Chemical-Reaction-Inspired Metaheuristic for Optimization

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

***Abstract*—We encounter optimization problems in our daily lives and in various research domains. Some of them are so hard that we can, at best, approximate the best solutions with (meta-) heuristic methods. However, the huge number of optimization problems and the small number of generally acknowledged methods mean that more metaheuristics are needed to fill the gap. We propose a new metaheuristic, called chemical reaction optimization (CRO), to solve optimization problems. It mimics the interactions of molecules in a chemical reaction to reach a low energy stable state. We tested the performance of CRO with three nondeterministic polynomial-time hard combinatorial optimization problems. Two of them were traditional benchmark problems and the other was a real-world problem. Simulation results showed that CRO is very competitive with the few existing successful metaheuristics, having outperformed them in some cases, and CRO achieved the best performance in the real-world problem. Moreover, with the No-Free-Lunch theorem, CRO must have equal performance as the others on average, but it can outperform all other metaheuristics when matched to the right problem type. Therefore, it provides a new approach for solving optimization problems. CRO may potentially solve those problems which may not be solvable with the few generally acknowledged approaches.**

***Index Terms*—Chemical reaction, metaheuristics, nature-inspired algorithms, optimization methods.**

I. Introduction

OPTIMIZATION is prevalent in almost every field of science and engineering, ranging from profit maximization in economics to signal interference minimization in electrical engineering. In our daily lives, we also encounter various optimization problems, such as finding the quickest route from one place to another, at minimum cost, and minimizing the construction costs of building facilities in a city, while, at the same time, avoiding congestion of human flow among such facilities. Optimization refers to the study of problems in which one seeks to optimize (either minimize or maximize) the result by systematically choosing the values of the variables in feasible regions. We normally define an optimization problem with several components: an objective function f, a vector of variables $X = \{x_1, x_2, \ldots, x_n\}$, and a vector of constraints $C = \{c_1, c_1, \ldots, c_m\}$ which limit the values assigned to X, where n and m correspond to the problem dimensions and the total number of constraints, respectively. We define a solution s as the set of values assigned to X confined by C, and the solution space S as the set of all possible solutions. For minimization problems, our goal is to find the minimum solution $s^* \in S$ where $f(s^*) \leq f(s)$ for all s. We can write

$$\min_{X \in R^n} f(X) \quad \text{subject to} \quad \begin{cases} c_i(X) = 0, & i \in E \\ c_i(X) \leq 0, & i \in I \end{cases} \tag{1}$$

where R, E, and I represent the real number set, the index set for equalities, and the index set for inequalities, respectively. Equation (1) represents the generic form for every type of optimization. Without loss of generality, we consider minimization problems throughout this paper.[1] What we need to do is to search the solution space and pick out solution points sequentially. Then one evaluates the objective function value of each solution point. An optimization method (i.e., algorithm) tells us which point should be picked from the current solution. We can get one, or multiple, points in an instance, depending on how the algorithm operates.

We can formulate many problems into this generic form, i.e., (1), and then apply the existing methods to obtain the optimal solutions, with the help of the computer. However, in computation complexity theory [1], there is a class of problems, namely, nondeterministic polynomial-time hard (NP-hard) problems, with no known algorithms in finding the optimal solutions in polynomial time, unless P = NP. In other words, for such problems, the computational effort required to obtain the best solutions grows exponentially with the problem size. They are normally not solvable by any optimization algorithms in a reasonable amount of time or we cannot guarantee that the computed results are of high quality. Most of the time, the formulated problems are of huge dimensions and examining every possible solution (i.e., the brute-force method) becomes impossible. It may take several years of CPU time to obtain the solutions, in spite of using the most powerful supercomputer. We often cannot tolerate such long computational time and sacrifice optimality for near-optimal solutions if the processing time is limited. Thus, we always adopt approximate algorithms, which can compute "good" solutions efficiently, to tackle the NP-hard problems.

Manuscript received August 8, 2008; revised February 23, 2009, June 4, 2009, and September 3, 2009. First version published December 15, 2009; current version published May 28, 2010. This work was supported in part by the Strategic Research Theme of Information Technology of The University of Hong Kong.

The authors are with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam, Hong Kong (e-mail: ayslam@eee.hku.hk; vli@eee.hku.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEVC.2009.2033580

[1]Maximization problems work similarly, by simply adding a negative sign to f.

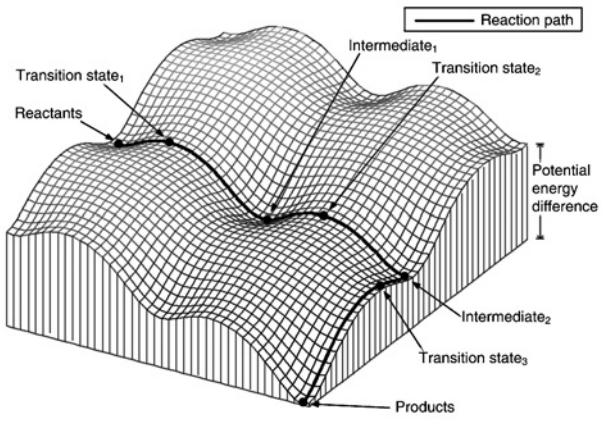


Fig. 1. Potential energy surface of a chemical reactive system.

In quantum mechanics and statistical mechanics, we can model chemical reactions and molecular interactions with potential energy surface (PES) (Fig. 1), which is subject to the Born–Oppenheimer separation of nuclear and electronic motion [2]. Fig. 1 depicts the potential energy (*PE*) changes of the atom arrangements in a chemical system. The z-axis represents the *PE* while the x and y-axes capture the molecular structures of the chemical substances, which correspond to the atomic positions and every possible orientation of all the involved atomic nuclei. PES can be a two, three, or multi-dimensional (hyper)surface, depending on how complicated the chemical system is. In any chemical reaction, the initial species (i.e., reactants) change to the products by the formation and destruction of chemical bonds. Before the formation of products, the reactants normally change to a series of intermediate species. These small chemical changes are called elementary steps. During each step, the chemicals are in the transition states. Fig. 1 shows a simple example of a chemical reaction involving three elementary steps. The solid line gives the reaction pathway from the reactants to products, via several transition states and intermediate species.

There is a rule of thumb for this natural tendency—"Every reacting system seeks to achieve a minimum of free energy"[2] [3]. That means chemical reactions tend to release energy, and thus, products generally have less energy than reactants. In terms of stability, the lower the energy of the substance, the more stable it is. Therefore, products are always more stable than reactants.

It is not difficult to discover the correspondence between optimization and chemical reaction. Both of them aim to seek the global minimum (but with respect to different objectives) and the process evolves in a stepwise fashion. With this discovery, we develop a chemical-reaction-inspired metaheuristic for solving optimization problems by mimicking what happens to molecules in chemical reactions. We name it chemical reaction optimization (CRO). It is a multidisciplinary design which loosely couples computation with chemistry.

The rest of this paper is organized as follows. Section II briefly describes some basic concepts of optimization and related work. In Section III, we give the design framework of CRO and show how the concept of chemical reaction is implemented in our algorithm. We show the workability of CRO with evaluations using computer simulations in Section IV. We conclude this paper and give some potential future work in Section V.

II. Background

Metaheuristics are collections of ideas aiming to solve general computational problems. A metaheuristic is usually in the form of a procedure framework which instructs computers how to search for solutions in the solution space. Each metaheuristic consists of several building blocks and control parameters for fine tuning. We can replace these components and/or change the parameter values in order to suit our purposes. From this, we can see metaheuristics contain a high degree of flexibility. Most of the metaheuristics involve randomization in the calculation, and thus, the outputs may vary in different runs of the computation. Since exact optimal solutions are not guaranteed, they belong to the group of approximate algorithms. We adopt them to solve NP-hard optimization problems because they can locate good solutions efficiently most of the time.

This paper is motivated by other efforts to apply natural phenomena to metaheuristics. Among the most famous ones are simulated annealing (SA) [4], genetic algorithm (GA) [5]–[7], and ant colony optimization (ACO) [8], [9]. Other proposed metaheuristics include particle swarm optimization (inspired by the social behavior of bird flocking) [10], bees algorithm (inspired by the behavior of honey bees in collecting nectar) [11], harmony algorithm (inspired by the improvisation process of musicians) [12], etc. There are non-nature-inspired metaheuristics also, like tabu search (TS) [13]. We will briefly introduce SA, GA, ACO, and TS in the following sections.

A. Simulated Annealing

SA is inspired by annealing in metallurgy. Annealing is the physical process of increasing the crystal size of a material and reducing the defects through a controllable cooling procedure. SA picks a solution in each iteration. By employing the Metropolis algorithm [14] from statistical mechanics, SA always allows downhill movements, while uphill movements are allowed with a probability whose distribution is controlled by a so-called temperature parameter. Therefore, it does not always get stuck at local minima. As the temperature drops, the ability to jump out of local minima decreases and the system converges to the final solution.

B. Genetic Algorithm

Holland [5] created GAs based on the idea of natural selection, which is the phenomenon that organisms with favorable characteristics have higher probability to survive

[2]Free energy is also known as Gibbs free energy, which indicates the amount of energy needed for a system to do useful work at constant temperature and pressure. In a chemical reaction, the reactants have higher free energy, and thus, they can do useful work (i.e., react with the others). At equilibrium (i.e., final stage of the reaction), the products have a minimum of free energy so they can no longer react.

and reproduce than those with unfavorable traits. GA is a population-based metaheuristic and simulates this biological process through producing generations of chromosomes, which represent possible solutions of the optimization problems. Through inheritance, selection, and crossover, those chromosomes which are favored by the objective functions, and which satisfy the constraints, can survive and reproduce the next generation of chromosomes with higher quality. It can escape from local optima through mutation.

C. Ant Colony Optimization

ACO is also population-based and mimics the ecological behavior of ants in finding food. Food paths represent solutions. When ants discover paths to the food locations from their colony, they lay down a chemical, called pheromone, along the paths to remind other ants about the food trails. Shorter paths have more pheromone as more ants shuttle around. It employs the effect of evaporation of pheromone to prevent getting stuck with local optima. We can obtain the best solution by checking the route with the greatest amount of pheromone.

D. Tabu Search

TS is introduced by Glover [13] and it is a non-nature-inspired metaheuristic. The core is local search together with a tabu mechanism. In each iteration, the algorithm searches the neighborhood of the current solution to get a new one with an improved functional value. At the same time, it maintains a tabu list, which contains the solutions obtained in the recent iterations. The purpose is to prevent looping in the recent solutions and to diversify the search to an unexplored region of the search space. Sometimes the tabu mechanism may be too restrictive and forbid some attractive moves. TS allows overriding the tabu list if the newly picked solution meets certain aspiration criteria.

E. Development of Optimization Algorithms

Generally, we can classify optimization algorithms into heuristics and metaheuristics. Heuristics are different from metaheuristics in that the former are tailor-made for specific problems. They may be able to solve some problems very well but may give poor solutions to others. On the other hand, (well-designed) metaheuristics can be applied to a broader range of problems and results in good performance. For a specific problem, a tailor-made heuristic normally performs better than a metaheuristic, but the heuristic may not be readily available. If the heuristic does not exist, we may utilize a metaheuristic to solve the problem. The relationship between heuristics and metaheuristics is an accuracy-flexibility tradeoff.

When we encounter a new problem with no polynomial-time algorithm available, we consider metaheuristics. If a metaheuristic seems to work well on the problem, greedy and heuristic components may be added in order to "maximize" its performance. Although the resultant algorithm may have better performance on this problem, it becomes more heuristic-like and may not be able to solve other problems well. In this way, we sacrifice flexibility for accuracy. We can take SA as an example. SA was firstly introduced in [4] and then applied to quadratic assignment problem (QAP) in [15]. Afterward modified versions were proposed sequentially in [16]–[19] with better performance, and they became more heuristic-like. Hybrid algorithms with SA and TS are also possible [20].

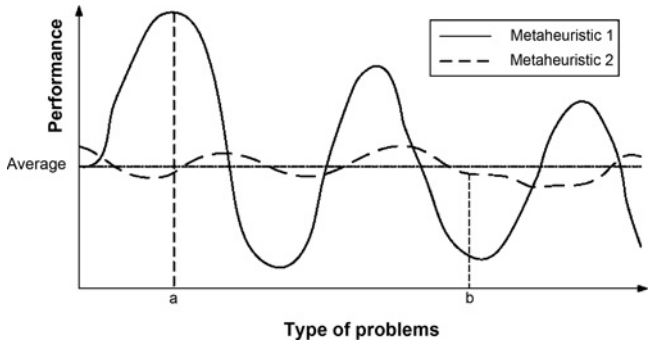


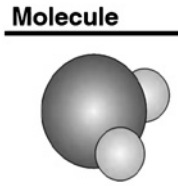
Fig. 2. Comparison of performance for different problem types.

