Parallel Chemical Reaction Optimization for the Quadratic Assignment Problem

Jin Xu, Albert Y.S. Lam, and Victor O.K. Li
Department of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam Road, Hong Kong, China

Abstract— *Chemical Reaction Optimization (CRO), a recently proposed metaheuristic, has demonstrated its capability in solving NP-hard optimization problems. CRO is a population-based evolutionary technique inspired by the interactions between molecules in a chemical reaction. In this paper, we present a parallel version of CRO (named PCRO) with a synchronous communication strategy. PCRO is applied to solve the Quadratic Assignment Problem (QAP), which is considered one of the great challenges in combinatorial optimization. Simulation results show that compared with the sequential CRO, our proposed PCRO can not only reduce the computation time but also improve the quality of the solution for instances of QAP with large sizes.*

Keywords: Parallel metaheuristics, chemical reaction optimization, quadratic assignment problem.

1. Introduction

Metaheuristics are powerful optimization techniques that have been gaining attention in recent years. The merits of metaheuristic methods are mainly two-fold. Firstly, these techniques have the capability to solve a wide range of problems with little or no knowledge of the search space. Thus, they are easily adjusted to fit the problem. Secondly, most optimization problems are intractable and *NP*-hard, which means that the optimal solution can not be found in polynomial time. Metaheuristics randomly search for possible solutions by following certain evolutionary rules. Most of the time, they can reach the optimal or near-optimal solutions in a reasonable running time. Some of these algorithms are nature-inspired, like Genetic Algorithm (GA) [1], Simulated Annealing (SA) [2], Ant Colony Optimization (ACO) [3], and Chemical Reaction Optimization [4]. Others are not nature-inspired, and examples are Tabu Search (TS) [5] and Threshold Accepting (TA) [6].

CRO mimics the interactions of molecules in a chemical reaction. According to the No Free Lunch theorem, "for any algorithm, any elevated performance over one class of problem is exactly paid for in performance over another class" [7]. Therefore, CRO has equal performance as other metaheuristics on the average and may outperform others in some type of problems (while it may perform worse in other types). As a new metaheuristic approach, CRO is still being improved. However, it has already shown its competitiveness with other existing methods in the Quadratic Assignment Problem (QAP), Resource-Constrained Project Scheduling Problem (RCPSP), Channel Assignment Problem (CAP) [4], Grid Scheduling Problem (GSP) [8], and Population Transition Problem [9].

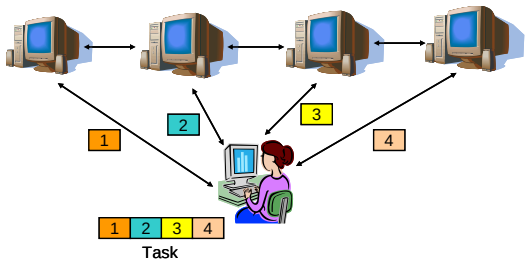


Fig. 1: Parallel Computing

With the continuous improvement of CPU speed and the advancement of multicore CPU, computer performance has been greatly boosted. Meanwhile, the network is also significantly improved and popularized all over the world. Therefore, more computing resources are readily available with the help of network, which accordingly makes the parallel and distributed computing a trend in the future (see Fig. 1). Grid computing and cloud computing are two kinds of such frameworks targeting at fully utilizing the idle resources in order to improve the system performance. On the other hand, though metaheuristics can tackle intricate problems and find satisfactory solutions, the CPU requirement is still quite demanding (i.e. computation time is high) especially for problems of large sizes. This often makes the algorithms impractical in the scenarios where solutions have to be quickly obtained for decision making, such as in dynamic scheduling [10]. Thus, it is necessary and important for us to parallelize algorithms to reduce computation time.

