<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Chemical-reaction-inspired metaheuristic for optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Lam, AYS; Li, VOK</td>
</tr>
<tr>
<td><strong>Citation</strong></td>
<td>IEEE Transactions On Evolutionary Computation, 2010, v. 14 n. 3, p. 381-399</td>
</tr>
<tr>
<td><strong>Issued Date</strong></td>
<td>2010</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10722/130634">http://hdl.handle.net/10722/130634</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>IEEE Transactions on Evolutionary Computation. Copyright © IEEE.; This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.; ©2010 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.</td>
</tr>
</tbody>
</table>
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 heuristics 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 heuristics, 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 heuristics 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 $C = \{c_1, c_2, \ldots, c_m\}$ 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

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

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. 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.

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

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" [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...
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].

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

Fig. 2. Comparison of performance for different problem types.
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 number of atoms and characterized by the atom type, 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. For example, if a problem defines the feasible solution set as the set of \( n \)-dimensional positive real numbers \( \mathbb{R}_n^+ \), 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. 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 \( \omega \) 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 \( \omega \) to \( \omega' \). The change is always possible if \( PE_\omega \geq PE_\omega' \). Otherwise, we allow the change only when \( PE_\omega + KE_\omega \geq PE_\omega' \). Thus, the higher the KE of the molecule, the higher change probability 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 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 unmolecular 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

---

4All 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.

5In 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.

6Note 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.
neighbors have similar PE. As the collision is not so vigorous, the resultant molecular structure varies accordingly.

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 \( \omega \). The molecule intends to obtain a new structure \( \omega' = \text{Neighbor}(\omega) \) (Table I) in its neighborhood on the PES in this collision. The change is allowed only if

\[
PE_{\omega'} + KE_{\omega'} \geq PE_{\omega}.
\]

The neighborhood structure is problem-dependent. Normally, \( \omega \) 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. The stored energy can be used to support decomposition. If (3) does not hold, the change is prohibited and the molecule retains its original \( \omega \), PE and KE. The pseudocode of the on-wall ineffective collision is as follows:

\[
\text{ineff_coll_on_wall}(M, \text{buffer})
\]

**Input:** A molecule \( M \) with its profile and the central energy buffer \( \text{buffer} \).

1. Obtain \( \omega = \text{Neighbor}(\omega) \)
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 \( \text{buffer} = \text{buffer} + (PE_{\omega} + KE_{\omega} - PE_{\omega'}) \times (1 - q) \)
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 \( \text{buffer} \)

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 \( \omega \) and those of the resultant molecules are \( \omega_1 \) and \( \omega_2 \). If the original molecule has sufficient energy (PE and KE) to endow the PE of the resultant ones, that is

\[
PE_{\omega} + KE_{\omega} \geq PE_{\omega_1} + PE_{\omega_2},
\]

the change is allowed. Let \( \text{temp} = PE_{\omega} + KE_{\omega} - PE_{\omega_1} - PE_{\omega_2} \)
We get

\[
KE_{\omega_1} = \text{temp} \times k
\]

and

\[
KE_{\omega_2} = \text{temp} \times (1 - k)
\]

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, \( PE_{\omega} \), \( PE_{\omega_1} \), and \( PE_{\omega_2} \) are of similar values (but much larger than those in the same neighborhood), (4) holds only when \( KE_{\omega} \) 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

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.
to hold in normal cases. To encourage decomposition, we use
the energy stored in the central buffer (buffer) to sustain $PE_ω$ and $KE_ω$. In other words, if (4) does not hold, we consider

$$PE_ω + KE_ω + \text{buffer} \geq PE_{\omega'} + PE_{\omega''}. \quad (5)$$

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

$$KE_{\omega'} = (temp_1 + \text{buffer}) \times m_1 \times m_2$$

and

$$KE_{\omega''} = (temp_1 + \text{buffer} - KE_{\omega'}) \times m_1 \times m_2$$

(6)