F. No-Free-Lunch (NFL) Theorem

We will never be satisfied with the existing metaheuristics, such as those mentioned above, even though they enjoy great success in solving many optimization problems, e.g., [21]–[23]. According to the NFL theorem [24], all metaheuristics which search for extrema are exactly the same in performance when averaged over all possible objective functions. All metaheuristics perform statistically identically on solving computational problems. Ho and Pepyne [25] further elaborate upon the idea, showing that it is theoretically impossible to have a best general-purpose universal optimization strategy, and the only way for one strategy to be superior to the others is when we focus on a particular class of problems only. With prior knowledge on the problem under consideration, a metaheuristic can be modified and, thus, may become more suited to the problem. However, this kind of modification may not always be successful. For example, it is natural to apply ACO to routing-related problems [26]. The performance may be worse with other methods, say SA [27]. One can deduce that a particular metaheuristic is, by nature, more easily transformed to suit specific classes of optimization problems than others.

G. Summary

No one algorithm can always, on average, surpass the others in all possible optimization problems. However, superior performance is still possible in a particular problem. Fig. 2 shows the comparison of performance for different problem types. Every metaheuristic has equal performance on the average. One may have superior performance for some types of problems but becomes inferior on other problems. At point (a), metaheuristic 1 outperforms metaheuristic 2, but at point (b), metaheuristic 2 outperforms metaheuristic 1. Hereafter successful metaheuristics refer to those which are governed by the NFL theorem, and which is successful in solving some problems. However, the "spectrum" of problems is so huge that we cannot find the best match for each of them. Thus, it is worthwhile to bring forth a new optimization search strategy if we can prove it works well in some problems. This helps open



Molecule	Chemical meaning	Mathematical meaning
	Molecular structure	Solution
	Potential energy	Objective function value
	Kinetic energy	Measure of tolerance of having worse solutions
	Number of hits	Current total number of moves
	Minimum structure	Current optimal solution
	Minimum value	Current optimal function value
	Minimum hit number	Number of moves when the current optimal solution is found

Fig. 3. Profile of a molecule. The first column contains the properties of a molecule used in CRO. The second column shows the corresponding meanings in the metaheuristic.

up new territory for optimization; new optimization methods may resolve the "unsolved" problems well. That is the reason why we propose CRO for solving optimization problems.

III. Design Framework

CRO loosely mimics what happens to molecules in a chemical reaction system microscopically. It tries to capture the phenomenon that reactions give products with the lowest energy on the PES. In the following sections, we will first describe the major components of the design, i.e., molecules and elementary reactions. Then we will give the basic idea of the design. Next, we will explain how to bring the idea into reality, i.e., how to do computation with the idea in terms of an algorithm. Finally, we discuss CRO from the angle of optimization and highlight its unique features.

A. Molecules

The manipulated agents are molecules and each has a profile containing some properties (Fig. 3). A molecule is composed of several atoms and characterized by the atom type, bond length, angle, and torsion. One molecule is distinct from another when they contain different atoms and/or different number of atoms. If two molecules have exactly the same set of atoms but with different molecular attributes (i.e., bond length, angle, and torsion), we will consider them distinct molecules. We utilize the term "molecular structure" to summarize all these characteristics and it corresponds to a solution in the mathematical domain. The presentation of a molecular structure depends on the problem we are solving, provided that it can express a feasible solution of the problem.[3] For example, if a problem defines the feasible solution set as the set of n-dimensional positive real numbers R_+^n, then any vector with n elements whose values are positive real numbers is a valid molecular structure, and no molecule structure can contain numbers with non-positive values.[4] A change in molecular structure is tantamount to switching to another feasible solution. A molecule possesses two kinds of energies, i.e., *PE* and kinetic energy (*KE*). The former quantifies the molecular structure in terms of energy and we model it as the objective function value when evaluating the corresponding solution. Let ω and f denote a molecular structure (or a solution) and an objective function. Then

$$PE_\omega = f(\omega). \tag{2}$$

The latter does not have such a direct analogy. We use it as a measure of tolerance for the molecule changing to a less favorable structure (i.e., a solution with higher functional value). For example, a molecule intends to change from ω to ω'. The change is always possible if $PE_\omega \geq PE_{\omega'}$. Otherwise, we allow the change only when $PE_\omega + KE_\omega \geq PE_{\omega'}$.[5] Thus, the higher the *KE* of the molecule, the higher the possibility it can possess a new molecular structure with higher *PE*. Recall that the molecules involved in a reaction attempt to reach the lowest possible potential state, but blindly seeking more favorable structures (i.e., a solution with lower functional value) will result in metastable states (i.e., getting stuck in local minima). *KE* allows the molecules to move to a higher potential state, and hence a chance of having a more favorable structure in a future change. Therefore, *KE* of a molecule symbolizes its ability of escaping from a local minimum. With the conservation of energy, energy cannot be created or destroyed. We cannot intentionally add or remove *KE* to a molecule. Nevertheless, we allow the conversion between *PE* and *KE*, within a molecule or among molecules, through some elementary reactions (or steps). As will be explained in the next section, we intend to draw *KE* of the molecules to a central energy buffer (*buffer*), and thus, the molecules are getting less *KE* as the algorithm evolves. In other words, we drive them to possess molecular structures with lower and lower *PE* in the subsequent changes. This phenomenon is the driving force in CRO to ensure convergence to lower energy state. The rest of the properties listed in Fig. 3, i.e., number of hits, minimum structure, minimum value and minimum hit number, are used in an implementation of the algorithm. Their uses will be discussed in Section IV-A.

B. Elementary Reactions

In a chemical reaction process, a sequence of collisions among molecules occurs. Molecules collide either with each other or with the walls of the container. Collisions under different conditions provoke distinct elementary reactions, each of which may have a different way of manipulating the energies of the involved molecule(s). There are four types of elementary reactions implemented in CRO (Fig. 4), namely, on-wall ineffective collision, decomposition, inter-molecular ineffective collision, and synthesis. These elementary reactions may be categorized in terms of molecularity and extent of change of the molecular structure. By molecularity, on-wall ineffective collision and decomposition are unimolecular reactions triggered when the molecule hits a wall of the container, while inter-molecular ineffective collision and synthesis involve more than one molecule. They take place when molecules collide with each other. By the extent of

[3]All molecular structures corresponding to a problem are of the same solution format. They vary only with the values assigned to them and do not represent partial solutions.

[4]In Section IV, CRO is applied to the quadratic assignment problem. Assume the problem dimension is n. The feasible solution set is the set of permutations of n numbers. Then the molecular structure of a molecule can be a permutation of n numbers.

[5]Note that the change is not restricted to a single molecule. It can involve more than one molecule simultaneously. This will be explained in the next section.

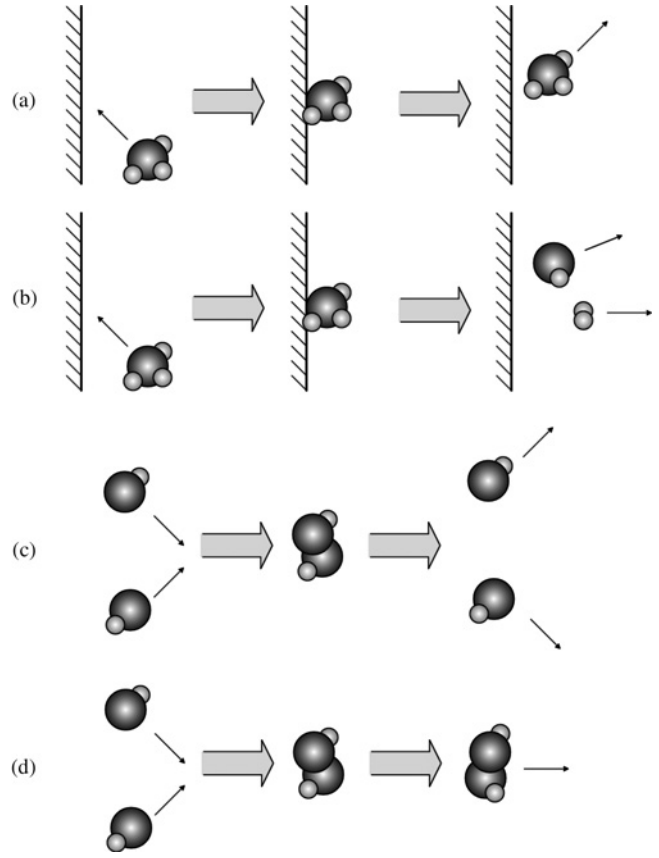


Fig. 4. Four elementary reactions implemented in CRO. (a) On-wall ineffective collision. (b) Decomposition. (c) Inter-molecular ineffective collision. (d) Synthesis.

change in the molecular structure of the resultant molecule(s), on-wall and inter-molecular ineffective collisions react much less vigorously than decomposition and synthesis. Ineffective collisions correspond to those cases in which the molecules get new molecular structures in their own neighborhoods on PES (i.e., they pick new solutions close to the original ones). Thus, the *PE* of the resultant molecules tends to be close to those of the original ones. Conversely, decomposition and synthesis tend to obtain new molecular structures which may be far away from their immediate neighborhoods on PES. When compared with ineffective collisions, the resultant molecules are apt to have greater change in PE than the original ones.

1) *On-wall Ineffective Collision:* An on-wall ineffective collision [Fig. 4(a)] occurs when a molecule hits the wall and then bounces back. Some molecular attributes change in this collision, and thus, the molecular structure varies accordingly. As the collision is not so vigorous, the resultant molecular structure should not be too different from the original one. Suppose the current molecular structure is ω. The molecule intends to obtain a new structure $\omega' = Neighbor(\omega)$ (Table I) in its neighborhood[6] on the PES in this collision. The change is allowed only if

$$PE_\omega + KE_\omega \geq PE_{\omega'}. \tag{3}$$

[6]The neighborhood structure is problem-dependent. Normally, ω and its neighbors have similar *PE*.

We get

$$KE_{\omega'} = (PE_\omega + KE_\omega - PE_{\omega'}) \times q$$

where $q \in [KELossRate, 1]$, and $(1 - q)$ represents the fraction of *KE* lost to the environment when it hits the wall. *KELossRate* is a system parameter which limits the maximum percentage of *KE* lost at a time. The lost energy is stored in the central energy buffer.[7] The stored energy can be used to support decomposition. If (3) does not hold, the change is prohibited and the molecule retains its original ω, *PE* and *KE*. The pseudocode of the on-wall ineffective collision is as follows:

ineff_coll_on_wall(M, *buffer*)

Input: A molecule M with its profile and the central energy buffer *buffer*.

1. Obtain $\omega' = Neighbor(\omega)$
2. Calculate $PE_{\omega'}$
3. **if** $PE_\omega + KE_\omega \geq PE_{\omega'}$ **then**
4. Get q randomly in interval [*KELossRate*, 1]
5. $KE_{\omega'} = (PE_\omega + KE_\omega - PE_{\omega'}) \times q$
6. Update $buffer = buffer + (PE_\omega + KE_\omega - PE_{\omega'}) \times (1 - q)$
7. Update the profile of M by $\omega = \omega'$, $PE_\omega = PE_{\omega'}$ and $KE_\omega = KE_{\omega'}$
8. **end if**
9. **Output** M and *buffer*

2) *Decomposition:* A decomposition [Fig. 4(b)] means that a molecule hits the wall and then decomposes into two or more (assume two in this framework) pieces. The collision is vigorous and leads the molecule to break into two pieces. The resultant molecular structures should be very different from the original one. Suppose the molecular structure of the original molecule is ω and those of the resultant molecules are ω_1' and ω_2'. If the original molecule has sufficient energy (*PE* and *KE*) to endow the *PE* of the resultant ones, that is

$$PE_\omega + KE_\omega \geq PE_{\omega_1'} + PE_{\omega_2'}, \tag{4}$$

the change is allowed. Let $temp_1 = PE_\omega + KE_\omega - PE_{\omega_1'} - PE_{\omega_2'}$. We get

$$KE_{\omega_1'} = temp_1 \times k$$

and

$$KE_{\omega_2'} = temp_1 \times (1 - k)$$

where k is a random number uniformly generated from the interval [0, 1]. However, it is rather unusual for (4) to hold. In normal cases, PE_ω, $PE_{\omega_1'}$ and $PE_{\omega_2'}$ are of similar values (but much larger than those in the same neighborhood), (4) holds only when KE_ω is large enough. However, *KE* of molecules tends to decrease in a sequence of on-wall ineffective collisions as the chemical process evolves. Thus, (4) is not likely

[7]The conservation of energy also prevents us from intentionally adding or removing energy from the energy buffer. The change of energy here is governed only by the mechanisms of the relevant elementary reactions.