Basically, there are two major benefits from using parallel metaheuristics. One is that the execution time for the algorithm will be greatly reduced if we employ more machines or processors cooperatively in solving the problem. Moreover, the quality of the solution can also be improved, provided that the parallel structure is well designed. For the classical metaheuristics, such as GA, SA and PSO, several parallel versions (named PGA, PSA and PPSO) [11][12][13] have already been developed and they have shown their

capability in solving large scale problems. Although most popular metaheuristics have their own parallel versions, it is important to notice that due to their intrinsic and different characteristics, their parallel designs are significantly different. For example, GA is a population-based algorithm and the population can be divided into different parts and each of them evolves semi-independently, allowing individuals to exchange between subpopulations. On the other hand, SA manipulates a single solution in each iteration, and its parallel design is rather limited and relatively simple. Though CRO is also a population-based algorithm, it is different from GA in that the number of solutions varies as it evolves. Thus, in order to test whether CRO is suitable for parallel implementation, in this paper, we develop the preliminary parallel version of CRO.

The remaining of the article is organized as follows. The basic concept and framework of CRO are described in Section II. In Section III, we address our parallel model for CRO. The experimental results are reported and analyzed when solving the Quadratic Assignment Problem in Section IV. Finally, some concluding remarks and topics for future investigation are given in Section V.

2. Chemical Reaction Optimization

In a chemical reaction, a set of chemical substances transform to another with the goal of reaching the minimum state of free energy. From the microscopic point of view, a molecule is the smallest indivisible unit in a chemical reaction that has unique chemical properties. Chemical reactions can be thought of making or breaking chemical bonds to form or dissociate molecules respectively, and thus, new molecules are generated. This can happen either spontaneously without requiring external energy, or non-spontaneously where the reactions are stimulated by external energy.

CRO mimics the process of a chemical reaction through a sequence of intermediate reactions, and the resultant molecules (i.e. the products in a chemical reaction) tend to stay at the most stable state with the lowest free energy. We will describe how to apply the chemical reaction concept to optimization in the following:

The chemical reaction is designed to happen in a closed container, starting with a certain number of molecules. The molecular structure of each molecule is viewed as a possible solution of the optimization problem. There are two kinds of molecular energies: potential energy (*PE*) and kinetic energy (*KE*). The former corresponds to the objective function value, while the latter is used to control the acceptance of new solutions with worse values. Let ω and f be a molecular structure (solution) and the objective function value respectively, then $PE_\omega = f(\omega)$. Consider that ω attempts to change to ω' in a collision. We replace ω with ω' if $PE_{\omega'} \leq PE_\omega$. Otherwise, we also accept the change when $PE_{\omega'} \leq PE_\omega + KE_\omega$. Therefore, *KE* can be considered as the ability of the molecule to escape from local optimums. Energy transformation plays an important role in the process of reaction. Moreover, a central energy buffer is also employed to adjust the energy distribution under the conservation of energy. These allow the algorithm to search in different regions of the solution space.

There are four types of elementary reactions in CRO: on-wall ineffective collision, decomposition, inter-molecular ineffective collision, and synthesis. The former two reactions happen when molecules collide on the wall of the container, while the latter two involve only molecules that interact with each other. Moreover, only one molecule involves the on-wall ineffective collision and decomposition reactions, while more than one molecules are needed in the inter-molecular ineffective collision and synthesis. For the two ineffective collisions (i.e. the on-wall ineffective collision and the inter-molecular ineffective collision), the number of molecules remain the same before and after the reactions, and only the neighborhoods of original solution are explored. For the decomposition and synthesis, the former combines all the molecules into one, while the latter splits one molecule into several. These two elementary reactions produce new solutions significantly different from the original ones, and different regions of the solution space can be searched.

Algorithm 1 Pseudocode for the basic CRO

```
1.  Initialize population with random solutions, and set
    the parameters.
2.  Compute the fitness value of each molecule as PE.
3.  While stopping criterion not met
4.      Choose one reaction from the four elementary
        collisions according to certain rules.
5.      Randomly select the molecule(s) for reaction.
6.      Generate the new molecule(s).
7.      If the new solution acceptance rules satisfied
8.          Substitute new molecule(s) for original one(s).
9.          Update the KE for the new molecule(s), and the
            central energy buffer.
10.     Else
11.         Keep the original molecule(s).
12. End While
13. Output the solution with the minimum PE in the
    population, and its fitness value.
```