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 $KE_{\omega'}$ and $KE_{\omega''}$ are not too large. Then buffer is updated by $temp_1 + \text{buffer} - KE_{\omega'} - KE_{\omega''}$. 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 $ω'$ and $ω''$. From $ω$, any mechanism, resulting in $ω'$ and $ω''$ 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 $PE_ω$ and $KE_ω$.
3. Let $temp_1 = PE_ω + KE_ω - PE_{\omega'} - PE_{\omega''}$.
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. $KE_{\omega'} = temp_1 \times k$
9. $KE_{\omega''} = temp_1 \times (1 - k)$
10. Create new molecules $M_1'$ and $M_2'$.

3) Inter-Molecular Ineffective Collision: An inter-molecular ineffective collision (Fig. 4c) 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

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

(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 $ω_1$, $ω_2$, $PE_{\omega_1}$, $PE_{\omega_2}$, $KE_{\omega_1}$, and $KE_{\omega_2}$ if (8) fails. Inter-molecular
inefficient 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 inefficient collision is as follows:

\[
\text{inter\_ineff\_coll}(M_1, M_2)
\]

**Input:** molecules \(M_1, M_2\) with their profiles.
1. Obtain \(a_1' = \text{Neighbor}(a_1)\) and \(a_2' = \text{Neighbor}(a_2)\)
2. Calculate \(PE_{a_1}\) and \(PE_{a_2}\)
3. Let \(\text{temp}_2 = (PE_{a_1} + PE_{a_2} + KE_{a_1} + KE_{a_2}) - (PE_{a_1'} + PE_{a_2'})\)
4. if \(\text{temp}_2 \geq 0\) then
5. Get \(p\) randomly in interval \([0, 1]\)
6. \(KE_{a_1} = \text{temp}_2 \times p\)
7. \(KE_{a_2} = \text{temp}_2 \times (1 - p)\)
8. Update the profile of \(M_1\) by \(a_1 = a_1', PE_{a_1} = PE_{a_1'}, KE_{a_1} = KE_{a_1'}\), and the profile of \(M_2\) by \(a_2 = a_2', PE_{a_2} = PE_{a_2'}\) and \(KE_{a_2} = KE_{a_2'}\)
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 \(a_0\) and \(a_2\). We attempt to generate a new molecule with molecular structure \(a'_1\) from the two existing \(a_0\) and \(a_1\). Since synthesis is vigorous, \(a'_1\) should be quite different from \(a_0\) and \(a_1\). As in decomposition, any mechanism which combines \(a_0\) and \(a_2\) to form \(a'_1\) may be used. We accept \(a'_1\) only if

\[
PE_{a_0} + PE_{a_2} + KE_{a_0} + KE_{a_2} \geq PE_{a_1'}.
\]

We get \(KE_{a_0} = PE_{a_1'} + PE_{a_2} + KE_{a_2} + KE_{a_0} - PE_{a_0}\). We retain \(a_0\), \(a_{12}\), \(PE_{a_{12}}\), \(KE_{a_0}\), \(KE_{a_2}\), instead of \(a_0\), \(a_{12}\), and \(KE_{a_0}\), if (9) does not hold. Interestingly, \(KE_{a_0}\) is large when compared with \(KE_{a_1}\) or \(KE_{a_2}\) because \(PE_{a_{12}}\) is expected to have similar value to \(PE_{a_1}\) or \(PE_{a_2}\), generally. In this way, we give the resultant molecule with \(a'\) greater ability to escape from a local minimum in subsequent elementary reactions involving it. The pseudocode of the synthesis is as follows:

\[
\text{synthesis}(M_1, M_2)
\]