TABLE I
SYMBOLS USED IN THE ALGORITHM

Type	Symbol	Algorithmic Meaning	Chemical Meaning
Function	*f*	Objective function	Function defining PES
	Neighbor	Neighbor candidate generator	Neighborhood structure on PES
Variable	*NVars*	Number of variables representing a solution; dimensions of the problem	Total number of characteristic of molecule
	Pop	Set of solution; 2-D matrix where each row carries the values of a solution	Set of molecules
	PE	Vector of objective function values; *PE = f (Pop)*	Potential energy of all the molecules
	KE	Vector of number measuring the tolerance of the solutions to have worse objective function values afterward	Kinetic energy of all the molecules
Parameter	*PopSize*	Initial number of solutions maintained; number of rows in *Pop*	Initial number of molecules in the container
	KELossRate	Percentage upper limit of reduction of *KE* in on-wall ineffective collisions	Percentage upper limit of *KE* lost to the environment in on-wall ineffective collisions
	MoleColl	Fraction of all elementary reactions corresponding to inter-molecular reactions	Same as the algorithmic meaning
	InitialKE	Initial value assigned t each element of *KE* in the initialization stage	*KE* of the initial set molecules

to hold in normal cases. To encourage decomposition, we use the energy stored in the central buffer (*buffer*) to sustain $PE_{\omega_1'}$ and $PE_{\omega_2'}$. In other words, if (4) does not hold, we consider

$$PE_\omega + KE_\omega + buffer \geq PE_{\omega_1'} + PE_{\omega_2'}. \quad (5)$$

If (5) holds, the change is allowed and we calculate

$$KE_{\omega_1'} = (temp_1 + buffer) \times m_1 \times m_2 \quad (6)$$

and

$$KE_{\omega_2'} = (temp_1 + buffer - KE_{\omega_1'}) \times m_3 \times m_4 \quad (7)$$

where m_1, m_2, m_3 and m_4 are random numbers independently uniformly generated from the interval [0, 1]. Multiplication by the two random numbers in both (6) and (7) ensure that the values assigned to $KE_{\omega_1'}$ and $KE_{\omega_2'}$ are not too large, as *buffer* is usually large. Then *buffer* is updated by $temp_1 + buffer - KE_{\omega_1'} - KE_{\omega_2'}$. If both (4) and (5) do not hold, the decomposition fails and the molecule retains its original ω, *PE* and *KE*.

In the design framework, we do not specify how to generate ω_1' and ω_2' from ω. Any mechanism, resulting in ω_1' and ω_2' that are quite different from ω, is reasonable. An example of this mechanism (circular shift) will be given in Section IV. The pseudocode of the decomposition is as follows:

decompose(M, *buffer*)

Input: A molecule M with its profile and the central energy buffer *buffer*.

1. Obtain ω_1' and ω_2' from ω
2. Calculate $PE_{\omega_1'}$ and $PE_{\omega_2'}$
3. Let $temp_1 = PE_\omega + KE_\omega - PE_{\omega_1'} - PE_{\omega_2'}$
4. Create a Boolean variable *Success*
5. **if** $temp_1 \geq 0$ **then**
6. *Success* = TRUE
7. Get k randomly in interval [0, 1]
8. $KE_{\omega_1'} = temp_1 \times k$
9. $KE_{\omega_2'} = temp_1 \times (1 - k)$
10. Create new molecules M_1' and M_2'
11. Assign ω_1', $PE_{\omega_1'}$ and $KE_{\omega_1'}$ to the profile of M_1', and ω_2', $PE_{\omega_2'}$ and $KE_{\omega_2'}$ to the profile of M_2'
12. **else if** $temp_1 + buffer \geq 0$ **then**
13. *Success* = TRUE
14. Get m_1, m_2, m_3, and m_4 independently randomly in interval [0, 1]
15. $KE_{\omega_1'} = (temp_1 + buffer) \times m_1 \times m_2$
16. $KE_{\omega_2'} = (temp_1 + buffer - KE_{\omega_1'}) \times m_3 \times m_4$
17. Update $buffer = temp_1 + buffer - KE_{\omega_1'} - KE_{\omega_2'}$
18. Assign ω_1', $PE_{\omega_1'}$ and $KE_{\omega_1'}$ to the profile of M_1', and ω_2', $PE_{\omega_2'}$ and $KE_{\omega_2'}$ to the profile of M_2'
19. **else**
20. *Success* = FALSE
21. **end if**
22. **Output** M_1' and M_2', *Success* and *buffer*

3) *Inter-Molecular Ineffective Collision:* An inter-molecular ineffective collision [Fig. 4(c)] describes the situation when two molecules collide with each other and then bounce away. The effect of energy change of the molecules is similar to that in an on-wall ineffective collision, but this elementary reaction involves more than one molecule (assume two molecules in this framework) and no *KE* is drawn to the central energy buffer. Suppose the original molecular structures are ω_1 and ω_2. We obtain two new molecular structures ω_1' and ω_2' from the neighborhoods of ω_1 and ω_2, respectively. We accept the changes to the molecules only if

$$PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} \geq PE_{\omega_1'} + PE_{\omega_2'}. \quad (8)$$

Let $temp_2 = (PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2}) - (PE_{\omega_1'} + PE_{\omega_2'})$. We get

$$KE_{\omega_1'} = temp_2 \times p$$

and

$$KE_{\omega_2'} = temp_2 \times (1 - p)$$

where p is a random number uniformly generated from the interval [0, 1]. The molecules maintain the original ω_1, ω_2, PE_{ω_1}, PE_{ω_2}, KE_{ω_1}, and KE_{ω_2} if (8) fails. Inter-molecular

ineffective collision allows the molecular structure to change in a larger extent, as two molecules are involved and so the sum of the possessed KE is larger. The pseudocode of the inter-molecular ineffective collision is as follows:

inter_ineff_coll(M_1, M_2)

Input: molecules M_1, M_2 with their profiles.
1. Obtain $\omega_1' = Neighbor(\omega_1)$ and $\omega_2' = Neighbor(\omega_2)$
2. Calculate $PE_{\omega_1'}$ and $PE_{\omega_2'}$
3. Let $temp_2 = (PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2}) - (PE_{\omega_1'} + PE_{\omega_2'})$
4. **if** $temp_2 \geq 0$ **then**
5. Get p randomly in interval [0, 1]
6. $KE_{\omega_1'} = temp_2 \times p$
7. $KE_{\omega_2'} = temp_2 \times (1 - p)$
8. Update the profile of M_1 by $\omega_1 = \omega_1'$, $PE_{\omega_1} = PE_{\omega_1'}$ and $KE_{\omega_1} = KE_{\omega_1'}$, and the profile of M_2 by $\omega_2 = \omega_2'$, $PE_{\omega_2} = PE_{\omega_2'}$ and $KE_{\omega_2} = KE_{\omega_2'}$
9. **end if**
10. **Output** M_1 and M_2

4) *Synthesis:* A synthesis [Fig. 4(d)] depicts more than one molecule (assume two molecules) which collide and combine together. Suppose the molecular structures of the two original molecules are ω_1 and ω_2. We attempt to generate a new molecule with molecular structure ω' from the two existing ω_1 and ω_1. Since synthesis is vigorous, ω' should be quite different from ω_1 and ω_2. As in decomposition, any mechanism which combines ω_1 and ω_2 to form ω' may be used. We accept ω' only if

$$PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} \geq PE_{\omega'}. \tag{9}$$

We get $KE_{\omega'} = PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} - PE_{\omega'}$. We retain ω_1, ω_2, PE_{ω_1}, PE_{ω_2}, KE_{ω_1}, and KE_{ω_2}, instead of ω', $PE_{\omega'}$ and $KE_{\omega'}$, if (9) does not hold. Interestingly, $KE_{\omega'}$ is large when compared with KE_{ω_1} or KE_{ω_2}, because $PE_{\omega'}$ is expected to have similar value to PE_{ω_1} or PE_{ω_2} generally. In this way, we give the resultant molecule with ω' greater ability to escape from a local minimum in subsequent elementary reactions involving it. The pseudocode of the synthesis is as follows:

synthesis(M_1, M_2)

Input: molecules M_1, M_2 with their profiles.
1. Obtain ω' from ω_1 and ω_2
2. Calculate $PE_{\omega'}$
3. Create a Boolean variable *Success*
4. Create a new molecule M'
5. **if** $PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} \geq PE_{\omega'}$ **then**
6. *Success* = TRUE
7. $KE_{\omega'} = PE_{\omega_1} + PE_{\omega_2} + KE_{\omega_1} + KE_{\omega_2} - PE_{\omega'}$
8. Assign ω', $PE_{\omega'}$ and $KE_{\omega'}$ to the profile of M'
9. **else**
10. *Success* = FALSE
11. **end if**
12. **Output** M' and *Success*

C. The Basic Idea

We try to explore different parts of PES as much as possible to locate the lowest PE point, but normally, the PES is so large that it is impossible to examine every point within a reasonable period of time. Therefore, we have to intelligently explore only those parts of PES, where the minimum may reside with high possibility. We implement the exploration with collisions among molecules, bringing them toward the lowest possible energy state through several types of elementary reactions. Generally, there are two approaches to do the search, i.e., intensification and diversification.[8] For each molecule starting at a point on the PES, intensification explores the immediately surrounding area. Whenever we fail to find an even lower energy state in this area for some time, diversification allows us to jump to a relatively distant area to continue the search. In CRO, intensification is mainly contributed by on-wall ineffective collision and inter-molecular ineffective collision, while decomposition and synthesis perform diversification. At the same time, the system tries to redistribute the energies among the molecules by interchanging the energies from one to another in different ways.

Suppose we decide to perform a chemical reaction. We initialize a set of reactant molecules by assigning molecular structures randomly and other properties according to the problem type (see the next section). Thus, we distribute the molecules over the whole PES evenly to reduce the chance of missing some important areas. We then put the reactant molecules in a closed container. The molecules collide randomly. Assuming we have powerful eyes which allow us to see things microscopically, we can observe a series of events, i.e., collisions, after the reaction process starts. Collisions trigger different types of elementary reactions depending on the underlying conditions.[9] We do not care about the inter-event time, but will focus on change of energy in each of the events. With time, molecules "explore" different parts of the PES and the elementary reactions bring them toward the least energy state. Whenever a molecule assumes a new molecular structure with lower PE than those checked before, we record it. The process stops once a stopping criterion is reached. We can decide on the appropriate stopping criteria, depending on the problem type. The final solution obtained in each run of CRO is the structure, whose PE is the lowest, during the whole course of the reaction.

D. Algorithm

Here, we demonstrate the above chemical reaction idea in terms of an algorithm. There are three stages in CRO: initialization, iteration, and the final stage. The computer implements CRO by following these three stages sequentially. Fig. 5 shows its flow chart. START and END indicate the beginning and termination of each run of CRO. In each run, we start with the initialization, perform a certain number of iterations, and

[8]Intensification and diversification are two fundamental approaches to performing a search in a metaheuristic. Different metaheuristics implement these approaches differently, and hence perform differently.

[9]The underlying conditions are subject to the optimization problems. This framework does not fix the conditions, and examples of the conditions are given in Section IV-A.

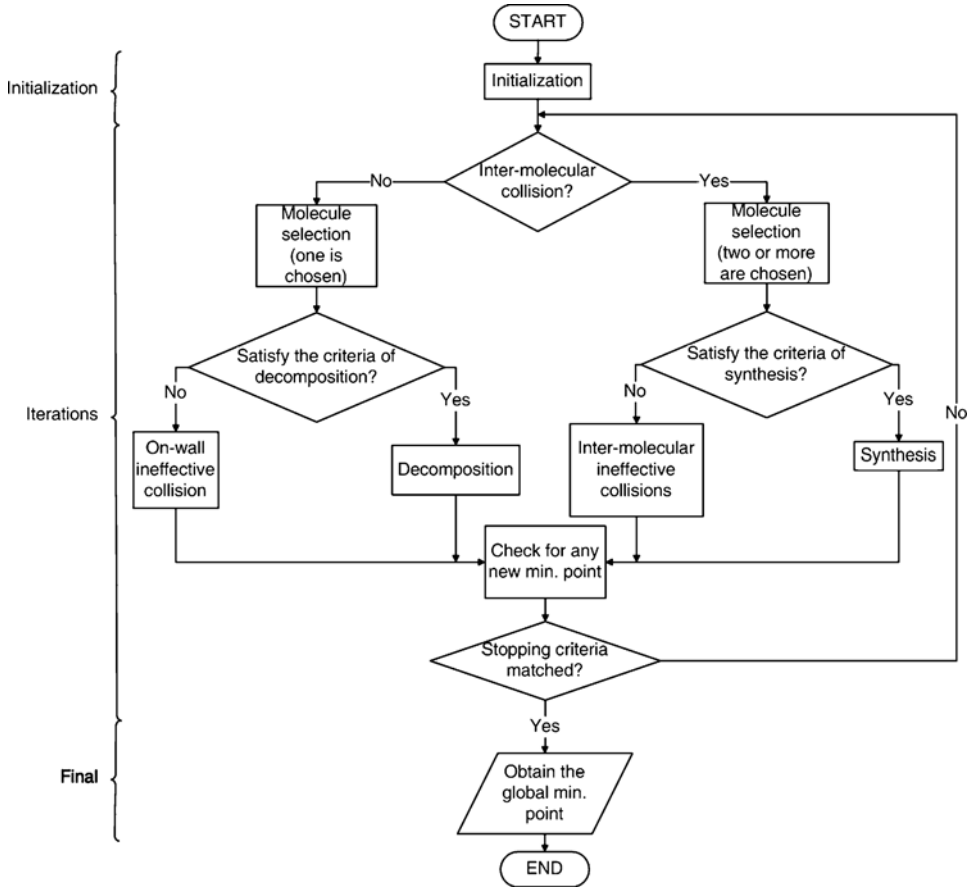


Fig. 5. Flow chart of CRO.

