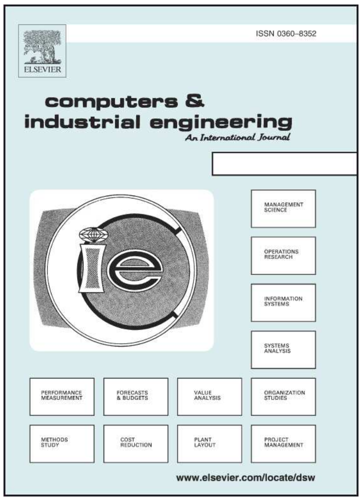
A Novel Chemical Reaction Optimization for the Distributed Permutation Flowshop Scheduling Problem with Makespan Criterion

Hafewa Bargaoui, Olfa Belkahla Driss, Khaled Ghedira



PII: S0360-8352(17)30322-4
DOI: http://dx.doi.org/10.1016/j.cie.2017.07.020
Reference: CAIE 4827

To appear in: *Computers & Industrial Engineering*

Received Date: 5 December 2016
Revised Date: 17 June 2017
Accepted Date: 15 July 2017

Computers & Industrial Engineering

Hafewa Bargaoui
Institut Supérieur de Gestion de Tunis,
Université de Tunis,
41, Avenue de la Liberté,
Cité Bouchoucha, Bardo, Tunis, Tunisie
E-mail: hafewa.bargaoui@gmail.com
Phone: +216 95 587 436

A novel Chemical Reaction Optimization for the Distributed Permutation Flow Shop Scheduling Problem with Makespan Criterion

Hafewa Bargaoui

Institut Supérieur de Gestion de Tunis, Université de Tunis,

41, Avenue de la Liberté, Cité Bouchoucha, Bardo, Tunis, Tunisie

Systems for Smart Optimization and Intelligent Engineering (SSOIE)

Olfa Belkahla Driss

Ecole Supérieure de Commerce de Tunis,

Université de la Manouba, Tunis, Tunisia

Systems for Smart Optimization and Intelligent Engineering (SSOIE)

Khaled Ghedira

Institut Supérieur de Gestion de Tunis, Université de Tunis,

41, Avenue de la Liberté, Cité Bouchoucha, Bardo, Tunis, Tunisie

Systems for Smart Optimization and Intelligent Engineering (SSOIE)

A Novel Chemical Reaction Optimization for the Distributed Permutation Flowshop Scheduling Problem with Makespan Criterion

Abstract

The Permutation Flowshop Scheduling Problem (PFSP) is among the most investigated scheduling problems in the fields of Operational Research (OR) and management science. During the last six decades, it has gained much attention and interest thanks to its applicability in a variety of domains such as industrial engineering and economics. Recently, the PFSP with multi-factory environment has been proposed in shop scheduling sphere. Since the problem is known to be NP-hard, exact algorithms can be extremely costly, computationally speaking. Chemical Reaction Optimization (CRO) is lastly proposed by Lam & Li (2010) to optimize hard combinatorial problems. Due to its ability to escape from local optima, CRO has demonstrated excellent performance in solving a variety of scheduling problems, such as flexible job-shop scheduling, grid scheduling, network scheduling etc. In such a paper, we address the Distributed Permutation Flowshop Scheduling Problem (DPFSP) with an artificial chemical reaction metaheuristic which objective is to minimize the maximum completion time. In the proposed CRO, the effective NEH heuristic is adapted to generate the initial population of molecules. Furthermore, a well-designed One-Point (OP) crossover and an effective greedy strategy are embedded in the CRO algorithm in order to ameliorate the solution quality. Moreover, the influence of the parameter setting on the CRO algorithm is being investigated on the base of the Taguchi method. To validate the performance of the proposed algorithm, intensive experiments are carried out on 720 large instances which are extended from the well known Taillard benchmark. The results prove the efficiency of the proposed algorithm in comparison with some powerful algorithms. It is also seen that more than 200 best-known solutions are improved.

Keywords: Scheduling, Distributed permutation flowshop, Chemical reaction optimization, Evolutionary algorithm