**Input:** molecules \(M_1, M_2\) with their profiles.
1. Obtain \(a'_1\) from \(a_0\) and \(a_2\)
2. Calculate \(PE_{a_1'}\)
3. Create a Boolean variable \(\text{Success}\)
4. Create a new molecule \(M'\)
5. if \(PE_{a_0} + PE_{a_2} + KE_{a_0} + KE_{a_2} \geq PE_{a_{12}}\) then
6. \(\text{Success} = \text{TRUE}\)
7. \(KE_{a_0} = PE_{a_0} + PE_{a_2} + KE_{a_0} + KE_{a_2} - PE_{a_{12}}\)
8. Assign \(a'_1, PE_{a_{12}},\) and \(KE_{a_0}\) to the profile of \(M'\)
9. else
10. \(\text{Success} = \text{FALSE}\)
11. end if
12. **Output** \(M'\) and \(\text{Success}\)

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.\(^\text{9}\) 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 inefficient collision and inter-molecular inefficient 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.\(^\text{8}\) 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 then calculate the best solution and the final solution. This framework does not fix the conditions, and examples of the conditions are given in Section IV-A.
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 per-
formed 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:

\[
\text{CRO}(f, Nvars) \\
\text{Input: Problem-specific information (the objective function} \\
f, \text{constraints, and the dimensions of the problem} \ Nvars) \\
\begin{enumerate}
  \item Assign parameter values to \( PopSize, KElossRate, \) \\
  \( \text{MoleColl and InitialKE} \)
  \item Let \( Pop \) be the set of molecule 1, 2, \ldots, \\
  \( PopSize \)
  \item for each of the molecules do
  \item Assign a random solution to the molecular \\
  structure \( \omega \)
  \item Calculate the \( PE \) by \( f(\omega) \)
  \item Assign the \( KE \) with \( InitialKE \)
  \item end for
  \item Let the central energy buffer be \( buffer \) and assign \\
  \( buffer = 0 \)
  \item while the stopping criteria not met do
  \item Get \( r \) randomly in interval \([0, 1]\)
  \item if \( r > MoleColl \) then
  \end{enumerate}
\]
buffer molecules, i.e., their inter-molecular ineffective collision involves two or more. So the possibility of a molecule having a larger value of $\omega$ 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

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

When a series of on-wall ineffective collisions happen, the molecules tend to have lower KE and result in subsequent solutions $\omega'$ 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 $\text{InitialKE}$ and $\text{PopSize}$. 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 $\text{InitialKE}$ 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 increase 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 among the molecules. The on-wall 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

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

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 NPL 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 × 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). 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_{ijkl} as the distance between locations i and l. We write the objective function and the constraints as follows:

$$\min \sum_{i,j} \sum_{k,l} f_{ij} d_{ijkl} x_{ij} x_{kl}$$ (11)

subject to

$$\sum_{j=1}^{n} x_{ij} = 1, \quad 1 \leq j \leq n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad 1 \leq i \leq n$$

$$x_{ij} \in [0,1], \quad 1 \leq i,j \leq n$$

10Combinatorial 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.

11The 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\text{14}.
evaluate this solution by computing

\[ \text{assign facility 2 to location 1, facility 4 to location 2, etc.} \]

we equal to four. One possible solution is \([2, 4, 3, 1]\), i.e., we

the products of flow and distance of every possible pair in

respond to the locations and the facilities, respectively. Then

every possible solution is in the form of a permutation of

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

\[ [1 \ 2 \ 3 \ 4 \ 5 \ 6] \]

\[ [1 \ 5 \ 3 \ 4 \ 2 \ 6] \]

\([2 \ 3 \ 4 \ 5 \ 6 \ 1] [5 \ 6 \ 1 \ 2 \ 3 \ 4]\]

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 \(\omega_1^\prime\) and \(\omega_2^\prime\)

from \(\omega_1\). 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., \((\text{number of hits} \rightarrow \text{minimum hit number}) > \alpha\)

(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 \(\alpha\).