The canonical CRO follows the pseudocode presented in Algorithm 1. Similar to other metaheuristics, the progress of CRO can also be divided into three stages: initialization, iterations, and the final output stage. We set the system parameters, including the population size, the initial *KE* of molecules, *MoleColl* and *KE* loss rate (defined next), and generate initial solutions randomly in the initialization step. Then, the algorithm enters iterations until a stopping criterion is met. We report the best solution in the final

stage. In Step 4, an elementary collision is selected. We use *MoleColl* $\in [0, 1]$ to decide whether the collision is inter-molecular or not. To do this, a random value t is generated in the interval [0, 1]. If t is larger than *MoleColl*, it will be a unimolecular collision. Otherwise, an inter-molecular collision will take place. α and β are two thresholds used to define the decomposition and synthesis criteria, respectively. For unimolecular collisions, α represents the maximum number of hits for a molecule without finding an improved solution. In other words, if a molecule can not find a better solution with the number of hits larger than α, decomposition will take place, otherwise the on-wall ineffective collision happens. For inter-molecular collisions, when both molecules do not have sufficient *KE*, i.e. $KE_{\omega_1} \leq \beta$ and $KE_{\omega_2} \leq \beta$, the synthesis reaction will be activated. Otherwise the inter-molecular ineffective collision occurs. Steps 7 to 11 illustrate that, if the energy conditions are not satisfied, the corresponding reaction will not be triggered, and the molecules will remain exactly the same for the next iteration. Otherwise, the reaction will take place, and the molecules and their corresponding energies will be updated. Note that the total number of molecules at a particular moment is changeable through decomposition and synthesis.

3. Parallel Chemical Reaction Optimization

CRO is a population-based metaheuristic involving a pool of molecules. Each molecule can be considered as an independent unit, which includes the molecular structure (solution), *PE* (fitness value), and *KE*. Each tank of molecules is handled by a processor and multiple tanks can be manipulated simultaneously in parallel. The molecules can be exchanged among the processors from time to time.

The main goal of this paper is to investigate the possibility of parallelizing CRO. We create a parallel version for CRO, called PCRO (Parallel Chemical Reaction Optimization), which can be efficiently executed on a parallel platform. The difference between the design for PCRO and PGA (Parallel Genetic Algorithm) are mainly three-fold. Firstly, in PCRO, it is the molecules (solutions) with associated attributes that are swapped during the communication interval, while in PGA, only individuals (solutions) can be exchanged. Moreover, energy in each buffer can also be transferred, and this influences the degree of intensification and diversification of the search. Secondly, the number of molecules controlled by each processor vary in the evolutionary process due to synthesis and decomposition, whereas for the PGA, the number of individuals in each subpopulation generally remains the same. Thirdly, CRO generates new solutions through four elementary reactions. However, new solutions are produced in GA mainly via crossover and mutation.

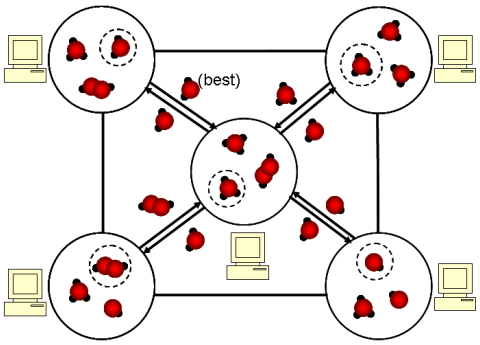


Fig. 2: Molecules' migration among containers

Algorithm 2 Pseudocode for Ψ

For the processor of the central node
1. Receives molecules from all of the other processors.
2. Chooses the best one from all the molecules.
3. Sends a copy of best molecule to other processors.
4. Removes all the molecules except the best one.
5. Updates the central energy buffer.

For other processors
1. Sends a copy of best molecule to central node.
2. Receives the molecule from central node.
3. Randomly substitutes one molecule.
4. Updates the central energy buffer.