1. Introduction

The scheduling problem can be defined as allocating the available production resources over the time to perform a collection of tasks Baker (1974). It can be applied in a myriad of domains including production planning, manufacturing system, logistics and computer design. During the last six decades, scheduling problems have captured the interest of a considerable number of researchers and a great deal of literature has been published on the latter field. One of the most competing combinatorial optimization problems in scheduling area is the flow shop. It can be formally described as follows: a set N of n independent jobs has to be processed. Each job j, $j \in N$ has to visit in order all m machines in the M set. Precisely, the technological order for any job will be given by $Machine_1$, $Machine_2$, ..., $Machine_m$. At one time, each machine can process one job at most and each job can be processed on one machine at most. Preemption is not allowed, i.e., once an operation is started, it must be completed before another one can be started on that machine. Each operation from a job j requires a given processing time on each machine i. The processing time is positive, fixed, known in advance and denoted as $p_{i,j}$. The objective is to find a schedule of the n jobs on the m machines so that a given criterion is optimized. The objective on which the majority of literature focused is the minimization of the makespan or the maximum completion time, which is denoted as C_{max}. For such a problem, there are generally $(n!)^m$ different alternatives for job permutations on machines. However, most of researches have focused on a common simplification of the problem that is called the Permutation Flowshop Scheduling Problem (PFSP). The PFSP can be considered as a classical flow shop problem when adding the assumption of forbidding job passing which means that the same job permutation is maintained throughout all machines. Due to this simplification, the solution space is reduced to $n!$. Using the well-known three fields notation of Graham et al. (1979), the PFSP with makespan criterion is denoted as $F|prmu|C_{max}$.

Algorithms for classical PFSP have been deeply studied. In fact, since the seminal paper of Johnson (1954) the problem has gained much attention due to its importance in both manufacturing systems, operations management and industrial process. Many previous studies were interested in exact methods such as the parallelization of the well-known Branch and Bound (*B&B*) algorithm by using a grid of computers Jemni et al. (2011). However, the difficulty and the huge computation time of exact algorithms have led researchers to focus their energies on developing approximate methods (heuristics and metaheuristics) which provide high quality solutions to large size problems in a reasonable time. Some early heuristics include the CDS procedure of Campbell et al. (1970) as an extension of the Johnson algorithm and the NEH algorithm of Nawaz et al. (1983) which is still regarded as one of the best heuristics for the PFSP.

In the second group of approximation methods, we find metaheuristics which have proven their efficiency to solve difficult optimization problems. Hence, a myriad of metaheuristics have been proposed to tackle PFSP, such as the combination between ant colony algorithm and local search procedure Ahmadizar & Barzinpour (2010), the hybrid discrete artificial bee colony algorithm (HD-ABC) Liu & Liu (2013), the tabu search algorithm (M.A.F.S.P) Bargaoui & Driss (2014) or the self-guided differential evolution with neighborhood search (NS-SGDE) Shao & Pi (2016) to name just a few.

Nowadays, thousands of papers have been published dealing with the problem and several reviews have been published. In Ruiz & Maroto (2005) 53 articles were presented, dealing with heuristic procedures for the PFSP with makespan objective. A total of 18 heuristics and 7 metaheuristics were tested. The results indicated that the NEH algorithm Nawaz et al. (1983) resulted in the best performance among the constructive heuristics, and the Iterated Local Search (ILS) method showed better performance than the other metaheuristics evaluated. Gupta & Stafford (2006) have also presented an overview of the evolution of the PFSP over five decades (1954-2004). More recently, Fernandez-Viagas et al. (2017) presented a comprehensive review and computational evaluation of the flow shop literature in the last 10 years where a total of 19 heuristics and 12 metaheuristics are compared.

As can be seen, the literature on the PFSP is very vast and the problem has been widely discussed. Namely, in most studies of the PFSP a common assumption is that all jobs are processed in the same factory, i.e., there is only one production center. This fact restricts the use of the PFSP when it is employed to model real-life problems. In fact, with the development of economic globalization, the distributed manufacturing is more and more common since ‘it allows companies to achieve higher product quality, lower production costs and lower management risks’ Kahn (2012). It is therefore of a paramount importance to study the scenarios that include distributed scheduling systems. Recently, a new generalization of the PFSP was put forward in shop scheduling sphere. This generalization has been introduced by Naderi & Ruiz (2010) and called the Distributed Permutation Flowshop Scheduling Problem (DPFSP). In a nutshell, the DPFSP extends the regular PFSP in the following way: a set of jobs need to be processed over a set of identical factories, each one contains a number of machines arranged as a flow shop.

In this paper, we propose to solve the distributed permutation flowshop scheduling problem with makespan criterion using a Chemical Reaction Optimization (CRO) metaheuristic. Motivated by the effectiveness of this newly emerging evolutionary algorithm in solving different kinds of optimization problems, this research, presents the first reported work of the CRO for solving the DPFSP. Our intention is to provide an adaptation of the different parameters of CRO to suit the problem characteristics. Hence, we propose four powerful neighborhood structures to balance exploration and exploitation abilities of the algorithm. Furthermore, we investigate on the influence of the different parameters by using the Taguchi method, and the best parameter setting is suggested. 720 benchmark instances have been used to evaluate the performance of the proposed CRO. The experimental results prove that the proposed CRO algorithm is both effective and efficient.

