA manipulate a single solution in each iteration, while GA and PSO control a population of solutions at a time. Though CRO is also a population-based metaheuristic, the number of manipulated solutions at each time instance varies in the course of simulation. With appropriate control through parameter settings, in the early stage, CRO acts more like GA but has more operators to use. As the simulation continues, it tends to keep a single solution in each iteration, like SA. Therefore, CRO enjoys the advantages of both GA and SA, and generally it performs the best. Besides, the structure of CRO is quite flexible. Through adjusting and combining the elementary reactions, we can get new frameworks of CRO which adapt to the specific problem.

5.4 Summary

Basically, the algorithm with vector-based representation is superior to that with permutation-based representation, and CRO_V2 performs the best. In order to further support the superiority of CRO_V2, we adopt the t-tests with 95 percent confidence levels. We compared the means of CRO_V2 and those of other algorihtms with vector-based representation for all the 48 cases, tabulated in [31, Table S5]. "$s+$," "$s-$," and "$\approx$" indicate that CRO_V2 is significantly better, significantly worse, and comparable in performance to the counterpart, respectively. It is clear that CRO_V2 is significantly better than other algorithms in almost every case.

6 Conclusion

Grid Computing has emerged as one of the hot research areas in the field of computer networking. Scheduling, which decides how to distribute tasks to resources, is one of the most important issues. CRO, inspired by the chemical reaction process, is a metaheuristic approach and can be applied to solve NP-hard problems, like grid scheduling. Compared with the previous version [32], we summarize the new contributions of this paper as follows:

1. Based on the canonical framework of CRO, we design a new framework according to the characteristics of the grid scheduling problem.
2. We compare the performance of CRO with other popular metaheuristics, for a large variety of test cases, and with the consideration of the heterogeneous environment of different configurations.
3. Two solution representations, namely, permutation-based and vector-based, are implemented and compared with each other.
4. We provide the optimal parameters for CRO in [31], which can be used as the reference for further research.
5. Weights for the multiobjective function $f(\omega)$ are analyzed in [31].
6. We demonstrate that the vector-based representation is better than the permutation-based one for independent task grid scheduling problem. From the simulation results, we find that CRO_V2 gives the best performance when compared with CRO_V1, GA_V, SA_V, TA_V, and PSO_V.

In the future, our work can be carried forward in the following three directions. First, other models of grid scheduling (e.g., workflow model [28], priority model [5], etc.) can be studied with the CRO approach. Second, to understand CRO further, we can design more operators for CRO, develop some heuristic components dedicated to the grid scheduling problem, and test the performance. In addition, other ways solving multiobjective optimization problem can be adopted.

Acknowledgments

This work was supported in part by the University of Hong Kong Strategic Research Theme of Information Technology. A.Y.S. Lam was also supported in part by the Croucher Foundation Fellowship.

References

[1] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *Int'l J. Supercomputer Applications,* vol. 15, no. 3, pp. 200-222, Aug. 2001.

[2] R. Ranjan, A. Harwood, and R. Buyya, "A Study on Peer-to-Peer Based Discovery of Grid Resource Information," *IEEE Comm. Surveys and Tutorials,* vol. 10, no. 2, pp. 1-42, Apr.-June 2008.

[3] D. Abramson, J. Giddy, and L. Kotler, "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?," *Proc. 14th Int'l Parallel and Distributed Processing Symp. (IPDPS '00),* pp. 520-528, May 2000.

[4] D.P. da Silva, W. Cirne, and F.V. Brasileiro, "Trading Cycles for Information: Using Replication to Schedule Bag-of-tasks Applications on Computational Grids," *Proc. Int'l Conf. Parallel and Distributed Computing, Euro-Par '03,* pp. 169-180, Aug. 2003.

[5] G. Sumathi and N.P. Gopalan, "Priority Based Scheduling for Heterogenous Grid Environments," *Proc. 10th IEEE Singapore Int'l Conf. Comm. Systems (ICCS '06),* pp. 1-5, Oct. 2006.