In synthesis, we attempt to generate a new molecule with

solution \(\omega^\prime\) from two existing molecules with solutions, \(\omega_1\) and

\(\omega_2\). We adopt the distance-preserving crossover operator [35]

to the minimum structures of the two existing molecules to get \(\omega^\prime\). 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}\text{Here, we generate two random integers independently. Then we apply}

circular shift with the two integers separately to obtain the independent solutions.\)
We program the simulation codes. We compare the results are reached. We do not adopt the computational time as the criterion is when a certain number of function evaluations. Thus, the stopping of computation [24], i.e., finding the best solution within a library [37] and compare the results with an oracle-based view problem instances (Table V) selected from a QAP digital library/taillard/). We modified their original codes to meet the stopping criterion.

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. 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]. 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. 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 III

<table>
<thead>
<tr>
<th>Implementation Details</th>
<th>QAP</th>
<th>RIBSP</th>
<th>IAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution space</td>
<td>Permutation set</td>
<td>Permutation set</td>
<td>Integer vector</td>
</tr>
<tr>
<td>Parameter settings</td>
<td>Second column of Table II</td>
<td>Third column of Table II</td>
<td>Fourth column of Table II</td>
</tr>
<tr>
<td>Neighborhood structure</td>
<td>Two-exchange neighborhood structure</td>
<td>Same as QAP</td>
<td>Same as QAP</td>
</tr>
<tr>
<td>Decomposition criterion</td>
<td>(number of hit minus hit number) &gt; α</td>
<td>Same as QAP</td>
<td>Same as QAP</td>
</tr>
<tr>
<td>Decomposition mechanism</td>
<td>Circular shift</td>
<td>Same as QAP</td>
<td>Same as QAP</td>
</tr>
<tr>
<td>Synthesis criterion</td>
<td>KEω &lt; β and KEω &lt; β</td>
<td>Same as QAP</td>
<td>Same as QAP</td>
</tr>
<tr>
<td>Synthesis mechanism</td>
<td>Distance-preserving crossover</td>
<td>Same as QAP</td>
<td>Same as QAP</td>
</tr>
</tbody>
</table>

### TABLE IV

<table>
<thead>
<tr>
<th>Elementary Reactions</th>
<th>Intensification</th>
<th>Diversification</th>
</tr>
</thead>
<tbody>
<tr>
<td>On-wall ineffective collision</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Decomposition</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Inter-molecular collision</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Synthesis</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

---

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 parts of the solution spaces are explored by the algorithms, and thus, the power of the algorithms cannot be revealed.

18 Computational times for a run of simulation on the test instances are also provided in Table V as reference.
solution space is of the order of $10^{35} (=32!)$, and its global minimal solution with real-world data used to plan Klinikum Regensburg in Germany. The simulation run, we examine the case of the instance kra3218 has function value equal to 88 700.

CRO outperforms all the others, in terms of the mean and the whole solution space only, in each run of simulation.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Problem Size</th>
<th>Global Function Evaluation Limit</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>StdDev</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>StdDev</th>
<th>Computational time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nug27</td>
<td>27</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>0.83</td>
</tr>
<tr>
<td>nug22</td>
<td>22</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>0.83</td>
</tr>
<tr>
<td>nug10</td>
<td>10</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>0.83</td>
</tr>
<tr>
<td>nug5</td>
<td>5</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>0.83</td>
</tr>
<tr>
<td>nug</td>
<td>1</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>0.83</td>
</tr>
</tbody>
</table>

In order to understand the performance of CRO during a simulation run, we examine the case of the instance kra3218 in detail. Each run of simulations is terminated once the number of function evaluations reaches 150,000, where we examine roughly a $5.70 \times 10^{-31}$ ($=\frac{150,000}{2500}$) 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.

In the table, the best results are highlighted in bold. The table shows simulation results of CRO on solving QAP compared with the other three metaheuristics.