The rest of the paper is organized as follows. In Section 2, the distributed permutation flowshop scheduling problem is formally defined. Then, a comprehensive literature review on the problem is provided in section 3. In Section 4, the original framework of the chemical reaction optimization algorithm is presented in details. Section 5 introduces the proposed chemical reaction optimization to solve the DPFSP with makespan criterion. Parameter setting and numerical testing results are provided in Section 6. Conclusions are finally drawn in section 7, along with recommendations for future researches.

2. The Distributed Permutation Flowshop Scheduling Problem

The DPFSP problem addressed in this study may be formally described as follows Naderi & Ruiz (2010): "a set N of n jobs must be processed on a set G of F identical factories. Each factory contains the same set M of m machines. Each job consists of m operations, which need to be processed one after another in the same factory. The processing time of job j on machine i at factory f is denoted as $P_{j,i}$". In addition to the assumptions made for the regular PFSP, the following 3 assumptions regarding the DPFPS should be considered:

- The processing times do not vary from factory to factory.
- Once a job is assigned to a factory it cannot be conveyed to another one (all its operations must be processed in the same factory).
- All factories are able to process all jobs.

The objective sought in this work is the minimization of the makespan (the maximum makespan among all the F factories or simply C_{max}). Following the well-known notation of Graham et al. (1979) the DPFSP with makespan criterion is denoted as $DF|prmu|C_{max}$.

The PFSP is a notoriously hard problem in combinatorial optimization field. Although even modest-sized instances remain computationally intractable.

However, scheduling in a distributed environment is more complicated than in a single production center. In a centralized environment a job schedule for each set of machines has to be defined, while in distributed scheduling problem, there are two interdependent decisions to be made:

1. assigning jobs to the different factories.
2. determining the sequence of jobs at each factory.

Naderi & Ruiz (2010) demonstrated that in a DPFSP with makespan objective, the total number of the feasible solutions is: $\binom{n-1}{F-1}n!$ given that there are F factories and n jobs and that the number of jobs is strictly superior than the number of factories. Hence, we can remark that the number of possible solutions of DPFSP is significantly greater than that of the regular PFSP, which is *n!*. Additionally, the DPFSP reduces to the regular PFSP if the number of factories ($F = 1$), and this latter problem is known *NP-Hard*, we can easily conclude that the DPFSP is also an *NP-Hard* problem.

The following example will aid clarifying the representation of the DPFSP. Given an instance with five jobs, two machines and two factories, the processing time matrix is given in Table 1 (let us remind that factories are identical). A feasible Gantt chart of this problem is shown in Figure 1.

Table 1: Processing times for the example

Machines	Jobs				
	1	2	3	4	5
1	10	8	5	3	8
2	8	2	8	11	4

3. Related works

Compared to the PFSP, the literature on the DPFSP is relatively limited. Naderi & Ruiz (2010) presented the first attempt to solve the problem. They proposed six different alternative Mixed Integer Linear Programming (MILP)

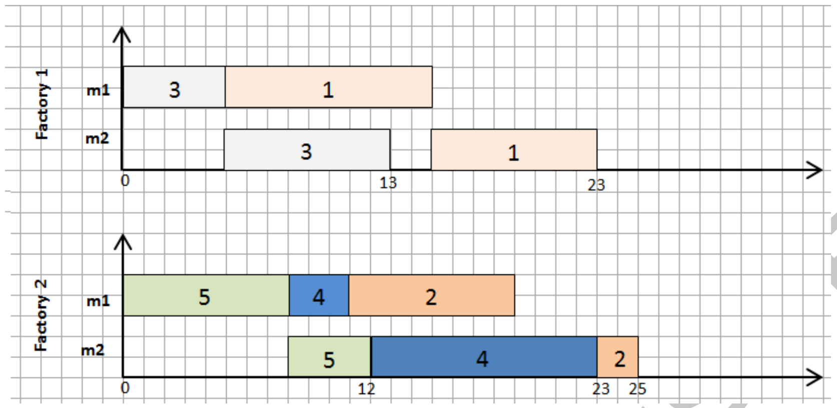


Figure 1: The Gantt Chart of the DPFSP

models which have been implemented on highly optimized CPLEX11.1 package. From the experimental results, it can be seen that only instances with 16 jobs and 4 factories were solved. Authors also presented two simple factory assignment rules which are the following:

- Attribute the job j to the factory with the lowest makespan, not including the job j.
- Attribute job j to the factory which completed it at the earliest time, i.e., the factory that has the lowest makespan after processing the job j.

These two factory assignment rules are used to implement 12 different heuristics based on 6 existing heuristics from the PFSP literature. Computational results indicate that the NEH algorithm Nawaz et al. (1983) with the second job to factory assignment rule (denoted by NEH2) outperforms all other heuristics. Naderi & Ruiz (2010) also proposed two different simple variable neighborhood descent heuristics referred to as VND(a) and VND(b). Computational experiments indicated that VND(a) is more effective than all other heuristics, though it requires longer computational time.