[6] I. Chlamtac and A. Farago, "Making Transmission Schedules Immune to Topology Changes in Multi-Hop Packet Radio Networks," *IEEE/ACM Trans. Networking,* vol. 2, no. 1, pp. 23-29, Feb. 1994.

[7] A. Capone and G. Carello, "Scheduling Optimization in Wireless MESH Networks with Power Control and Rate Adaptation," *Proc. Third Ann. IEEE Comm. Society on Sensor and Ad Hoc Comm. Networks (SECON '06)*, pp. 138-147, Sept. 2006.

[8] B. Rood and M.J. Lewis, "Resource Availability Prediction for Improved Grid Scheduling," *Proc. IEEE Fourth Int'l Conf. eScience (eScience '08)*, pp. 711-718, Dec. 2008.

[9] G. Ritchie and J. Levine, "A Hybrid Ant Algorithm for Scheduling Independent Jobs in Heterogeneous Computing Environments," *Proc. 23rd Workshop UK Planning and Scheduling Special Interest Group (PlanSIG '04)*, Dec. 2004.

[10] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, May 1983.

[11] J.H. Holland, *Adaptation in Natural and Artificial Systems*. Univ. of Michigan Press, 1975.

[12] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for Optimization from Social Insect Behavior," *Nature*, vol. 406, pp. 39-42, July 2000.

[13] J. Kennedy and R.C. Eberhart, "Particle Swarm Optimization," *Proc. IEEE Int'l Conf. Neural Networks (ICNN '95)*, pp. 1942-1948, Nov. 1995.

[14] F. Glover and M. Laguna, *Tabu Search*. Kluwer Academic Publishers, 1997.

[15] G. Dueck and T. Scheuer, "Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing," *J. Computational Physics*, vol. 90, pp. 161-175, 1990.

[16] A.Y.S. Lam and V.O.K. Li, "Chemical-Reaction-Inspired Metaheuristic for Optimization," *IEEE Trans. Evolutionary Computation*, vol. 14, no. 3, pp. 381-399, June 2010.

[17] R.P. Bruin, M.T. Dove, M. Calleja, and M.G. Tucker, "Building and Managing the eMinerals Clusters: A Case Study in Grid-Enabled Cluster Operation," *Computing in Science and Eng.*, vol. 7, no. 6, pp. 30-37, Nov./Dec. 2005.

[18] F. Gagliardi, B. Jones, F. Grey, M.-E. Begin, and M. Heikkurinen, "Building an Infrastructure for Scientific Grid Computing: Status and Goals of the EGEE project," *Philosophical Trans. Series A Math., Physical and Eng. Sciences*, vol. 363, no. 1833, pp. 1729-1742, Aug. 2005.

[19] J. Frey, T. Tannenbaum, M. Livny, F. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *J. Cluster Computing*, vol. 5, no. 3, pp. 237-246, 2002.

[20] F. Berman, R. Wolski, H. Casanova, W. Cirne et al., "Adaptive Computing on the Grid Using AppLes," *IEEE Trans. Parallel and Distributed Systems*, vol. 14, no. 4, pp 369-382, Apr. 2003.

[21] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," *Proc. 14th Conf. Scientific and Statistical Database Management*, pp. 37-46, 2002.

[22] F. Xhafa and A. Abraham, "Meta-Heuristics for Grid Scheduling Problems," *Studies in Computational Intelligence*, vol. 146, pp. 1-37, 2008.

[23] V. Ungureanu, B. Melamed, M. Katehakis, and P.G. Bradford, "Deferred Assignment Scheduling in Cluster-Based Servers," *J. Cluster Computing*, vol. 9, no. 1, pp. 57-65, 2006.

[24] S. Zikos and H.D. Karatza, "Resource Allocation Strategies in a 2-Level Hierarchical Grid System," *Proc. 41st Ann. Simulation Symp. (ANSS)*, pp. 157-164, Apr. 2008.

[25] T.D. Braun et al., "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810-837, June 2001.