**Table V**

Simulation Results of CRO on Solving QAP Compared with the Other Three Metaheuristics

<table>
<thead>
<tr>
<th>Instance</th>
<th>Problem Size</th>
<th>Global Function Evaluation Limit</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>StdDev</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>nug27</td>
<td>27</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug22</td>
<td>22</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug10</td>
<td>10</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug5</td>
<td>5</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug</td>
<td>1</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
</tbody>
</table>

In the table, the best results are highlighted in bold. The table shows simulation results of CRO on solving QAP compared with the other three metaheuristics.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Problem Size</th>
<th>Global Function Evaluation Limit</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>StdDev</th>
<th>Min.</th>
<th>Max.</th>
<th>Mean</th>
<th>StdDev</th>
</tr>
</thead>
<tbody>
<tr>
<td>nug27</td>
<td>27</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug22</td>
<td>22</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug10</td>
<td>10</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug5</td>
<td>5</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
<tr>
<td>nug</td>
<td>1</td>
<td>5234</td>
<td>5234.00</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
<td>(0)</td>
</tr>
</tbody>
</table>
to 1. cost increases and then decreases as the last two data points for kra32 at the ends of the 50 runs. As shown in Fig. 9, the parameter basis. We repeat the simulations on kra32 for 50 times. The reasonable ones and fix the rest with the values as in the investigated, we replace the parameter value with some other settings in Table II be the basis. For each of the parameters Table II, we do an analysis on the parameters. Let the results of the whole 50 runs.

For problems other than QAP, there are also real-life instances common characteristic is that the fitness landscapes formed the website of the QAP digital library [37], most of the test instances imply that CRO can achieve the most stable results. From Table V, CRO can outperform the other three metaheuristics in many of the test instances. For esc32c, especially when the number of runs is not sufficiently large. As shown in Table V, CRO can generate the minimum standard deviations among all the metaheuristics in many of the test instances. For esc32c, 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-preemptible 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:

$$\min f_{n+1} \quad (13)$$

subject to $f_0 = 0$, $f_i \leq f_j - d_j \quad 1 \leq j \leq n + 1, h \in P_j$, $\sum_{h \in P_j} r_{i,h} \leq R_k \quad k \in K, t \geq 0$, $f_j \geq 0 \quad 0 \leq j \leq n + 1$.

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. We use the
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 /20 from a digital library PSPLIB [45]. /20 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.
Serial schedule generation

**Input**: Activity list \( \omega \)

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

**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.

### Table VII: Number of Instances with Best Solution Found Reported in PSPLIB

<table>
<thead>
<tr>
<th>Authors</th>
<th>Year</th>
<th>Number of Instances With Best Makespan Found</th>
</tr>
</thead>
<tbody>
<tr>
<td>D. Sun, S. Zhou</td>
<td>2007</td>
<td>129</td>
</tr>
<tr>
<td>K. Bouleziers</td>
<td>1997</td>
<td>120</td>
</tr>
<tr>
<td>Authors of this paper</td>
<td>2009</td>
<td>116</td>
</tr>
<tr>
<td>V. Valls, M. Quintanilla, F. Balleano</td>
<td>2002</td>
<td>73</td>
</tr>
<tr>
<td>K. Nonote and T. Inoue</td>
<td>1998</td>
<td>69</td>
</tr>
<tr>
<td>F. Xiao, A. Lin and B. Rodriguez</td>
<td>2005</td>
<td>57</td>
</tr>
<tr>
<td>T. Beir, P. Broders, S. Kwast</td>
<td>1997</td>
<td>42</td>
</tr>
<tr>
<td>D. Debels and M. Vanhoucke</td>
<td>2005</td>
<td>36</td>
</tr>
<tr>
<td>M. R. Ramybar, F. Kamaf</td>
<td>2007</td>
<td>24</td>
</tr>
<tr>
<td>V. Valls, M. Quintanilla, F. Balleano</td>
<td>2009</td>
<td>14</td>
</tr>
<tr>
<td>Jose Fernando Goncalves &amp; Jorge Magalhae</td>
<td>2006</td>
<td>11</td>
</tr>
<tr>
<td>D. Merkle, M. Millendorf, H. Schmeck</td>
<td>2000</td>
<td>5</td>
</tr>
<tr>
<td>J. Horstma and J. Humberger</td>
<td>2005</td>
<td>5</td>
</tr>
<tr>
<td>A. Horbach</td>
<td>2008</td>
<td>4</td>
</tr>
<tr>
<td>M. Palpant, C. Antigues, P. Michelon</td>
<td>2002</td>
<td>4</td>
</tr>
<tr>
<td>D. Sun</td>
<td>2007</td>
<td>2</td>
</tr>
<tr>
<td>J. Alcaraz and C. Maroto</td>
<td>1998</td>
<td>2</td>
</tr>
<tr>
<td>K. Hexas, K. S. Chen</td>
<td>2002</td>
<td>1</td>
</tr>
<tr>
<td>L.Y. Tseng and S.C. Chen</td>
<td>2004</td>
<td>1</td>
</tr>
<tr>
<td>P. Laborie</td>
<td>2005</td>
<td>1</td>
</tr>
</tbody>
</table>
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 \(G_c(V, E_c)\). \(V\) 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 \(u, v \in V\), an edge \((u, v)\) means that \(u\) and \(v\) interfere each other.