After the publication of the seminal paper of Naderi & Ruiz (2010), the DPFSP have attracted the attention of researchers and many papers have been recently presented to tackle the problem. The first attempt was suggested by

Liu & Gao (2010). Authors proposed a complex discrete electromagnetism algorithm which we simply refer to as EM. To improve the performance of the EM, a Variable Neighborhood Search (VNS) composed of four types of neighborhood has been incorporated. In their computational evaluation, authors did not directly compare their EM algorithm against any algorithm from Naderi & Ruiz (2010) but only indicated that they improved 151 solutions of the benchmark presented by Naderi & Ruiz (2010).

Next, in 2011 a genetic algorithm allied with a local search procedure called GA_LS Gao & Chen (2011) has been presented. The algorithm uses NEH2, VND(a) and random solutions to generate the initial population. Authors also proposed an efficient local search method which includes mechanisms of insertion and exchange jobs to ameliorate the solution quality. Computational evaluation show that GA_LS outperforms VND(a) but it requires larger computational times.

Later, Gao et al. (2013) have proposed a tabu search (TS) metaheuristic. The algorithm is initialized by NEH2 heuristic in order to accelerate the optimization process. The proposed algorithm builds upon the three job movement methods presented in Gao & Chen (2011): Insertion_Jobs, Exchange_Jobs and Move_Jobs and it includes several enhanced local search schemes based also on the principle of insertion and exchange of jobs. Computational analysis show that the tabu search outperforms the GA_LS algorithm of Gao & Chen (2011). However, comparisons on CPU runtimes indicate that TS is about 117 times slower than the VND(a).

In the same year, Lin et al. (2013) have proposed a modified iterated greedy algorithm (MIG) to solve the DPFSP with makespan criterion. The authors employed NEH2 heuristic as initialization. The MIG algorithm applies two phases iteratively; a destruction phase, where some jobs are removed from previously constructed complete candidate solution, and construction phase, where the eliminated jobs are reinserted into the sequence. The number of removed jobs used in the destruction phase is variable and chosen between two bounds (min=2 and max=7). The authors also employed a novel acceptance criterion which is commonly used in simulated annealing in order to yield better quality solutions. However, they do not incorporate any local search mechanism to ameliorate the solution.

Wang et al. (2013) implemented an estimation of distribution algorithm (EDA). Basically, the authors design a probability model to describe the probability distribution of the solution space. They also defined a mechanism to update the probability model, at each generation, with superior individuals. The new individuals are generated by sampling the probability model to track the promising search regions. However, due to the limited exploitation capability of EDA, authors presented four different local search operators in order to improve the solutions and enhance the exploitation. In the experimental section, EDA was only compared with the algorithms presented in the seminal paper of Naderi & Ruiz (2010). Furthermore, recent paper Fernandez-Viagas & Framinan (2015) demonstrates that the performance of EDA is below TS and MIG metaheuristics with nearly 6% relative percentage deviation.

In 2014, Naderi & Ruiz (2014) have presented a scatter search (SS) metaheuristic. The algorithm is based on the principles of solution generation, improvement recombination and limited use of randomization. It also incorporates a local search phase which is inspired from the VND heuristics of Naderi & Ruiz (2010). In the computational experiments, SS method is shown to outperform existing algorithms except for instances with F = 7 where the modified iterated greedy algorithm Lin et al. (2013) obtains better solutions.

Recently, Fernandez-Viagas & Framinan (2015) have proposed a bounded search iterated greedy algorithm referred to as BSIG. The algorithm is composed of three main phases: an iterative destruction and construction of the solution are carried out and finally an improvement phase where three efficient local search schemes are incorporated in order to improve the solutions. In their computational analysis, BSIG is shown to outperform the TS, EDA and MIG algorithms by a significant margin.

More recently, Komaki et al. (2015) have presented a general variable neighborhood search algorithm referred to as GVNS. The GVNS is composed of 2 phases: the first one is the shake phase which aims to escape from local search optimum trap, while the second phase is local search which is used to search the neighborhood of the current solution in order to find a better solution. The algorithm employs the NEH2 heuristic as initialization. Then, a shake procedure is included. The severity of the shaking procedure depends on the status of the current solution. Hence, if the algorithm is able to improve the current solution a slight perturbation (intensification) is applied. Otherwise, if the algorithm can't improve the current solution for a predefined number of iterations, a stronger shaking procedure will be applied. For the local search phase, authors proposed four neighborhood structures based on swap and insertion within a factory and across factories. In the computational experiments, GVNS is compared against the TS, EDA, IG, VND(a) and BSIG. The results favor the GVNS in comparison with TS, EDA, IG, VND(a). However, in average, BSIG has the best performance among all the tested algorithms.