terminate at the final stage. In initialization, we need to define the solution space and some algorithmic functions, and assign values to several variables and control parameters. CRO is a population-based metaheuristic, but the number of solutions held in memory is subject to change, depending on the effects of decomposition and synthesis. Table I shows the symbols used in CRO. We first produce *Pop* by generating the *PopSize* number of solutions randomly in the solution space. This increases the scope of searching over *f*. In the iteration stage, a number of iterations are performed. In each iteration, we choose a collision. We first decide whether it is a unimolecular or an inter-molecular collision. To do this, we generate a random number *t*, in the interval of [0, 1]. If *t* is larger than *MoleColl*, it will result in an event of unimolecular collision. Otherwise, an inter-molecular collision will take place. (Note that we will always have unimolecular collision when there remains only one molecule in *Pop*.) Then we randomly select a suitable number of molecules from *Pop*, according to the just-decided collision type (left or right side of Fig. 5). In fact, the molecules involved in a collision highly depend on their physical locations in the container. Nevertheless, this fact does not bear any importance to the algorithm and so we ignore it for simplicity. Next, we examine the criteria of decomposition or synthesis to decide which type of collision (*left*: on-wall ineffective collision or decomposition; *right*: inter-molecular ineffective collision or synthesis) it is. After that, we check for any new minimum point found and record it. This iteration stage repeats until any one of the stopping criteria is matched. A stopping criterion may be defined based on the maximum amount of CPU time used, the maximum number of iterations performed, an objective function value less than a predefined threshold obtained, the maximum number of iterations performed without improvements or any other appropriate criteria. In the final state, we output the solution with the lowest value found over *f*. The pseudocode of CRO is as follows:

CRO(*f*, *NVars*)

Input: Problem-specific information (the objective function *f*, constraints, and the dimensions of the problem *NVars*)

1. Assign parameter values to *PopSize*, *KELossRate*, *MoleColl* and *InitialKE*
2. Let *Pop* be the set of molecule $1, 2, \ldots,$ *PopSize*
3. **for** each of the molecules **do**
4. Assign a random solution to the molecular structure ω
5. Calculate the *PE* by $f(\omega)$
6. Assign the *KE* with *InitialKE*
7. **end for**
8. Let the central energy buffer be *buffer* and assign $buffer = 0$
9. **while** the stopping criteria not met **do**
10. Get t randomly in interval [0, 1]
11. **if** $t >$ *MoleColl* **then**

```
12.      Select a molecule M from Pop randomly
13.      if decomposition criterion met then
14.        (M'_1, M'_2, Success )=decompose(M, buffer)
15.        if Success then
16.          Remove M from Pop
17.          Add M'_1 and M'_2 to Pop
18.        end if
19.    else
20.        ineff_coll_on_wall(M, buffer)
21.        end if
22.    else
23.      Select molecules M_1 and M_2 from Pop randomly
24.      if synthesis criterion met then
25.      (M', Success)=synthesis(M_1, M_2)
26.        if Success then
27.          Remove M_1 and M_2 from Pop
28.          Add M' to Pop
29.        end if
30.      else
31.        inter_ineff_coll(M_1, M_2)
32.      end if
33.    end if
34.    Check for any new minimum solution
35.  end while
36.  Output the overall minimum solution and its function
       value
```

E. Relationship to Optimization

We are going to discuss CRO from the perspective of optimization. In the initial stage, we define the *PopSize* and *InitialKE*. The total energy (*TE*) of the system is given by

$$TE = \sum_{i=1}^{PopSize} PE_{\omega_i} + KE_{\omega_i} = \sum_{i=1}^{PopSize} f(\omega_i) + InitialKE \times PopSize. \tag{10}$$

We assign random solutions to the molecules at initialization and this is tantamount to the placement of "solution seeds" evenly over the whole solution space.

In iterations, the molecules undergo a series of on-wall and inter-molecular ineffective collisions. In any one of these events, they search around their own neighborhoods for (local) minimum solutions. Suppose a molecule corresponding to solution ω attempts to change to one corresponding to a new solution ω'. The change is allowed with probability one when $\Delta PE = f(\omega) - f(\omega') \geq 0$. If $\Delta PE < 0$, the change is only allowed when *KE* is large enough to compensate ΔPE. One difference between the two ineffective collisions is that an inter-molecular ineffective collision involves two or more molecules, i.e., their *KE* can be shared among the molecules. So the possibility of a molecule having a larger value of $f(\omega')$ is higher in an inter-molecular ineffective collision than in an on-wall one. Recall that in the latter, a part of the *KE* is transferred to *buffer* (whose initial value is 0), and thus

$$TE = \sum_{i=1}^{PopSize} PE_{\omega_i} + KE_{\omega_i} + buffer.$$

When a series of on-wall ineffective collisions happen, the molecules tend to have lower *KE* and result in subsequent solutions ω with lower $f(\omega)$. The local minimums can then be reached.

When it is estimated that the solution corresponding to a molecule is a local minimum, the molecule tries to jump to a new region of the search space through decomposition. One may decide if a local minimum has been reached based on the number of collisions which have occurred without resulting in a lower function value. This is known as the decomposition criterion. To increase the scope of searching in other regions, the molecule splits into two (or more) molecules.

A molecule with too little *KE* lacks the ability to transform to a new molecule with higher function value and gets stuck in a local minimum. When two (or more) such molecules collide, synthesis takes place and results in a single molecule with a solution far removed from the original solutions. The resultant molecule can have higher *KE* due to the combination of energy from multiple molecules. It allows the exploration of a new region of the solution space.

If we observe the whole "life cycle" of a molecule, it searches a region of the solution space for a certain period and then jumps to another region to continue the search. This process can repeat since we "recycle" the excessive energy of some molecules through *buffer*. If we do not limit the searching time, CRO can explore every possible region of the solution space and eventually find the global minimum.

We can see that "energy" plays a key role in obtaining new solutions. *TE* is constant in the whole course of searching. In (10), *PE* is solely determined by the objective function *f*. We can control the size of *TE* with *InitialKE* and *PopSize*. A lower *TE* will increase the convergence rate, but this is at the expense of getting stuck in a local minimum. If *f* has a very uneven "landscape," large *InitialKE* is more favorable.

F. Characteristics

CRO is a variable population-based metaheuristic, where the total number of solutions kept simultaneously by the algorithm may change from time to time. Decomposition and synthesis increases and decreases the number of molecules in the container, respectively—decomposition breaks a molecule into two or more, whereas synthesis combines two or more molecules into one.

Inequalities (3)–(5), (8), and (9) capture the idea of the conservation of energy. The main idea is to redistribute the energy among the molecules. The on-wall ineffective collision and the inter-molecular ineffective collision give the effect of local search. We weaken the ability of molecules to escape from local minima by drawing their *KE* out to the central energy buffer (through on-wall ineffective collision). To prevent the molecules from getting stuck at local minima, decomposition and synthesis give the molecules a way to explore other regions of the PES.

The use of the central energy buffer and the concept of energy exchange are the most distinguishable features of CRO. Energy provides molecules the ability to find new solutions. The central energy buffer allows implicit cooperation among the molecules; the excessive energy of molecules accumulated

through on-wall ineffective collision can be reused in decomposition when the resultant molecules (in decomposition) do not have adequate energy. Moreover, explicit cooperation among molecules is realized through energy exchange due to inter-molecular collisions (i.e., inefficient collision and synthesis); the sum of the energy of two or more molecules is typically larger than that of a single molecule, and thus, the probability of getting a new solution in an inter-molecular collision is higher than in a unimolecular collision.

The basic unit of CRO is a molecule. The events in CRO constitute several types of elementary reactions. In each iteration, a subset of the molecules is involved. These characteristics make CRO particularly suited to be implemented in an object-oriented programming language, e.g., C++ and Java. We can create a class to define a molecule, with several methods of the class corresponding to the elementary reaction types. When a molecule is created or destroyed, we can just simply add or remove the corresponding object in the main program.

In general, CRO may be considered an optimization framework, which allows users to use their favorable heuristic components for their own optimization problem. The changeable components include the decomposition and synthesis criteria, the neighborhood structure used in on-wall and inter-molecular ineffective collisions, and the mechanisms in generating new solutions in decomposition and synthesis. In this way, CRO can be applied to a wide range of optimization problems.

Moreover, CRO can be made to manipulate multiple tanks of molecules in parallel. Suppose we have a problem which can be broken down into several modules. We implement several CRO programs corresponding to the different modules simultaneously. At certain moments, we may arrange certain CRO programs to swap their molecules or to exchange information, like the best functional value obtained. A distributed (or parallel) version of CRO can be implemented in this way and synchronization among different CRO programs is not required. Therefore, CRO is best suited to those types of problems which will benefit from parallel processing rather than sequential processing.

Readers may have questions about the similarities between CRO and SA. The differences are three-fold. First of all, the inspiration of CRO is from the "first principles" of a chemical reaction, which is different from annealing in metallurgy of SA. CRO looks at things microscopically, while SA mimics the system macroscopically. Secondly, the main concern of CRO is the redistribution of energy from *PE* to *KE* (and vice versa) among different molecules. It does not use the Metropolis condition, which is the core of SA. Thirdly, CRO is a variable population-based metaheuristic. The number of solutions maintained by the system is more than one and subject to change, depending on the problem we are solving. However, SA only keeps one solution at a time. These characteristics establish the uniqueness of CRO compared to SA, and to other metaheuristics.

IV. Simulation

We are going to show that CRO is a successful metaheuristic governed by the NFL theorem. To do this, we apply CRO to the QAP [28], the resource-constrained project scheduling problem (RCPSP) [29], [30], and the channel assignment problem (CAP) [31].

A. Quadratic Assignment Problem

QAP is very easy to state but it is one of the most difficult NP-hard combinatorial optimization problems.[10] Mathematicians and computer scientists consider instances of size larger than 20 intractable. It has been proved [32], that it is impossible to find an ε-approximation algorithm for QAP. QAP has many real-life applications [28] and we can transform it easily to other well-known combinatorial optimization problems, e.g., traveling salesman problem [33], maximum clique problem [34], and graph-partitioning problem [1].

QAP tries to minimize the total cost when assigning facilities to locations.[11] The facilities are of different types and the number of facilities and that of locations are equal. Given the distance between each pair of locations and the human flow between any two facilities,[12] we define the cost as (flow $\times$ distance), and obtain the total cost by summing the cost of any possible pairs of facilities and locations. Each type of facility must be built at a unique location. In other words, we cannot assign duplicate facilities to distinct locations and each location must have a facility assigned. In fact, this constraint makes the problem very hard to solve. Moreover, the number of possible solutions grows exponentially with the problem dimensions (i.e., n).[13] Most of its real world applications have n larger than 20, i.e., beyond the size of computational tractability, and thus, the brute force method is fruitless. Different QAP instances have different n, flow, and distance values.

We are going to define QAP mathematically and to represent it in the generic form of (1) from Section I. Consider a problem of size n. We have n facilities to be assigned to n locations. We define f_{ij} as the flow between facilities i and j, and d_{kl} as the distance between locations k and l. We write the objective function and the constraints as follows:

$$\min \sum_{i,j=1}^{n} \sum_{k,l=1}^{n} f_{ij} d_{kl} x_{ik} x_{jl} \tag{11}$$

$$\begin{aligned} \text{subject to} \quad & \textstyle\sum_{i=1}^{n} x_{ij} = 1, && 1 \le j \le n \\ & \textstyle\sum_{j=1}^{n} x_{ij} = 1, && 1 \le i \le n \\ & x_{ij} \in \{0, 1\}, && 1 \le i, j \le n. \end{aligned}$$

[10]Combinatorial optimization problem is a category of optimization problems whose solution space is discrete, usually large, and limited. It is important as it represents the whole class of optimization utilizing the computer. Nowadays, most optimization problems are solved with the computer, and the solution space becomes discrete and limited when we describe and analyze it in the computer.

[11]We use the terms locations and facilities for ease of explanation. We can replace them with any other meaningful terms to suit our purposes.

[12]Human flow from facility i to facility j means the rate of people (e.g., the number of people per hour) moving from facility i to facility j. The human flow in the opposite direction, i.e., from facility j to facility i, can be different. If they are identical, the flow matrix is symmetric.