Algorithm 3 Pseudocode for PCRO

```
1.  Builds the parallel platform, and sets the update rate.
2.  For each processor, initializes its molecules, and
    sets the system parameters.
3.  Do in parallel in each processor :{
4.  While stopping criterion not met
5.    While update rate not met
6.      Chooses one reaction from the four elementary
        collisions according to certain rules.
7.      Randomly selects the molecule(s) for reaction.
8.      Generates the new molecule(s).
9.      If the new solution acceptance rules satisfied
10.       Substitutes new molecule(s) for original one(s).
11.       Updates the KE for the new molecule(s), and
          the central energy buffer.
12.     Else
13.       Keeps the original molecule(s).
14.   End While
15. Ψ // communication
16. End While
17. } // End of parallel code
18. Outputs the best solution and its fitness value from
    all of the processors.
```

In our parallel CRO model, all molecules are divided into small groups and distributed to different processors

(containers). Processors communicate with each other in a synchronized way, in which the molecules are redistributed among the processors with a certain frequency. The network topology of the processors are assumed fully connected, which is also the case in our experiment. Assume that one of the processors is the central node. In each communication, the molecule with the minimum *PE* (best fitness value) in each container is copied to the central node. Then, we deliver the best of such solutions to each container, and it randomly substitutes one of the molecules in that container (see Fig. 2). To maintain the conservation of energy in each container, the central energy buffer will be updated after exchanging molecules. The pseudocode for this communication mechanism (named as Ψ) is shown in Algorithm 2. We also give the whole PCRO in Algorithm 3.

The delivery of the best molecule among processors in each communication period will speed up the convergence of the algorithm. Meanwhile, the energy is different in each container, and this allows PCRO to explore a wider solution space to avoid getting stuck in the local optimums. Thus, these two characteristics give PCRO a good balance between intensification and diversification.

4. Simulation Results

In this section, one of the classical *NP*-hard problems, QAP [14], is used to study the behavior of the parallel model for CRO introduced in the previous section. We will first briefly describe QAP, and simulation background and methodology will then be explained. Finally we discuss the simulation results.

4.1 Quadratic Assignment Problem

QAP is a fundamental combinatorial optimization problem. Shani and Gonzalez [15] have proved that it is an *NP*-hard problem, and there is no ϵ-approximation algorithm for the QAP unless *P*=*NP*. Moreover, QAP is the generalization of many other well-known *NP*-hard problems, including the traveling salesman problem [16], bin packing problem [17], and graph-partitioning problem [18].

In QAP, a set of n facilities are assigned to n locations in such a way that each facility must be allocated to exactly one location. We try to minimize the total cost, which is the summation of the distance of any pair of locations multiplied by the flow generated between them. This can be defined as follow.

$$\min_{p\in\Omega} \sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} \times d_{p(i)p(j)} \tag{1}$$

where f_{ij} is the flow between the facilities i and j, and $p(i)$ and $p(j)$ represent the locations assigned for i and j, respectively. Thus, $d_{p(i)p(j)}$ is the distance between locations $p(i)$ and $p(j)$. Ω is the solution space with all the permutations of n elements, and p is a possible permutation. Usually, the

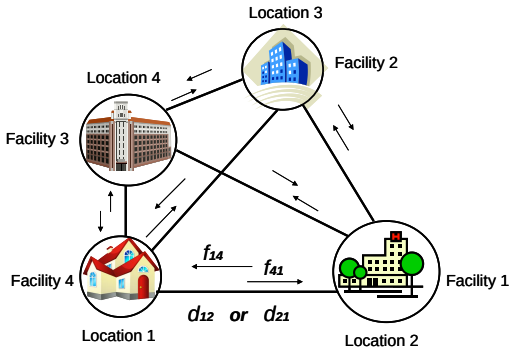


Fig. 3: A example for QAP

flow f's and distance d's are arrayed in square matrices with non-negative values. Therefore, QAP can be looked at as a minimization problem with the variables of p.