As we can see, in the last years, many metaheuristics have been applied to solve DPFSP. Computational experiments and statistical analysis have shown that some algorithms produce better solutions than others. Hence, adopting and testing new metaheuristics remains a great challenge and an important way towards reaching, or hope to approach, optimal solutions. Based on the difficulty of the problem and the main characteristics of the chemical reaction optimization, we deem that this newly established metaheuristic seems to be an attractive way to tackle difficult optimization problems. In the next section, we present an overview of CRO algorithm as well as the main motivation to employ it as a solution for the DPFSP with makespan criterion

4. Chemical Reaction Optimization

Evolutionary computation techniques have received a great deal of attention regarding their effectiveness in solving combinatorial optimization problems. Evolutionary algorithms involve a number of artificial creatures which search over the space of the problem in order to discover optimal areas. Recently, Lam

& Li (2010) introduced a novel inspired metaheuristic algorithm, named Chemical Reaction Optimization (CRO). It loosely mimics what happens to molecules in a chemical reaction system i.e., molecules incessantly alter in an attempt to attain the lowest free energy. CRO is a population-based evolutionary metaheuristic hence, molecules represent the population individuals and chemical reactions correspond to the different operators. Each molecule is presented by a set properties (see Table 2). The representation of a molecular structure (ω) is problem-dependent and it expresses a feasible solution. Each molecule has two types of energies, i.e., potential energy and kinetic energy. The potential energy (PE) represents the objective function value of the corresponding solution ω while the kinetic energy (KE) represents the tolerance of the system to accept a worse solution (i.e., a solution with higher functional value). In order to switch from one feasible solution to another one in the search space, four elementary reaction operators are defined, i.e., (1) the on-wall ineffective collision, (2) decomposition collision, (3) inter-molecular ineffective collision, and (4) synthesis collision. These four reactions can be classified into single molecular collisions and multiple molecular collisions. The on-wall ineffective collision and decomposition are single molecular collisions (i.e., they involve one single molecule), while the inter-molecular ineffective collision and synthesis collision are of the second category (i.e., they involve two or more molecules).

Similar to other evolutionary algorithms, CRO consists of three essential steps: firstly the initialization then the iterations and finally the final stage. During the first step, the different attributes of the algorithm are initialized. These attributes are presented below:

- *PopSize*: the population size, i.e., initial number of solutions maintained.
- *MoleColl*: a number varying between 0 and 1 which decides whether a unimolecular or inter-molecular collision will take place in the next iteration.
- *InitialKE*: the initial kinetic energy assigned to the initial set of molecules.
- *KELossRate*: represents the maximum percentage of kinetic energy lost

Table 2: Profile of a molecule in CRO Lam & Li (2010)

Terminology	
Chemical sciences	Metaheuristic meaning
Molecular Structure (ω)	Solution
Potential energy (PE)	Objective function value
Kinetic Energy (KE)	Measuring tolerance of having worse solutions
Number of Hits (*NumHit*)	Current total number of moves
Minimum structure	Current best found solution
Minimum value	best function value
Minimum hit number (*MinHit*)	Moves number at current best found solution

by an individual molecule during the reaction ($0 \leq KELossRate \leq 1$).

- *Buffer*: the energy stored in the buffer.
- α and β: two parameters that control the ratio between intensification and diversification.

During the iteration phase the algorithm explores the solution space through a sequence of four different chemical reactions. In fact, molecules in a reaction system tend to collide with each other or with the walls of the container. These collisions trigger the change in the molecule. The change can be subtle (intensification) or vigorous (diversification). The intensification (exploitation) is the process that tends to search in the neighborhood of the current solution to improve the quality through a subtle perturbation while diversification (exploration) tends to search different regions of the solution space through a major perturbation. The reader is invited to Črepinšek et al. (2013) for a detailed survey on intensification techniques in evolutionary algorithms. In the CRO algorithm exploitation is carried out through the on-wall ineffective collision and the intermolecular ineffective collision while exploration is carried out through decomposition and synthesis. Each elementary reaction has its own characteristic features which are detailed below:

1. **On-wall ineffective collision:** this reaction occurs when a molecule ω hits the wall of the container and then bounces back remaining in one new unit ω'. After this reaction, the molecular structure and the potential energy of the molecule undergo just a small change. In this collision, only a slight perturbation is made to the existing molecule ω, i.e., $\omega \longrightarrow \omega'$.
2. **Decomposition:** this reaction takes place when a molecule ω hits the wall of the container and then decomposes into several parts (suppose two ω'_1 and ω'_2 in our implementation), i.e., $\omega \longrightarrow \omega'_1 + \omega'_2$. As this collision is vigorous, the molecular structures of ω'_1 and ω'_2 have greater differences from that of ω. The main idea of decomposition is to explore other regions corresponding to ω'_1 and ω'_2 after enough local search by the on-wall ineffective collision. In fact, if the selected molecule ω has stayed in a stable state for a predefined period: NumHit - MinHit $> \alpha$ then the system undergoes a decomposition reaction. Otherwise, the system triggers an on-wall ineffective collision.
3. **Intermolecular ineffective collision:** this reaction occurs when two molecules ω_1 and ω_2 collide with each other and then separate. The number of molecules involved in this reaction remains unchanged before and after the process: there will produce two new molecules ω'_1 and ω'_2, i.e., $\omega_1 + \omega_2 \longrightarrow \omega'_1 + \omega'_2$. The intermolecular ineffective collision is very similar to the on-wall ineffective collision since it is not vigorous and the new produced molecules are in the neighborhood of ω_1 and ω_2.
4. **Synthesis:** this reaction occurs when two molecules ω_1 and ω_2 hit against each other and combine together to form one new single molecule ω', i.e., $\omega_1 + \omega_2 \longrightarrow \omega'$. The change is vigorous and the molecule ω' is in a new region different from the search regions around ω_1 and ω_2. In fact, if the selected molecules have KE less than or equal to β it means that they lost the flexibility of escaping from local minima. Hence, the

system triggers a synthesis reaction in order to bring those molecules to other solution regions for diversification. Otherwise, the system triggers an inter-molecular ineffective collision.

The CRO algorithm undergoes these four elementary reactions until the satisfaction of the stopping criteria. Finally, it terminates and outputs the best found solution. For more details about the CRO metaheuristic, the interested reader is invited to refer to (Lam & Li (2012) and Lam et al. (2013)).

Our main motivation of using CRO is determined by several factors. Firstly, the CRO has been widely used to solve many optimization problems in different disciplines. It has been successfully applied to the resource constrained project scheduling problem Lam & Li (2010), the 01 knapsack problem Truong et al. (2013), the set covering problem James et al. (2014) and the flexible job-shop scheduling problems (Li & Pan (2012), Marzouki & Driss (2015)) to name just a few. In addition, experiment results show that CRO has better performance when compared with other powerful metaheuristics such as Genetic Algorithm (AG), Tabu Search (TS) and Simulated Annealing (SA). Furthermore, it is worth noting that evolutionary algorithms have proven their ability to explore huge search spaces, however they are comparatively poor at exploiting the promising regions where the algorithm converges to. CRO overcomes such serious weakness thanks to its different elementary reaction operators that establish a good trade-off between intensification (through on-wall ineffective collision and inter-molecular ineffective collision) and diversification (through decomposition and synthesis). What is also interesting, CRO examines not just one solution, but a pool of probable solutions simultaneously, which are organized as molecules and form a population. CRO incrementally generates new molecules by iteratively applying different operators. This diversity of operators helps one to customize the algorithm to suit different problems. Based on the above-mentioned benefits, it is believed that CRO can accomplish satisfactory improvement for DPFSP with makespan criterion.

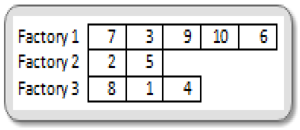


Figure 2: An example of the solution representation.

5. The proposed CRO algorithm for DPFSP

In order to adapt the chemical reaction optimization algorithm to solve the DPFSP, a number of elements have to be defined:

1. The way in which a solution (a molecule) should be represented.
2. How to generate the initial population of solutions.
3. Which operator design we should use for each elementary reaction in order to explore the search space.

In what follows, we present the design of all these elements.

5.1. Solution representation

Solution representation is a crucial issue for every optimization algorithm. When coding a solution of PFSP, permutation of n jobs is the most common representation. However, in the DPFSP jobs are assigned to the F available factories, i.e., the permutation of jobs is divided among the factories. Hence, the above representation does not suffice. In our implementation, each molecule is represented by a two-dimensional array which consists of F rows, one per factory. Each row consists of a partial sequence of jobs, indicating the order in which they passed at a given factory. For example, we consider a DPFSP with ten jobs and three factories, a possible solution of the problem is given in Figure 2. The operation sequences of jobs for factory 1, 2 and 3 are {7-3-9-10-6}, {2-5}, and {8-1-4}, respectively.

5.2. Initial solution

It has been shown that the effectiveness of the approaches for solving high-combinatorial problems depends heavily on the quality of the initial solution.

Based on the review of flow shop heuristics by Ruiz & Maroto (2005), the NEH heuristic Nawaz et al. (1983) is commonly regarded as the best performing heuristic for the PFSP with makespan criterion. Naderi & Ruiz (2010) have proposed an adaptation of the NEH heuristic to solve the new generalization of the PFSP. In their work, experimental results showed that the solution quality of NEH heuristic with the second rule for job assignment to factory (NEH2) provides the best results among all the simple constructive heuristics. The NEH2 heuristic can be described by the following two steps:

1. Sort the n jobs by decreasing total processing times on the m machines.
2. Jobs are inserted, one by one and according to the sorted list into all positions of all factories. The job is finally placed in the position with the best (lowest) partial makespan value.