[13]The size of the solution space is equal to $n!$. When n equals five, there are 5! = 120 possible solutions. However, when n increases four times to 20, it expands to roughly 2.43×10^{18}.

TABLE II
VALUES ASSIGNED TO THE CONTROL PARAMETERS IN THE SIMULATIONS

Parameter	Value		
	QAP	RSPSP	CAP
PopSize	25	10	10
KELossRate	0.8	0.5	0.2
MoleColl	0.2	0.2	0.2
InitialKE	1 000 000	10 000	800
α^a	1300	200	300
β^b	10 000	100	10

[a]α and [b]β are thresholds defined for the conditions for decomposition and synthesis, respectively.

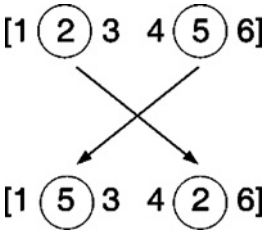


Fig. 6. Example of neighbors in the two-exchange neighborhood structure.

If we examine the constraints more carefully, we find that every possible solution is in the form of a permutation of n elements. The positions and the values of the permutation correspond to the locations and the facilities, respectively. Then we can determine the objective function value by summing the products of flow and distance of every possible pair in the permutation. Consider an instance of the problem with n equal to four. One possible solution is [2, 4, 3, 1], i.e., we assign facility 2 to location 1, facility 4 to location 2, etc. We evaluate this solution by computing

$$\begin{aligned} &f_{22}d_{11} + f_{24}d_{12} + f_{23}d_{13} + f_{21}d_{14} \\ &+f_{42}d_{21} + f_{44}d_{22} + f_{43}d_{23} + f_{41}d_{24} \\ &+f_{32}d_{31} + f_{34}d_{32} + f_{33}d_{33} + f_{31}d_{34} \\ &+f_{12}d_{41} + f_{14}d_{42} + f_{13}d_{43} + f_{11}d_{44}. \end{aligned} \tag{12}$$

1) *Implementation Details:* This section is dedicated to explaining how to apply CRO to solve QAP. In Section III, the general design framework has been illustrated. To build the actual program codes from the framework, we need to specify the implementation details which are suitable for QAP.

The second column of Table II shows the values assigned to the control parameters of CRO used in the simulation. A possible solution is in the form of a permutation of the problem dimensions. Fig. 6 shows two possible solutions when the problem dimension is equal to six. The first solution means that we assign the first facility to location 1, the second facility to location 2 and so on. (Note that a solution can have a different form if we apply CRO to other problems.) We adopt the two-exchange neighborhood structure (Fig. 6) since there is no natural neighborhood structure defined for permutations, compared with continuous functions. The second solution gives a neighbor of the first. In this representation, we assign the fifth facility to location 2 and the second facility to location 5.

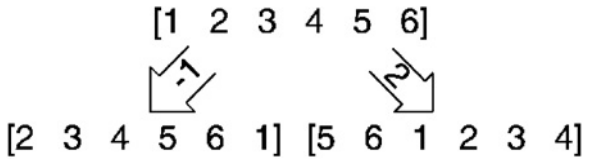


Fig. 7. Two examples of the circular shift operator.

This neighborhood structure is used in the implementations of on-wall ineffective collision and inter-molecular ineffective collision.

We are going to explain the last four attributes in Fig. 3 which have not been discussed in Section III-A. The number of hits indicates how many times the molecule has been involved in a collision of any kind. Equivalently, it tells us how many solutions the molecule has carried. The minimum structure and the minimum value mean the best molecular structure and the corresponding functional value that the molecule has experienced, respectively. The minimum hit number records when the molecule obtains its latest minimum structure, in terms of the number of hits. For example, if the molecule gets its latest minimum molecular structure in its tenth collision, then the minimum hit number is set to 10. The number does not change provided that the molecule does not experience any new minimum molecular structure in any subsequent collision. Naturally, when a molecule is just created, the minimum structure and the minimum value are equal to its initial molecular structure and *PE*, respectively. Both the number of hits and minimum hit number are set to zero.

In decomposition, we obtain two solutions ω_1' and ω_2' from ω. We adopt the circular shift operator (Fig. 7) to generate new solutions. We obtain a new solution by generating an integer[14] in the range of $[-n, n]$, where n is the size of the permutation, indicating how many steps we need to shift from the original one. Negative and positive values mean shifting to the left and right, respectively. In Fig. 7, we get the left permutation by shifting to the left for one step, and the right one by shifting to the right for two steps. On the left-hand side of Fig. 5, the decomposition criterion determines whether a unimolecular collision is a decomposition or an on-wall ineffective collision. Here, we specify the decomposition criterion to test if the selected molecule has stayed in a stable state for a certain period of time, i.e., (number of hits − minimum hit number) > α (Fig. 3 and Table II). It means that it has not moved to a lower energy state for certain time in terms of number of hits α.

In synthesis, we attempt to generate a new molecule with solution ω' from two existing molecules with solutions, ω_1 and ω_2. We adopt the distance-preserving crossover operator [35] to the minimum structures of the two existing molecules to get ω'. On the right-hand side of Fig. 5, the synthesis criterion determines whether an inter-molecular collision is a synthesis or an inter-molecular ineffective collision. Here, we specify the synthesis criterion to test if both molecules, at the same time,

[14]Here, we generate two random integers independently. Then we apply circular shift with the two integers separately to obtain the independent solutions.

TABLE III
IMPLEMENTATION DETAILS AND SETTINGS USED IN THE SIMULATIONS

Implementation Details	Settings		
	QAP	RCPSP	CAP
Solution space	Permutation set	Permutation set	Integer vector
Parameter settings	Second column of Table II	Third column of Table II	Fourth column of Table II
Neighborhood structure	Two-exchange neighborhood structure	Same as QAP	One-difference neighborhood structure
Decomposition criterion	(number of hits–minimum hit number) $>\alpha$	Same as QAP	Same as QAP
Decomposition mechanism	Circular shift	Same as QAP	Half-total-change
Synthesis criterion	$KE_{\omega 1} < \beta$ and $KE_{\omega 2} < \beta$	Same as QAP	Same as QAP
Synthesis mechanism	Distance-preserving crossover	Same as QAP	Probabilistic select

have insufficient *KE*, i.e., $KE_{\omega_1} \leq \beta$ and $KE_{\omega_2} \leq \beta$, where β (Table II) defines the least amount of *KE* a molecule should possess. When a molecule has too little *KE*, it potentially loses its ability to escape from a local minimum. When two such molecules collide, they combine and form a new molecule whose *KE* becomes large again. Then the resultant molecule can continue to explore different parts of the PES with the ability to avoid getting stuck in a local minimum. We give a recapitulation of the above mentioned implementation details in the second column of Table III. When applying CRO to other problems, the implementation details are subject to change in order to have better performance. In other words, the implementation details are problem-dependent.

We summarize the simulated versions of these four elementary reactions in terms of intensification and diversification in Table IV. More ticks indicate stronger effects. Note that the distance-preserving crossover operator used in synthesis has some effect of intensification. For QAP, we can replace the circular shift and distance-preserving crossover mechanisms with other suitable ones. The extent of intensification and diversification may vary, depending on how we implement the mechanisms in the simulation.

The values of the parameters, neighborhood structure, decomposition and synthesis criteria, and the mechanisms used in decomposition and synthesis to generate new molecule(s) can be changed and tuned to match the nature of the problem. To the best of our knowledge, there are no theoretical guidelines on how to tune them and to match them with the problem, as is also true in other metaheuristics. The adjustments rely on the experience and preferences of researchers [36].

2) *Results:* We tested the effectiveness of CRO with 23 problem instances (Table V) selected from a QAP digital library [37] and compare the results with an oracle-based view of computation [24], i.e., finding the best solution within a certain number of function evaluations. Thus, the stopping criterion is when a certain number of function evaluations are reached. We do not adopt the computational time as the performance metric, simply because it highly depends on how we program the simulation codes.[15] We compare the results of CRO with those of three other famous metaheuristics for solving QAP, i.e., fast ant system (FANT) [38], an improved simulated annealing (ISA) [17], and tabu search (TABU) [39].[16] They are among the best and available algorithms to solve QAP. All simulation codes are programmed in C++ and tested in Windows environment. We ran all the simulations on the same computer, with configuration of Intel Pentium 4 3.2 GHz and 512 MB dual RAM. This removes any machine-dependent variations and provides more objective comparisons. Table V shows the simulation results of CRO and the other three algorithms. As in [40], in Table VI, we also give the result of *t*-test with 95% confidence level comparing the means for CRO and those of the other algorithms in solving the 23 problem instances. "s+," "s−," and "≈" indicate that CRO is significantly better, significantly worse, and comparable in performance to the counterpart, respectively. All the results of CRO are generated with the same implementation details explained in the previous section. We adopt the same stopping criterion, that the function evaluation limit is set to 150 000, for all the test cases, except tai10b, tai12b, and tai15b.[17] We obtained the statistics by repeating the simulation run 50 times. They are the "minimum," "maximum," "mean," and "StdDev" of each set of data and represent the best case, and the worst case, the average and the standard deviation, respectively, in the 50 runs. The results from different runs vary as they employ randomization in the calculation. Values in brackets mean it is the best among all the metaheuristics.

TABLE IV
INTENSIFICATION AND DIVERSIFICATION OF THE FOUR ELEMENTARY REACTIONS

Elementary Reactions	Intensification	Diversification
On-wall ineffective collision	√√	√
Decomposition		√√
Inter-molecular collision	√√	√
Synthesis	√	√√

[15]Computational times for a run of simulation on the test instances are also provided in Table V as reference.

[16]Their original codes can be found on the Internet (http://mistic.heig-vd.ch/taillard/). We modified their original codes to meet the stopping criterion.

[17]150 000 function evaluation limit is too large when compared with their solution spaces. If we still use 150 000, many parts of the solution spaces are explored by the algorithms, and thus, the power of the algorithms cannot be revealed.

TABLE V
SIMULATION RESULTS OF CRO ON SOLVING QAP COMPARED WITH THE OTHER THREE METAHEURISTICS

Instance	Problem Size	Global Min.	Function Evaluation Limit	FANT				ISA			
				Min.	Max.	Mean	StdDev	Min.	Max.	Mean	StdDev
nug21	21	2438	150 000	(2438)	2464	2444.44	8.03	(2438)	2462	2445.72	9.42
nug22	22	3596	150 000	(3596)	3632	3599.80	7.39	(3596)	3644	3607.84	17.88
nug24	24	3488	150 000	(3488)	3546	3500.40	13.69	(3488)	(3526)	3498.40	13.07
nug25	25	3744	150 000	(3744)	3772	3750.36	7.50	(3744)	3768	(3746.96)	5.36
nug27	27	5234	150 000	(5234)	5324	(5249.52)	24.87	(5234)	5314	5259.04	31.47
nug28	28	5166	150 000	(5166)	5266	5203.24	24.99	(5166)	5278	(5201.28)	30.17
nug30	30	6124	150 000	6128	6210	6158.56	23.00	(6124)	6214	(6146.96)	22.08
kra30a	30	88 900	150 000	(88 900)	93 200	(90 601.80)	788.29	90 160	94 340	91 644.20	1029.06
kra30b	30	91 420	150 000	(91 420)	93 010	92 031.00	425.24	91 590	94 990	92 752	899.41
kra32	32	88 700	150 000	(88 700)	91 490	90 373.80	746.27	(88 700)	93 060	90 664.60	1085.55
tai10b	10	1 183 760	50 000	(1 183 760)	(1 183 760)	(1 183 760.00)	(0)	(1 183 760)	1 213 671	1 184 925.80	4938.19
tai12b	12	39 464 925	50 000	(39 464 925)	(39 464 925)	(39 464 925.00)	(0)	(39 464 925)	45 097 713	41 801 386.02	1 674 363
tai15b	15	51 765 268	50 000	(51 765 268)	(51 855 477)	(51 744 385.84)	(25 041.2)	(51 765 268)	50 035 184	51 814 064.70	59 535.52
esc32a	32	130	150 000	(130)	146	138.60	3.21	134	150	140.60	3.16
esc32b	32	168	150 000	(168)	(192)	178.88	9.70	188	224	208.96	(7.94)
esc32c	32	642	150 000	(642)	(642)	(642.00)	(0)	(642)	(642)	(642.00)	(0)
esc32d	32	200	150 000	(200)	(200)	(200.00)	(0)	(200)	208	202.44	1.82
esc32e	32	2	150 000	(2)	(2)	(2.00)	(0)	(2)	(2)	(2.00)	(0)
esc32g	32	6	150 000	(6)	(6)	(6.00)	(0)	(6)	(6)	(6.00)	(0)
esc32h	32	438	150 000	(438)	440	438.12	0.48	(438)	442	439.80	1.30
tai64c	64	1 855 928	150 000	(1 855 928)	(1 857 646)	(1 856 255.96)	664.31	(1 855 928)	1 857 660	1 859 213.60	(4081.4)
wil50	50	48 816	150 000	48 964	49 254	49 098.72	71.41	(48 844)	49 296	(48 937.12)	(96.30)
wil100	100	273 038	150 000	274 800	(275 980)	275 436.48	(279.92)	(273 816)	276 034	(274 683.32)	525.24