For example, suppose four facilities are allocated to four locations. p = [4, 1, 2, 3] is a possible solution, where the 4^{th} facility is distributed to location 1, the 1^{st} facility is assigned to location 2, and the 2^{nd} and 3^{rd} facilities to locations 3 and 4 respectively (see Fig. 3). The cost is evaluated by computing:

$$\begin{aligned} &f_{44}d_{11} + f_{41}d_{12} + f_{42}d_{13} + f_{43}d_{14} + \\ &f_{14}d_{21} + f_{11}d_{22} + f_{12}d_{23} + f_{13}d_{24} + \\ &f_{24}d_{31} + f_{21}d_{32} + f_{22}d_{33} + f_{23}d_{34} + \\ &f_{34}d_{41} + f_{31}d_{42} + f_{32}d_{43} + f_{33}d_{44}. \end{aligned} \tag{2}$$

Usually, the QAP instances with the size n larger than 25 are considered intractable [19]. The number of possible solutions of size n is $n!$ (When n=25, $n! \approx 1.55 \times 10^{25}$). However, in practice, many real world applications formulated as QAP, such as image processing [20], are often with the size of several hundreds. Compared with other metaheuristics, CRO in [4] has already shown its good performance in solving QAP [19]. However, we also find that the computing time consumed by CRO grows rapidly with the size. Reducing the computation time as well as keeping or improving the quality of the solution are the motivations for us to design PCRO, and thus, we adopt the large scale benchmark instances (Wil100, Tho150, and Tai256c in QAP library [19], which have sizes of 100, 150, 256 respectively) to test the performance of our PCRO.

4.2 Simulation Setup

In order to test PCRO, we compare it with the sequential CRO introduced in [4] when solving the three instances described in the previous subsection. For fairness, we also use the same operators for CRO in PCRO, including the two-exchange operator for on-wall ineffective collision and intermolecular ineffective collision, the circular shift operator for decomposition, and the distance-preserving crossover operator for synthesis [4]. The algorithm parameters are

Table 1: Parameters for PCRO used in simulation

	Parameter	Value
Particular for PCRO	Total population size	24
	Number of processors	1, 2, 4, 8
	Subpopulation size	24/ number of processors
	exchange/update rate	analyzed below
Same as CRO [4]	*KELossRate*	0.8
	MoleColl	0.2
	InitialKE	1000000
	α	1300
	β	10000

shown in Table 1. CRO and PCRO are coded in C++ and the simulation is performed on a cluster of computers with an Intel Core Quad 2.66GHz CPU and 4G RAM connected in a Ethernet. We realize the communication between the processors by MPICH2 [21], which is a high-performance and widely used implementation of the Message Passing Interface (MPI). The software library is convenient for the users to design a parallel computing algorithm, and shortens the development time.

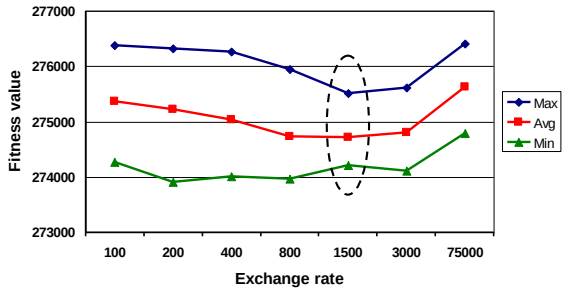
Since CRO and PCRO are stochastic algorithms, the results obtained in different runs may be different. We repeat the simulation 50 times, and record the minimum, maximum and average values. In each simulation run, the initial molecules (solutions) are generated randomly. Exchange (update) rate is the parameter representing how often we exchange the molecules between the processors (analyzed in the next subsection).

4.3 Analysis of Results

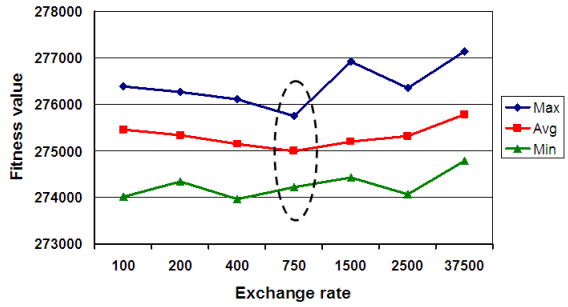
In this section, we study the behavior of PCRO in two ways when solving the QAP. Firstly, we adopt the same number of evaluations (150000) as the stopping criterion for both CRO and PCRO, and the exchange rate, speedup, and quality of solution are analyzed. After that, we use the average and minimum fitness values of CRO obtained from the former simulation as the stopping criterion to test and compare the execution time and hit rate of both CRO and PCRO.