[26] S. Fidanova, "Simulated Annealing for Grid Scheduling Problem," *Proc. IEEE John Vincent Atanasoff Int'l Symp. Modern Computing (JVA '06)*, pp. 41-45, Oct. 2006.

[27] H. Izakian, B.T. Ladani, K. Zamanifar, and A. Abraham, "A Novel Particle Swarm Optimization Approach for Grid Job Scheduling," *Proc. Third Int'l Conf. Information Systems, Technology and Management*, vol. 31, pp. 100-109, Mar. 2009.

[28] S. Benedict, R.S. Rejitha, and V. Vasudevan, "Threshold Accepting Scheduling Algorithm for Scientific Workflows in Wireless Grids," *Proc. IEEE Fourth Int'l Conf. Networked Computing and Advanced Information Management (NCM '08)*, pp. 686-691, Sept. 2008.

[29] S. Ali, H.J. Siegel, M. Maheswaran, and D. Hensgen, "Task Execution Time Modeling for Heterogeneous Computing Systems," *Proc. Ninth Heterogeneous Computing Workshop (HCW '00)*, pp. 185-199, May 2000.

[30] A. Abraham, H. Liu, C. Grosan, and F. Xhafa, "Nature Inspired Meta-Heuristics for Gird Scheduling: Single and Multi-Objective Optimization Approaches," *Metaheuristics for Scheduling in Distributed Computing Environments*, vol. 146, pp. 247-272, 2008.

[31] Supporting Document

[32] J. Xu, A.Y.S. Lam, and V.O.K. Li, "Chemical Reaction Optimization for the Grid Scheduling Problem," *IEEE Int'l Conf. Comm. (ICC)*, May 2010.

Jin Xu received the BEng degree in electronic engineering from the Tsinghua University, Beijing, China, in 2008. Currently, he is working toward the PhD degree in electrical and electronic engineering from the University of Hong Kong (HKU). His research interests include evolutionary algorithms and their applications, parallel and distributed computing, grid computing, bioinformatics, wireless and opportunistic networks. He is student member of the IEEE.

Albert Y.S. Lam (S'03-M'10) received the BEng degree (First Class Honors) in information engineering and the PhD degree in electrical and electronic engineering from the University of Hong Kong, China, in 2005 and 2010, respectively. He is now a postdoctoral scholar at the Department of Electrical Engineering and Computer Sciences of University of California, Berkeley, California. He is a Croucher research fellow. He also serves the IEEE Computational Intelligence Society GOLD Subcommittee and an ad hoc committee as a member. His research interests include optimization theory and algorithms, evolutionary computation, information theory, wireless and mobile networking, Internet protocols and applications. He is a member of the IEEE.

Victor O.K. Li (S'80-M'81-SM'86-F'92) received SB, SM, EE, and ScD degrees in electrical engineering and computer science from MIT in 1977, 1979, 1980, and 1981, respectively. He is associate dean of Engineering and chair professor of information engineering at the University of Hong Kong (HKU), visiting professor at King Saud University, Saudi Arabia, and guest chair professor of Wireless Communication and Networking at Tsinghua University, Beijing, China. He also served as managing director of Versitech Ltd., the technology transfer and commercial arm of HKU, and on the boards of Sunevision Holdings Ltd. and China.com Ltd. Previously, he was professor of electrical engineering at the University of Southern California (USC), Los Angeles, California, and Director of the USC Communication Sciences Institute. Sought by government, industry, and academic organizations, he has lectured and consulted extensively around the world. He has received numerous awards, including the PRC Ministry of Education Changjiang Chair Professorship at Tsinghua University, the UK Royal Academy of Engineering Senior Visiting Fellowship in Communications, the Croucher Foundation Senior Research Fellowship, and the Order of the Bronze Bauhinia Star, Government of the Hong Kong Special Administrative Region, China. He is a Registered Professional Engineer and a Fellow of the IAE, and the HKIE. He is a fellow of the IEEE.