Instance	TABU				CRO				
	Min.	Max.	Mean	StdDev	Min.	Max.	Mean	StdDev	Computational time (s)
nug21	(2438)	2484	2452.28	10.63	(2438)	(2456)	(2443.64)	(5.39)	1.046
nug22	(3596)	3696	3618.92	22.76	(3596)	(3606)	(3597.80)	(2.78)	1.109
nug24	(3488)	3554	3503.20	15.74	(3488)	(3526)	(3494.88)	(10.21)	1.265
nug25	(3744)	3788	3751.76	10.84	(3744)	(3760)	3749.68	(4.19)	1.343
nug27	(5234)	5382	5285.56	40.33	(5234)	(5298)	5259.36	(18.92)	1.500
nug28	(5166)	5282	5219.76	35.68	(5166)	(5238)	5202.52	(18.24)	1.610
nug30	(6124)	6234	6175.28	25.93	6128	(6206)	6170.12	(19.48)	1.781
kra30a	(88 900)	95 280	92 428.40	1638.10	(88 900)	(61 800)	90 664.20	(670.28)	1.797
kra30b	91 490	96 050	93 029.60	1183.56	91 490	(92 840)	(92 022.80)	(332.98)	1.797
kra32	(88 700)	94 430	91 714.60	1344.64	(88 700)	(91 260)	(90 190.80)	(635.02)	1.984
tai10b	(1 183 760)	(1 183 760)	(1 183 760.00)	(0)	(1 183 760)	1 187 126	1 184 029.28	922.44	0.407
tai12b	(39 464 925)	40 063 583	39 526 972.08	172 115.2	(39 464 925)	40 063 573	39 511 175.94	139 496.6	0.485
tai15b	(51 765 268)	51 944 836	51 822 408.02	58 390.18	(51 765 268)	52 205 386	52 035 537.10	88 128.42	0.640
esc32a	134	162	145.80	6.95	(130)	(142)	(136.84)	(2.68)	2.000
esc32b	(168)	224	196.32	10.72	(168)	(192)	(175.36)	8.53	1.985
esc32c	(642)	646	642.24	0.96	(642)	(642)	(642.00)	(0)	2.016
esc32d	(200)	216	205.32	5.07	(200)	(200)	(200.00)	(0)	2.015
esc32e	(2)	(2)	(2.00)	(0)	(2)	(2)	(2.00)	(0)	2.031
esc32g	(6)	(6)	(6.00)	(0)	(6)	(6)	(6.00)	(0)	2.016
esc32h	440	478	453.00	12.72	(438)	(438)	(438.00)	(0)	2.000
tai64c	(1 855 928)	1 883 516	1 863 245.04	6081.09	(1 855 928)	1 860 348	1 856 796.04	1004.97	6.984
wil50	48 996	49 828	49 343.88	232.61	48 918	(49 214)	49 071.12	68.26	4.407
wil100	280 634	283 190	281 779.56	596.49	274 618	276 278	275 291.16	345.02	15.985

In order to understand the performance of CRO during a simulation run, we examine the case of the instance kra32[18] in detail. Each run of simulations is terminated once the number of function evaluations reaches 150 000, where we examine roughly a 5.70×10^{-31} (=150 000/32!) fraction of the whole solution space only, in each run of simulation. Although all metaheuristics can obtain the global minimum, CRO outperforms all the others, in terms of the mean and maximum costs obtained. For further examination, we plot the results versus the number of function evaluations in the period of a simulation run in Fig. 8. We record the best result after each interval of 2500 function evaluations. There are 61 (=1 + 150 000/2500) data points in each run, including the one before each simulation run starts. Assume each function evaluation takes roughly equal amount of time, for all the plots above, each data point represents the statistical result of 50 runs at a time instance. Fig. 8(a)–(c) show the plots of mean cost, maximum cost, and minimum cost, respectively. For easier observation, Fig. 8(d)–(f) give the zoomed portions of the graphs on their left, corresponding to the dotted-line

[18]kra32 has problem size of 32. It defines the distance and flow information with real-world data used to plan Klinikum Regensburg in Germany. The solution space is of the order of 10^{35} (=32!), and its global minimal solution has function value equal to 88 700.

TABLE VI
RESULT OF T-TEST WITH 95% CONFIDENCE LEVEL COMPARING THE MEANS FOR CRO AND THE OTHER ALGORITHMS IN SOLVING THE QAP INSTANCES

Instant	FANT	ISA	TABU
nug21	≈	≈	s+
nug22	s+	s+	s+
nug24	s+	≈	s+
nug25	≈	s−	≈
nug27	s−	≈	s+
nug28	≈	≈	s+
nug30	s−	s−	≈
kra30a	≈	s+	s+
kra30b	≈	s+	s+
kra32	≈	s+	s+
tai10b	s−	≈	s−
tai12b	s−	s+	≈
tai15b	s−	s−	s−
esc32a	s+	s+	s+
esc32b	s+	s+	s+
esc32c	≈	≈	s+
esc32d	≈	s+	s+
esc32e	≈	≈	≈
esc32g	≈	≈	≈
esc32h	s+	s+	s+
tai64c	s−	s+	s+
wil50	s+	s−	s+
wil100	s+	s−	s+

boxes. They show CRO obtain better or equally good results eventually. For Fig. 8(b), (c), (e), and (f), points on the same curve do not necessarily belong to the same run and they are the results of the whole 50 runs.

To understand how the parameter values are set as in Table II, we do an analysis on the parameters. Let the settings in Table II be the basis. For each of the parameters investigated, we replace the parameter value with some other reasonable ones and fix the rest with the values as in the basis. We repeat the simulations on kra32 for 50 times. The performance is checked with the mean of the best total costs at the ends of the 50 runs. As shown in Fig. 9, the parameter settings in Table II (except for β) are the best. In Fig. 9(e), the last two data points for β appear to be an anomaly as the total cost increases and then decreases as β changes from 1.2×10^4 to 1.6×10^4. This is because CRO is a stochastic metaheuristic, and the results may vary in different runs of the simulation, especially when the number of runs is not sufficiently large.

As shown in Table V, CRO can outperform the other three metaheuristics in many of the test instances. For esc32c, esc32d, esc32e, esc32g, and esc32h, CRO can even find the global minimum in all the 50 runs. Moreover, CRO can generate the minimum standard deviations among all the metaheuristics in most of the problem instances. This implies that CRO can achieve the most stable results. From the website of the QAP digital library [37], most of the test cases in which CRO works well are real-life instances, whose flow and distance data are from real-life examples. Their common characteristic is that the fitness landscapes formed from the problems are in a certain pattern, unlike the randomly generated instances whose fitness landscapes are unstructured. For problems other than QAP, there are also real-life instances as well as randomly generated instances. By applying the phenomenon shown in the QAP, CRO is expected to have better performance for the real-life instances.

We note that CRO is already at a disadvantage in this comparison, because the chosen competitors (FANT, ISA, and TABU) are already the QAP-adapted versions of their original metaheuristics. We believe that if we can get the research community interested in this proposed approach, improved versions of CRO will soon be available. Nevertheless, we are confident that CRO is one of the successful metaheuristics which can be used to solve optimization problems in general.

B. Resource-Constrained Project Scheduling Problem

Similar to QAP, RCPSP is one of the classical NP-hard optimization problems. It "is one of the most intractable problems in Operations Research" [41]. Job-shop, flow-shop, and open-shop problems [42] are its special cases. Comprehensive surveys about the heuristics on solving RCPSP can be found in [43] and [44].

Consider that we have a project consisting of a set of n activities, $S = \{1, 2, \ldots, n\}$, where n indicates the dimensions of the problem. There are a set of (renewable) resources K and each resource $k \in K$ has its own capacity limit R_k. For each activity $j \in S$, there are a set of immediate predecessor activities P_j, and it has a non-preemptable duration d_j. We denote the finish time of activity j as f_j (and thus the start time is $f_j - d_j$). When activity j is being processed, it consumes $r_{j,k}$ units of resource k. Let $A(t) = \{j \in S | f_j - d_j \leq t < f_j\}$ be the set of activities being processed at time t. We add two dummy activities (indicated as activity 0 and activity $n + 1$) with zero duration and zero resource consumption to S and they represent the "project start" and "project end," respectively. They must be the first and the final activities to be scheduled accordingly. For the rest of the activities, an activity j can only be scheduled starting at time t provided that all its immediate predecessors P_j have been finished at or before t, and that there are adequate resources between time t and $t + d_j$. The makespan is the time we spend on the project (i.e., the finish time of activity "project end"). All variables shown above take non-negative integer values. RCPSP tries to minimize the makespan of the schedule of a project subject to the precedence constraints and the resource constraints. Mathematically, in the generic form of (1) from Section I, we can formulate RCPSP as follows:

$$\min f_{n+1} \tag{13}$$

$$\text{subject to} \quad \begin{aligned} & f_h \leq f_j - d_j && 1 \leq j \leq n+1; h \in P_j, \\ & \textstyle\sum_{j \in A(t)} r_{j,k} \leq R_k && k \in K; t \geq 0, \\ & f_j \geq 0 && 0 \leq j \leq n+1. \end{aligned}$$

1) *Implementation Details:* Each solution of RCPSP can be represented in the form of permutation, i.e., activity list [42]. Consider we have a project of four activities, with activity 0 and activity 5 as the "project start" and "project end." A solution [0, 3, 1, 4, 2, 5] means that activity 3 has higher priority to be scheduled than activity 1, 4 and 2.[19] We use the

[19]Activity 0 and 5 must be in the first and the last place in the solution because they indicate the "project start" and "project end."

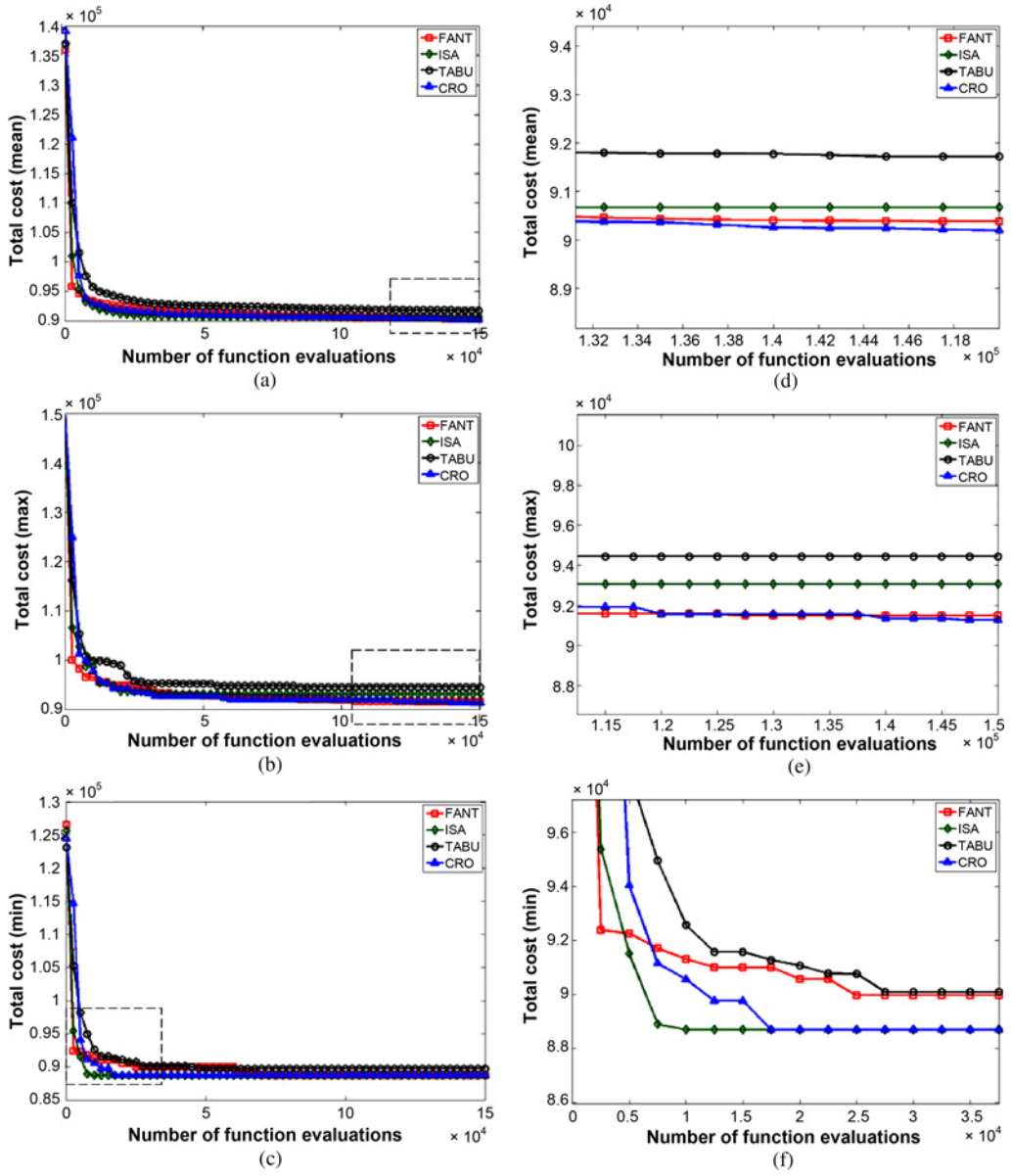


Fig. 8. Plots versus the duration of a simulation run. (a) Mean cost. (b) Maximum cost. (c) Minimum cost. (d) Zoomed portions of (a). (e) Zoomed portions of (b). (f) Zoomed portions of (c).

following serial schedule generation scheme [42] to generate the finish times of all activities and to get the objective function value, i.e., *makespan*.