4.3.1 Number of function evaluations as the stopping criterion

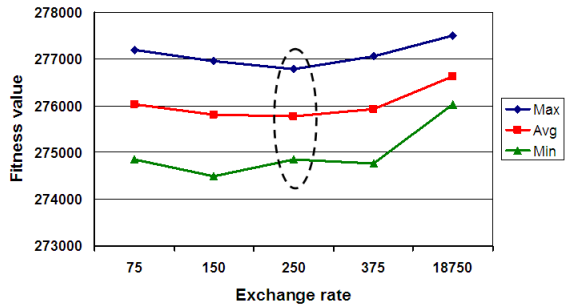
Fig. 4 shows the performance of using different exchange rates for paralleling 2, 4, and 8 processors, respectively. From observing these figures, we can come to the conclusions: When the exchange rate is low (i.e. the molecules migrate between processors in a high frequency.) the algorithm performs badly. This is because the communication will enhance the algorithm's intensification. If the exchange happens too often, the probability of getting stuck in the local optimum will be very high. Meanwhile, the performance



(a) 2 processors



(b) 4 processors



(c) 8 processors

Fig. 4: Fitness values with different exchange rates in the instance Wil100

of PCRO is also poor when the exchange rate is high. Seldom communications between processors will lead the algorithm to be too diversified to obtain a good solution. Moreover, the more the processors, the wider the solution space searched simultaneously. In other words, the diversification is strengthened with the number of processors. Thus, in order to reach a good balance between diversification and intensification, we swap the molecules more frequently when we have more processors. According to the average fitness value, we choose the exchange rates of 1500, 750, 250 for 2, 4, and 8 processors, respectively.

One important measure for a parallel algorithm is *speedup* [22]. This metric calculates the ratio of the computation time of CRO and PCRO. Since both are stochastic, we should

instead compare the *mean* serial execution time against the *mean* parallel time as the *speedup*, i.e.,

$$Speedup_m = \frac{E[T_1]}{E[T_m]} \quad (3)$$

According to this definition, we can distinguish among: *sublinear* speedup ($Speedup_m < m$), *linear* speedup ($Speedup_m = m$), and *superlinear* speedup ($Speedup_m > m$). Moreover, there are two types of speedup: strong and weak [22]. *Strong speedup* compares the execution time of parallel algorithm against the best-so-far sequential algorithm. However, due to the difficulty of finding the most efficient algorithm, most designers do not adopt it. On the other hand, *weak speedup* compares the run time of parallel algorithm against its own sequential one, which is used by many researchers. Thus, in this paper, we use the *weak speedup* to test PCRO.

Tables 2, 3 and 4 show the results of applying PCRO in the instances of Wil100, Tho150, and Tai256c, respectively. It can be observed that for each instance the *speedup* is almost *linear* when using 2 or 4 processors, and *sublinear* when there are 8 processors. This is mainly due to the overhead of communication in 8 processor scenario much more than that in the 2 or 4 processor scenario. From comparing the average fitness values, we can find that in most cases the quality of the final solution generated by PCRO is better than that of the sequential one (in bold). However, we notice that in Wil100 and Tho150, as the number of processors increases, the performance of PCRO becomes worse. It indicates that PCRO should be carefully designed when more and more processors are used.

Table 2: Results of the instance Wil100

Number of processors	Fitness values Min	Max	Avg	Average time(s)	Speedup
1	274716	276046	275360	8.4711	N/A
2	**274214**	**275518**	**274715**	4.2264	2.00
4	**274222**	**275746**	**274987**	2.1674	3.91
8	274850	276786	275767	1.1175	7.58

Table 3: Results of the instance Tho150

Number of processors	Fitness values ($\times 10^6$) Min	Max	Avg	Average time(s)	Speedup
1	8.25044	8.35089	8.30671	19.2875	N/A
2	**8.21229**	**8.32513**	**8.26641**	9.6225	2.00
4	8.26528	8.40193	8.31295	4.8431	3.98
8	8.30217	8.46206	8.38669	2.6422	7.30