Consider a channel assignment as a function \(f: V \rightarrow K\), i.e., assigning each link to exactly one channel in the channel set \(K\). We define \(E(i) \subseteq V\) 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 interference subject to the interface constraint. Mathematically, we have

\[
\min I(f)
\]

subject to \(|\{(u, v) \in E_c\} | f(u) = f(v)\| \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. 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 \(|V|\) 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 \(\omega_i\) is a neighbor of solution \(\omega\) 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 \(\omega_i\) and \(\omega_j\) from \(\omega\). The two new solutions are obtained in the same way: randomly select \(|V|/2\) number of vertices in \(\omega\) and assign a different channel to each of them. We name this decomposition mechanism as “half-total-change.”

In synthesis, we get a new solution \(\omega\) from two solutions, \(\omega_i\) and \(\omega_j\). We generate \(\omega\) as follows: for each vertex in \(\omega\), we generate a random number \(\xi \in [0, 1]\). If \(\xi > 0.5\), we assign the channel number at that vertex from \(\omega_i\). Otherwise, we take the channel number at that vertex from \(\omega_j\). We name this synthesis mechanism as “probabilistic select.” We summarize the above implementation details in the fourth column of Table III.

2) Results: As there is no benchmark problem library for this CAP, we generate a problem instance ourselves.\(^{21}\) 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 having a tabu-based algorithm in 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.
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.

TABLE VIII

<table>
<thead>
<tr>
<th>Number of Radio Interfaces Per Node</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean (%)</td>
<td>47.98</td>
<td>36.94</td>
<td>21.28</td>
<td>14.94</td>
<td>7.92</td>
<td>5.10</td>
<td>3.98</td>
<td>3.76</td>
<td>3.74</td>
<td>3.74</td>
</tr>
<tr>
<td>StdDev</td>
<td>0.62</td>
<td>1.06</td>
<td>1.56</td>
<td>1.85</td>
<td>0.84</td>
<td>0.52</td>
<td>0.22</td>
<td>0.09</td>
<td>0.08</td>
<td>0.06</td>
</tr>
<tr>
<td>Maximum (%)</td>
<td>49.54</td>
<td>33.33</td>
<td>23.59</td>
<td>18.25</td>
<td>9.95</td>
<td>6.91</td>
<td>4.96</td>
<td>4.09</td>
<td>3.87</td>
<td>3.84</td>
</tr>
<tr>
<td>Minimum (%)</td>
<td>46.66</td>
<td>27.67</td>
<td>17.55</td>
<td>9.55</td>
<td>6.50</td>
<td>3.96</td>
<td>3.62</td>
<td>3.59</td>
<td>3.54</td>
<td>3.61</td>
</tr>
<tr>
<td>ResultTS ResultCRO × 100%</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

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