For fast implementation, we adopt the same algorithmic details of CRO for QAP to solve RCPSP, since both problems have the same solution structure, i.e., a permutation of n numbers. The only difference is in the parameter values used, as shown in the third column of Table II.

2) *Results:* We test the performance of CRO with the benchmark instance set *j120* from a digital library PSPLIB [45]. *j120* contains 600 problem instances and each has 120 activities for scheduling, excluding the dummy activities. We do the simulation for the whole set of instances once and, for each instance, we end the iterations when 60 000 function evaluations have been reached, i.e., 60 000 calls to the above schedule generation scheme.

As shown in the official website of PSPLIB, the current best results generated by heuristics (and metaheuristics) are reported. We tabulate the authors, the reported year and the number of first best *makespan* found in Table VII. The data are sorted in descending order of the number of instances. We also include the number of instances found by CRO that correspond to the currently best makespan. As indicated, CRO can obtain the best makespans of 116 instances. Note that we adopt the same neighborhood structure, decomposition and synthesis criteria, and decomposition and synthesis mechanisms used for QAP directly for this problem. Moreover, we do not employ any RCPSP-favorable heuristic components [44] into CRO.[20] This implies that CRO can indeed solve complex optimization problems like RCPSP.

[20]We leave the tasks of implementing RCPSP-favorable heuristic components into CRO for future work.

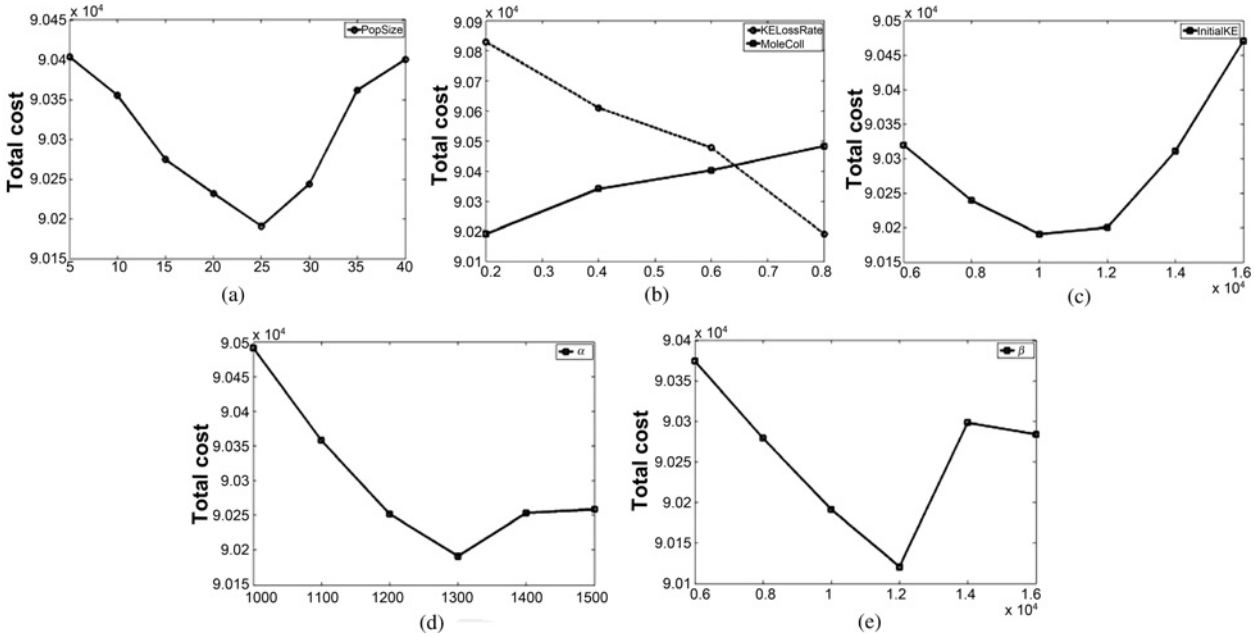


Fig. 9. Performance on kra32 versus variations of the control parameters. (a) PopSize. (b) KELossRate and MoleColl. (c) InitialKE. (d) α. (e) β.

Serial_schedule_generation(ω)

Input: Activity list ω
1. Create a set *Unscheduled* = $\{1, 2, \ldots, n+1\}$
2. Create a set *Scheduled*=$\{0\}$
3. Create an empty set *Eligible* = $\{\phi\}$
4. Set $f_0 = 0$
5. **while** *Unscheduled* is not empty **do**
6. Check *Unscheduled* to see which activities are eligible for scheduling according to the precedence constraints and put them into *Eligible*
7. Select the highest priority activity j in *Eligible* according to ω
8. Remove j from *Eligible* and put it into *Scheduled*
9. Determine the maximum finish time $T_{\max}$ of all activities in P_j
10. For any time $t \geq T_{\max}$, determine the minimum $t_{\min}$ such that for each $k \in K$, available resource $\geq r_{j,k}$ in the period $[t_{\min}, t_{\min} + d_j)$
11. Set $f_j = t_{\min} + d_j$
12. **end while**
13. Set *makespan* = f_{n+1}
14. **Output** *makespan*

TABLE VII
NUMBER OF INSTANCES WITH BEST SOLUTION FOUND REPORTED IN PSPLIB

Authors	Year	Number of Instances With Best Makespan Found
D. Sun, S. Zhou	2007	129
K. Bouleimen	1997	120
Authors of this paper	2009	116
V. Valls, M. Quintanilla, F. Ballestin	2002	73
K. Nonobe and T. Ibaraki	1998	69
F. Xiao, A. Lim and B. Rodrigues	2005	57
T. Baar, P. Brucker, S.Knust	1997	42
D. Debels and M. Vanhoucke	2005	36
M. R. Ranjbar, F. Kianfar	2007	24
V. Valls, M. Quintanilla, F. Ballest	1999	14
Jose Fernando Goncalves & Jorge Magalhae	2006	11
D. Merkle, M. Middendorf, H. Schmeck	2000	5
J. Horstmann and J. Homberger	2005	5
A. Horbach	2008	4
M. Palpant, C. Antigues, P. Michelon	2002	4
D. Sun	2007	2
J. Alcaraz and C. Maroto	1998	2
K. Fleszar, K. S. Chen	2002	1
L.Y. Tseng and S.C. Chen	2004	1
P. Laborie	2005	1

C. Channel Assignment Problem

After testing CRO with two classic optimization problems, we investigate the performance of CRO in a practical channel assignment problem in multiradio wireless mesh networks [31].

We consider a set of stationary wireless routers N in a wireless mesh network. Each router $i \in N$ is equipped with a number of radio interfaces R_i. There are a set of radio channels $K = \{1, 2, \ldots, |K|\}$. A router needs to assign a channel to its interface in order to communicate with other routers. A router with multiple interfaces can assign multiple channels at the same time. It can transmit radio signals for a certain distance, and thus, it has a transmission range and an interference range. (For simplicity, we assume that the values of the two ranges are the same.) For two routers to communicate with each other, each of them must assign the identical channel to one of its interfaces and they must be within the transmission range of each other. From this, we can produce a communication graph, which is an undirected graph with nodes representing the routers. An edge (i.e., communication link) between two nodes means that they are within each other's communication range.

For $i, j, a, b \in N$, a communication link (i, j) can interfere another communication link (a, b) if i or j is within the interference range of a or b, and vice versa. Then we can produce the corresponding conflict graph $G_c(V_c, E_c)$. V_c is the set of vertices and each vertex represents a communication link in the communication graph. E_c is the set of edges and, for $u,v \in V_c$, an edge (u, v) means that u and v interfere each other.

Consider a channel assignment as a function $f : V_c \rightarrow K$, i.e., assigning each link to exactly one channel in the channel set K. We define $E(i) \subseteq V_c$ as the communication links incident on router i. Since a router i cannot be assigned a number of channels larger than R_i, we have the interface constraint. We also define network interference $I(f) = |\{(u, v) \in E_c | f(u) = f(v)\}|$. The objective of CAP is to minimize the overall network inference subject to the interface constraint. Mathematically, we have

$$\min I(f)$$
$$\text{subject to } |\{k | f(e) = k \text{ for some } e \in E(i)\}| \leq R_i \quad (14)$$

where f is a solution to the problem. For the detailed formulation, interested readers may refer to [31]. This formulation of CAP for multiradio wireless mesh network is firstly proposed by the authors of [31] and it is proved to be NP-hard.

1) *Implementation Details:* In [31], the authors propose a centralized tabu-based algorithm for the above defined CAP. The algorithm consists of two phases. The first phase employs a TS technique to find a channel assignment function f without considering the interface constraint. In the second phase, the interface constraint violation is removed and a feasible f is obtained. Our goal is to compare the performance of various metaheuristics. For simplicity, we replace the first phase (i.e., the TS component) with CRO.[21] Since this tabu-based algorithm is the only algorithm proposed to solve this problem, we can only compare the performance of CRO with TS.

Hereafter, we focus on the CRO component in the first phase of the algorithm. In general, the implementation details of this version of CRO are more or less the same as that for QAP. The fourth column of Table II shows the parameter values used. A possible solution is in the form of an integer vector with length equal to the number of vertices $|V_c|$ in the conflict graph generated from the problem. Each element of a solution can be any integer from one to the number of channels specified by the problem. For example, if there are four vertices in the conflict graph and three channels available, [2, 1, 1, 3] is a possible solution and it means that we assign channel 2 to the first vertex, channel 1 to the second vertex and so on. Solution ω' is a neighbor of solution ω if they are assigned the same channels for all vertices except one. For example, [2, 1, 2, 3] is a neighbor of [2, 1, 1, 3], but [2, 1, 3, 1] is not. We name this structure as the "one-difference" neighborhood structure.

The decomposition and synthesis criteria are the same as those for QAP. In decomposition, we obtain two solutions ω'_1 and ω'_2 from ω. The two new solutions are obtained in the same way: randomly select $|V_c|/2$ number of vertices in ω and assign a different channel to each of them. We name this decomposition mechanism as "half-total-change."

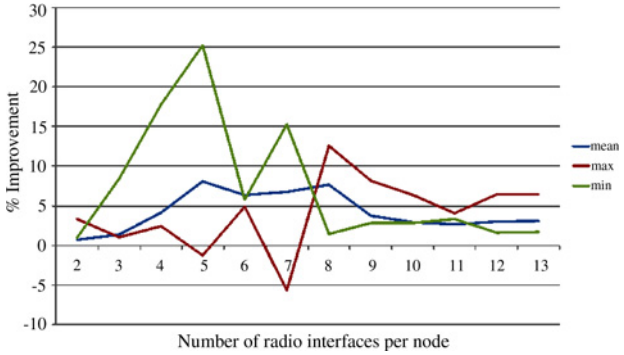


Fig. 10. % Improvement of fractional network interference of CRO-based over tabu-based method.

In synthesis, we get a new solution ω' from two solutions, ω_1 and ω_2. We generate ω' as follows: for each vertex in ω', we generate a random number $\xi \in [0, 1]$. If $\xi > 0.5$, we assign the channel number at that vertex from ω_1. Otherwise, we take the channel number at that vertex from ω_2. We name this synthesis mechanism as "probabilistic select." We summarize the above implementation details in the fourth column of Table III.[22]

2) *Results:* As there is no benchmark problem library for this CAP, we generate a problem instance ourselves.[23] We first define a square with each side equal to 500 units. We generate the x and y-coordinates of 50 nodes randomly in the square; each coordinate can be any real number in [0, 500]. For each node, the transmission and interference ranges are set to 150 units. All nodes have equal number of radio interfaces R and we do the simulation with various values of R ranging from 2 to 13. There are a total of 12 channels available. We can then produce the corresponding conflict graph, which has 280 vertices. Thus, a solution is a vector with 280 elements and each element can take a value from 1 to 12.