The complexity of NEH heuristic for the regular PFSP is $O(n^3m)$ which can lead to extensive computation times especially for large-scaled instances. However, thanks to the acceleration method of Taillard (1990) it is possible to reduce the complexity of the algorithm to just $O(n^2m)$. In our algorithm we adopt this data structure, as it is rather straightforward to implement.

Since CRO is an evolutionary metaheuristic, it evolves a population of molecules simultaneously. In our algorithm, we start with a PopSize of 40 initial solutions. In 39 of these solutions we use a random job ordering then we apply the three latter steps of the NEH2 method. For the last 40th solution we employ the regular jobs ordering of the NEH2 heuristic.

5.3. Reaction operators

In the following four subsections, we will describe each operator of the proposed CRO algorithm. Note that in order to explain each reaction operator, we consider a DPFSP instance with 10 jobs and 3 factories.

5.3.1. On-wall ineffective collision

In this operator one molecule will be created from one original molecule. It responds to the intensification and it aims to define a perturbation operator as a simple move to obtain ω' in the immediate neighborhood of ω. In our implementation, the on-wall ineffective collision starts by selecting the critical factory i.e., the factory which have the largest makespan value. Then, each job from this factory is randomly selected, removed and then inserted into all possible positions of all the factories (including the critical one). The procedure is repeated until an improvement in the makespan of the schedule is found or all jobs from the critical factory are moved. In the latter case, the job is inserted in the position with the lowest makespan. Note that to accelerate the procedure of evaluation of the whole neighborhood, the acceleration method of Taillard (1990) is employed.

After performing the on-wall ineffective collision, ω can transform to ω' only if the condition given by Eq. (1) is satisfied.

$PE_{\omega} + KE_{\omega} \geq PE_{\omega'}$. (1)

If the above condition holds, that means either a better solution is found ($PE_{\omega} > PE_{\omega'}$), or the original molecule ω has enough KE to compensate PE$_{\omega}$$.Aftercheckingthisconditionsomeattributesof$$(\omega)$ will change, i.e., the current molecular structure ω turns into another state ω' and the molecule loses a certain portion of KE to the central energy *buffer*. Thus, the KE of the new molecule ω_1 is as follows:

$KE\omega'$ = (PEω + KEω - PEω') $\times$ a. (2)

Where $a \in$ [*KELossRate*, 1] and (a-1) represents the fraction of KE lost to the buffer when the molecule hits the wall of the container. By losing a portion of the KE to the *buffer*, the algorithm can escape from its local minimum. The energy in the *buffer* is updated as follows:

$Buffer = Buffer$+($PE\omega$ + KEω - PEω') $\times$ (1 - a). (3)

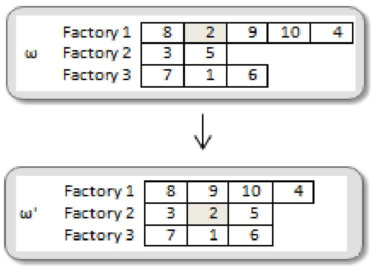


Figure 3: An example of on-wall Operator

It is worth noting that if condition (1) does not hold, the on-wall ineffective collision fails and the molecule retains its original ω, PE and KE. An example of the on-wall operator is illustrated in Figure 3. ω represents the selected molecule from the current population. The sequence of jobs for factory 1, 2 and 3 are {8-2-9-10-4}, {3-5}, and {7-1-6}, respectively. Suppose the critical factory is the first one and the randomly selected job is job {2}. This job is removed from the critical factory and, then, inserted into all possible positions of the three factories. Finally, it is placed in the position that results the lowest makespan which is, in our example, the second one of factory 2.

5.3.2. Decomposition operator

Decomposition refers to the situation when a molecule ω hits the wall of the container and breaks up into two or more pieces (for simplicity we suppose only two ω'_1 and ω'_2). This operator is implemented for diversification and it makes the algorithm explore new regions in the search space after a predefined number of local search iterations (by means of the on-wall operator). In our procedure, the selected molecule ω is broken up into two parts. The first new molecule (ω'_1) inherits the left-sided set of jobs of ω while the second new molecule (ω'_2) inherits the right-sided ones. Then, we obtain two partial solutions. To yield complete solutions we use the NEH2 heuristic in order to insert the non-scheduled jobs in the best positions. Note that the accelerations of Taillard (1990) are also used here.

Two situations should be considered for the decomposition criterion:

- Situation 1. The molecule has sufficient energy (PE and KE) to complete the decomposition, that is

$PE_{\omega} + KE_{\omega} \geq PE_{\omega'_1} + PE_{\omega'_2}$.(4)
If (4) holds the change is allowed and the molecule ω is replaced by ω'_1 and ω'_2. The kinetic energy of the resultant molecules is given by following formulations.