Table 4: Results of the instance Tai256c

Number of processors	Fitness values ($\times 10^7$) Min	Max	Avg	Average time(s)	Speedup
1	4.50920	4.53371	4.52195	55.8196	N/A
2	**4.49573**	4.54016	**4.51117**	27.9000	2.00
4	**4.49847**	4.54544	**4.51770**	14.1247	3.95
8	**4.49508**	4.53848	**4.51520**	7.2950	7.65

4.3.2 Fitness values as the stopping criterion

we also repeat the simulation 50 times for each case. We use the *hit rate* (the percentage of independent runs finding the specified quality of solution) to compare the robustness between the parallel CRO and its sequential one. Both the minimum and average fitness values of sequential CRO recorded in previous simulations are used as the stopping criterion, and we also stop the simulation when the solution is found without improvement for 50000 function evaluations. Notice that we also re-run the sequential CRO and use the same stopping criterion as for PCRO in this part.

We show the *hit rate* for each scenario in Tables 5, 6, and 7. It is obvious that the *hit rate* significantly increase for PCRO when compared with sequential CRO. In other words, PCRO has a strong robustness and is much more stable than the sequential CRO. Particularly in Wil100 and Tho150, the *hit rate* decreases with the number of processors, but are still much higher than the sequential CRO. However in Tai256c, the *hit rate* is more or less the same for the PCRO with various number of processors. We can also see that PCRO fits the Tai256c the best of all the three instances, where it can almost reach 100% *hit rate* when the stopping criterion is the average fitness value, and 80% *hit rate* for the minimum fitness value. In general, PCRO with multiple processors can search larger solution space than CRO during the same time, and this can help PCRO escape from the local optimums. Hence, Parallel CRO performs better than the sequential one in terms of the solution quality.

Since PCRO has a much higher *hit rate*, which means it searches much more of the solution space than the sequential CRO, it is not fair to use the speedup here as a metric. Instead, we also list the average execution time among those with hit for each case in Tables 5, 6, and 7. It can be observed that the run time decreases with the number of processors though the reduction is not as much as that in the previous section. Therefore, PCRO can not only improve the quality of the solution, but also reduce the computing time. We conclude that the efficiency could be consistently improved with the number of processors, provided PCRO is well designed.

Table 5: Execution time and hit rate for the instance Wil100

Number of processors	Average fitness value Avg_time(s)	Hit rate	Minimum fitness value Avg_time(s)	Hit rate
1	4.0427	46%	6.7580	4%
2	2.0944	96%	3.0889	58%
4	1.5326	90%	2.2436	54%
8	1.0589	70%	1.9386	26%

Table 6: Execution time and hit rate for the instance Tho150

Number of processors	Average fitness value		Minimum fitness value	
	Avg_time(s)	Hit rate	Avg_time(s)	Hit rate
1	8.5908	42%	20.3750	2%
2	6.4123	98%	13.0432	86%
4	5.0564	98%	9.7831	68%
8	3.7791	86%	6.1611	32%

Table 7: Execution time and hit rate for the instance Tai256c

Number of processors	Average fitness value		Minimum fitness value	
	Avg_time(s)	Hit rate	Avg_time(s)	Hit rate
1	29.7378	36%	40.5310	2%
2	16.5766	98%	29.9550	80%
4	11.7328	98%	16.3001	78%
8	6.4766	100%	10.4179	86%

5. Conclusions

With the advancement of computers and networking, parallel and distributed computing becomes popular and we can find its application in almost every field of science and technology. Thus, it is urgent for us to parallelize optimization algorithms. We are particularly interested in CRO because of its capability in solving difficult optimization problems [4]. The contributions of this paper can be summarized as follows: 1) We propose the first parallel model for CRO (PCRO), which is designed according to the intrinsic characteristics of CRO and parallelized in a synchronized manner. 2) Simulations are performed for three QAP instances with large problem sizes. We compare PCRO with the canonical sequential CRO. To have comprehensive comparison, two stopping criteria are used in our simulations. The results show that PCRO can consistently improve the quality of solution as well as reduce the computation time with more and more processors involved.