In each simulation run, the iterations terminate when the number of function evaluations reaches 150 000 (same as QAP). As in [31], the performance of each run is evaluated in terms of the fractional network interference, which is the ratio of network interference $I(f)$ and the total number of edges in the conflict graph. Smaller fractional network interference indicates that the assignment can result in less interference. For each value of R, we repeat the simulation 50 times. In Table VIII, we give the statistical results of the 50 runs in terms of mean, standard deviation, maximum and minimum, where mean, maximum and minimum are shown as percentages. As in Table VI, we also show the t-test result with 95% confidence level comparing the means for CRO and TS. We can see that CRO is significantly better than TS for all cases. Fig. 10 shows the percentage improvement of CRO over TS. The percentage improvement is calculated by

[21] We can definitely propose a completely CRO-based algorithm for CAP, without taking the second phase to get a feasible f. However, we are not purposely building an improved algorithm to solve CAP, but comparing with other metaheuristics. This can be reserved for future work.

[22] We use different decomposition and synthesis mechanisms because the solution structures in QAP and CAP are different.

[23] We follow the settings in [31] to generate the problem instances.

TABLE VIII
FRACTIONAL NETWORK INTERFERENCE OF CRO

	Number of Radio Interfaces Per Node											
	2	3	4	5	6	7	8	9	10	11	12	13
Mean (%)	47.98	30.94	21.28	14.04	7.92	5.10	3.98	3.76	3.74	3.74	3.74	3.73
StdDev	0.62	1.06	1.56	1.85	0.84	0.52	0.22	0.09	0.08	0.08	0.06	0.06
Maximum (%)	49.54	33.33	23.59	18.25	9.95	6.91	4.96	4.09	3.87	3.87	3.84	3.88
Minimum (%)	46.66	27.67	17.55	9.55	6.50	3.96	3.62	3.59	3.54	3.54	3.61	3.59
t-test result	s+	s+	s+	s+	s+	s+	s+	s+	s+	s+	s+	s+

$$\frac{\text{Result}_{\text{TS}} - \text{Result}_{\text{CRO}}}{\text{Result}_{\text{CRO}}} \times 100\%.$$

A positive value means CRO outperforms TS. The larger the value, the lower interference CRO can get relative to that from TS. For the maximum (the worst result), CRO gives worse performance only when R equals 5 and 7. For the rest of the cases, CRO gives better performance. From R equal to 4 to 8, the average improvement of CRO is above 5%. In particular, when there are five radio interfaces per node, CRO can improve the minimum (best) result of TS by 25%.

V. CONCLUSION

CRO is a chemical-reaction-inspired metaheuristic. It mimics the interactions of molecules in a chemical reaction, and guides the transformation of molecules along the PES toward the more stable state by redistributing the energies among molecules and by interchanging the energies from one form to another. It tries to locate the global minima of the objective function. It is an inter-disciplinary research problem, in which we exploit the characteristics of chemical reactions in solving optimization problems. As shown in the simulation section, we can apply it to solve well-known NP-hard problems, QAP and RCPSP, and a practical NP-hard problem CAP. We show that CRO is effective in solving optimization problems, and among the few successful metaheuristics. With the NFL theorem, CRO must have equal performance as the others on the average but it can outperform all other metaheuristics when matched to the right problem type. As the "spectrum" of optimization problems is huge, we will never be satisfied with the small number of generally acknowledged methods, such as SA, GA and ACO. As no single metaheuristic can work equally well over a wide range of problems, the large number of optimization problems and the small number of satisfactory metaheuristics mean that we need more metaheuristics to fill the gap. CRO provides a new approach for solving optimization problems and belongs to the few acknowledged successful metaheuristics.

We decide to demonstrate the fundamental version of CRO in this paper. In the future, we can develop even better versions of CRO through hybridation with the others (e.g., CRO + SA, CRO + ACO, etc.) or through the incorporation of greedy approaches. It is possible to give good results to a broader range of problems sets and find the global optima of some "unsolved" problems, based on this fundamental version of CRO.

ACKNOWLEDGMENT

The authors would like to thank A. P. Subramanian for providing them with his simulation codes, and for comparing the results of CAP. They are also grateful to W.K. Kwan of the University of Hong Kong Computer Center High Performance Computing Facilities for assisting with the testing of the simulation programs for RCPSP.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co. Ltd., 1979.
[2] D. M. Hirst, *Potential Energy Surfaces: Structure and Dynamics*. London, U.K.: Taylor & Francis, 1985.
[3] *Gibbs Free Energy*, Wikipedia [Online]. Available: http://en.wikipedia.org/wiki/Gibbs_free_energy
[4] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[5] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: Univ. Michigan Press, 1975.
[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.
[7] S. Forrest, "Genetic algorithms: Principles of natural selection applied to computation," *Science*, vol. 261, no. 5123, pp. 872–878, Aug. 1993.
[8] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, MA: MIT Press, 2004.
[9] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for optimization from social insect behavior," *Nature*, vol. 406, pp. 39–42, Jul. 2000.
[10] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm Intelligence*. San Francisco, CA: Morgan Kaufmann Publishers, 2001.
[11] D. T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, and M. Zaidi, "The bees algorithm–a novel tool for complex optimisation problems," in *Proc. 2nd Virtual Int. Conf. Innovative Production Machines Syst. (IPROMS)*, Jul. 2006, pp. 454–461.
[12] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: Harmony search," *Simulation*, vol. 76, no. 2, pp. 60–68, Feb. 2001.
[13] F. Glover and M. Laguna, *Tabu Search*. Boston, MA: Kluwer Academic Publishers, 1997.
[14] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *J. Chem. Phys.*, vol. 21, pp. 1087–1092, Jun. 1953.
[15] R. E. Burkard and F. Rendl, "A thermodynamically motivated simulation procedure for combinatorial optimization problems," *Eur. J. Oper. Res.*, vol. 17, no. 2, pp. 169–174, Aug. 1984.
[16] M. R. Wilhelm and T. L. Ward, "Solving quadratic assignment problems by 'simulated annealing'," *IIE Trans.*, vol. 19, no. 1, pp. 107–119, Mar. 1987.
[17] D. T. Connolly, "An improved annealing scheme for the QAP," *Eur. J. Oper. Res.*, vol. 46, no. 1, pp. 93–100, May 1990.
[18] P. Tian, H. Wang, and D. Zhang, "Simulated annealing for the quadratic assignment problem: A further study," *Comput. Ind. Eng.*, vol. 31, nos. 3–4, pp. 925–928, Oct. 1996.
[19] A. Misevičius, "A modified simulated annealing algorithm for the quadratic assignment problem," *Informatica*, vol. 14, no. 4, pp. 497–514, Dec. 2003.
[20] J.-C. Wang, "Solving quadratic assignment problems by a tabu based simulated annealing algorithm," in *Proc. Int. Conf. Intelligent Advanced Syst. (ICIAS)*, Kuala Lumpur, Malaysia, Nov. 2007, pp. 75–80.

[21] C. Y. Ngo and V. O. K. Li, "Fixed channel assignment in cellular radio networks using a modified genetic algorithm," *IEEE Trans. Veh. Technol.*, vol. 47, no. 1, pp. 163–172, Feb. 1998.
[22] S. M. Bhandarkar and H. Zhang, "Image segmentation using evolutionary computation," *IEEE Trans. Evol. Comput.*, vol. 3, no. 1, pp. 1–21, Apr. 1999.
[23] R. S. Parpinelli, H. S. Lopes, and A. A. Freitas, "Data mining with an ant colony optimization algorithm," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 321–332, Aug. 2002.
[24] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67–82, Apr. 1997.
[25] Y. C. Ho and D. L. Pepyne, "Simple explanation of the no-free-lunch theorem and its implications," *J. Optim. Theory Appl.*, vol. 115, no. 3, pp. 549–570, Dec. 2002.
[26] K. M. Sim and W. H. Sun, "Ant colony optimization for routing and load-balancing: Survey and new directions," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 33, no. 5, pp. 560–572, Sep. 2003.
[27] J. E. Bell and P. R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Adv. Eng. Informatics*, vol. 18, no. 1, pp. 41–48, Jan. 2004.
[28] E. M. Loiola, N. M. M. de Abreu, P. O. Boaventura-Netto, P. Hahn, and T. Querido, "A survey for the quadratic assignment problem," *Eur. J. Oper. Res.*, vol. 176, no. 2, pp. 657–690, Jan. 2007.
[29] E. L. Demeulemeester and W. S. Herroelen, *Project Scheduling: A Research Handbook*. Boston, MA: Kluwer Academic Publishers, 2002.
[30] D. Merkle, M. Middendorf, and H. Schmeck, "Ant colony optimization for resource-constrained project scheduling," *IEEE Trans. Evol. Comput.*, vol. 6, no. 4, pp. 333–346, Aug. 2002.
[31] A. P. Subramanian, H. Gupta, S. R. Das, and J. Cao, "Minimum interference channel assignment in multiradio wireless mesh networks," *IEEE Trans. Mobile Comput.*, vol. 7, no. 12, pp. 1459–1473, Dec. 2008.
[32] S. Sahni and T. Gonzalez, "P-complete approximation problems," *J. ACM*, vol. 23, no. 3, pp. 555–565, Jul. 1976.
[33] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton, NJ: Princeton Univ. Press, 2006.
[34] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo, "The maximum clique problem," in *Handbook of Combinatorial Optimization*, D.-Z. Du and P. M. Pardalos, Eds., vol. 4. Boston, MA: Kluwer Academic Publishers, 1999, pp. 1–74.
[35] P. Merz and B. Freisleben, "A genetic local search approach to the quadratic assignment problem," in *Proc. 7th Int. Conf. Genetic Algorithms*, T. Bäck, Ed. San Francisco, CA: Morgan Kaufmann, 1997, pp. 465–472.
[36] J. G. Digalakis and K. G. Margaritis, "On benchmarking functions for genetic algorithms," *Int. J. Comput. Math.*, vol. 77, no. 4, pp. 481–506, 2001.
[37] R. E. Burkard, S. E. Karisch, and F. Rendl, "QAPLIB: A quadratic assignment problem library," *J. Global Optim.*, vol. 10, no. 4, pp. 391–403, Jun. 1997.
[38] É. D. Taillard, "FANT: Fast ant system," Swiss inst. Artif. Intell., Lugano, Switzerland, Tech. Rep. IDSIA-46-98, 1998.
[39] É. D. Taillard, "Robust taboo search for the quadratic assignment problem," *Parallel Comput.*, vol. 17, nos. 4–5, pp. 443–455, Jul. 1991.
[40] Q. H. Nguyen, Y.-S. Ong, and M. H. Lim, "A probabilistic memetic framework," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 604–623, Jun. 2009.
[41] R. H. Möhring, A. S. Schulz, F. Stork, and M. Uetz, "Solving project scheduling problems by minimum cut computations," *Manage. Sci.*, vol. 49, no. 3, pp. 330–350, 2003.
[42] J. Y. Leung, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL: CRC Press, 2004.
[43] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *Eur. J. Oper. Res.*, vol. 127, no. 2, pp. 394–407, 2000.
[44] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: An update," *Eur. J. Oper. Res.*, vol. 174, no. 1, pp. 23–37, 2006.
[45] R. Kolisch and A. Sprecher, "PSPLIB: A project scheduling problem library," *Eur. J. Oper. Res.*, vol. 96, no. 1, pp. 205–216, 1997.

Albert Y.S. Lam (S'02) received the B.Eng. degree (First Class Honors) in information engineering from the University of Hong Kong (HKU), Pokfulam, Hong Kong, in 2005. Currently, he is working toward the Ph.D. degree from the Department of Electrical and Electronic Engineering, HKU.

His research interests include optimization theory and algorithms, evolutionary computation, wireless and mobile networking, and internet protocols and applications.

Victor O.K. Li (S'80–M'81–SM'86–F'92) received the S.B., S.M., E.E., and Sc.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, in 1977, 1979, 1980, and 1981, respectively.

He joined the University of Southern California (USC), Los Angeles, in February 1981, and became a Professor of Electrical Engineering and the Director of the USC Communication Sciences Institute. Since September 1997, he has been with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pokfulam, Hong Kong, where he is the Associate Dean (Research) of Engineering, and the Chair Professor of Information Engineering. He also served as the Managing Director of Versitech Ltd. (http://www.versitech.com.hk/), Cyberport, Hong Kong. He serves on various corporate boards. Having been invited by government, industry, and academic organizations, he has lectured and consulted extensively around the world. He has given distinguished lectures at various universities around the world, and keynote speeches at many international conferences. His research interest is in information technology, including the design, analysis, and optimization of communication networks.

Dr. Li is very active in the research community, and has chaired various international conferences and served on the editorial boards of various international journals. He has received numerous awards, including the People's Republic of China Ministry of Education Changjiang Chair Professorship at Tsinghua University, Beijing, China, the U.K. Royal Academy of Engineering Senior Visiting Fellowship in Communications, the KC Wong Education Foundation Lectureship, the Croucher Foundation Senior Research Fellowship, and the Order of the Bronze Bauhinia Star, Government of the Hong Kong Special Administrative Region, China.