For future work, first of all, we plan to design and try more molecule communication patterns, such as exchanging central energy and kinetic energy. Secondly, taking into account the heterogeneous environment, asynchronous transfer will be employed and the algorithm needs to be re-designed accordingly. Thirdly, in order to extend the application of PCRO, we need to test it on other types of problems. Finally, we try to study the PCRO analytically.

Acknowledgment

This research is supported in part by the University of Hong Kong Strategic Research Theme of Information Technology.

References

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: Univ. of Michigan Press, 1975.

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

[3] E. Bonabeau, M. Dorigo and G. Theraulaz, "Inspiration for optimization from social insect behavior," *Nature,* vol. 406, pp. 39-42, Jul. 2000.

[4] A. Y. S. Lam and V. O. K. Li, "Chemical-Reaction-Inspired metaheuristic for optimization," *IEEE Trans. Evol. Comput.*, accepted for publication.

[5] F. Glover and M. Laguna, *Tabu Search.* Boston, MA: Kluwer Academic Publishers, 1997.

[6] G. Dueck and T. Scheuer, "Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing," *J. of Computational Physics,* pp. 161-175, Sept. 1990.

[7] D. H. Wolpert and W. G. Macready, "No free lunch theorems for search," *IEEE Trans. Evol. Comput.*, vol. 1, no. 1, pp. 67-82, Apr. 1997.

[8] J. Xu, A. Y. S. Lam, and V. O. K. Li, "Chemical reaction optimization for the grid scheduling problem," *IEEE Int'l Conf. on Commun. (ICC2010)*, accepted for publication.

[9] A. Y. S. Lam, J. Xu, and V. O. K. Li, "Chemical reaction optimization for population transition in peer-to-peer live streaming," *IEEE Congress on Evolutionary Computation (2010 IEEE World Congress on Computational Intelligence)*, accepted for publication.

[10] G. Manimaran and C. Siva Ram Murthy, "A fault-tolerant dynamic scheduling algorithm for multiprocessor real-time systems and its analysis," *IEEE Trans. Parallel Distrib. Syst.* vol. 9, no. 11, pp. 1137-1152, Nov. 1998.

[11] Y. Fukuyama and H. Chiang, "A parallel genetic algorithm for generation expansion planning," *IEEE Tran. Power Syst.*, vol. 11, no. 2, pp. 955-961, May 1996.

[12] K. Krishna, K. Ganeshan, and D. J. Ram, "Distributed simulated annealing algorithms for job shop scheduling," *IEEE Trans. Syst. Man Cybern.*, vol. 25, no. 7, Jul. 1995.

[13] A. W. McNabb, C. K. Monson, and K. D. Seppi, "Parallel PSO using MapReduce," *IEEE Congress on Evolutionary Computation (CEC'07)*, 2007.

[14] 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. Operational Res.*, vol. 176, no. 2, pp. 657-690, Jan. 2007.

[15] S. Shani and T. Gonzalez, "P-complete approximation problems," *Journal of the ACM*, vol. 23, no. 3, pp. 555-565, Jul. 1976.

[16] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study.* Princeton, NJ: Princeton Univ. Press, 2006.

[17] S. Martello, D. Pisinger, and D. Vigo, "The three-dimensional bin packing problem," *Operational Research,* vol. 48, no. 2, pp. 256-267, Mar. 2000.

[18] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY: W. H. Freeman & Co Ltd, 1979.

[19] R. E. Burkard, E. Cela, S. E. Karisch, and F. Rendl. (2010) QAPLIB Home Page. [Online]. Available: http://www.seas.upenn.edu/qaplib/

[20] L. M. Gambardella, E. D. Taillard, and M. Dorigo, "Ant colonies for the quadratic assignment problem," *The Journal of the Operational Research Society,* vol. 50, no. 2, pp. 167-176, Feb. 1999.

[21] MPICH2: High-performance and Widely Portable MPI. [Online]. Available: http://www.mcs.anl.gov/research/projects/mpich2/index.php

[22] E. Alba, *Parallel Metaheuristics: A New Class of Algorithms.* Wiley-Interscience, 2005, ch. 2.