$KE_{\omega'_1} = (PE_{\omega} + KE_{\omega} - PE_{\omega'_1} - PE_{\omega'_2}) \times q_1$.(5)

$KE_{\omega'_2} = (PE_{\omega} + KE_{\omega} - PE_{\omega'_1} - PE_{\omega'_2}) \times (1 - q_1)$.(6)
Where q_1 is a random number uniformly generated from the interval [0, 1].

- Situation 2. The selected molecule has not enough energy to endow the PE of the resultant ones i.e.,

$PE_{\omega} + KE_{\omega} < PE_{\omega'_1} + PE_{\omega'_2}$.(7)
In this case the molecule should obtain energy from the central energy *buffer*. Hence, the energy conservation condition for decomposition becomes as follows.

$PE_{\omega} + KE_{\omega} + buffer \geq PE_{\omega'_1} + PE_{\omega'_2}$.(8)
If (8) holds the change is allowed and the kinetic energy of the resultant molecules is given by following formulations.

$$KE_{\omega'_1} = (PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2} + buffer) \times \mathrm{q}_2 \times \mathrm{q}_3. \ (9)$$

$$KE_{\omega'_2} = (PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2} + buffer) \times \mathrm{q}_4 \times \mathrm{q}_5. \ (10)$$

Where q_2, q_3, q_4, and q_5 are random numbers from the interval [0, 1]. Then, the energy in the *buffer* is updated as follows.

$$buffer = PE_\omega + KE_\omega - PE_{\omega'_1} - PE_{\omega'_2} + buffer - KE_{\omega'_1} - KE_{\omega'_2}. (11)$$

It should be noted that if both (4) and (8) do not hold, the change is prohibited and the molecule maintains its original ω, PE and KE.

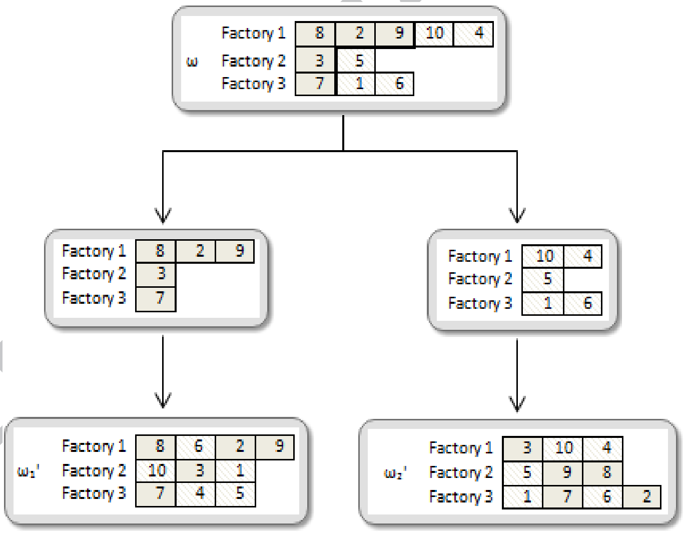


Figure 4: An example of decomposition

An example of decomposition operator is given in Figure 4. The operation sequence of jobs for factory 1 is {8-2-9-10-4} and the sequence of jobs to be processed in the factory 2 is {3-5} while jobs {7-1-6} are processed in the factory 3.

The three random separation points for each factory are 3, 1 and 1 respectively. These points are used to divide job sequences of the molecule ω into two parts: Left and Right. Hence, the set 'Left' is composed of jobs 8, 2, 9, 3 and 7 while the set 'Right' is composed of jobs 10, 4, 5, 1 and 6. The set of jobs on the Left side are inherited from molecule ω to the new molecule ω'_1. Then, the jobs of the 'Right' side are sequentially inserted into, to the best positions in $\omega'1$, using the NEH2 heuristic. The same steps are applied to generate the second new molecule ω'_2, but this time jobs on the 'Right' side are inherited from molecule ω to the molecule ω'_2 accordingly the jobs of the 'Left' side are sequentially scheduled until we obtain a complete solution.

5.3.3. Intermolecular ineffective collision

This operator consists in generating ω'_1 and ω'_2 in the neighborhood of ω_1 and ω_2. The major advantage of this operator is to explore different neighborhoods each corresponding to a molecule. Note that "any mechanism, which can produce ω'_1 and ω'_2 is allowed" Lam & Li (2012).

Ruiz & Stützle (2007), in their inspiring paper, presented a high performing Iterated Greedy (IG) to solve the PFSP with makespan criterion. The algorithm is remarkably simple and easy to implement. The astonishing results of IG in solving the PFSP have encouraged us to incorporate it in our CRO algorithm in the following way. During the destruction phase, the algorithm selects randomly and without repetition a set S of jobs from a complete solution. S is denoted as the size of the destruction phase and it is a random number chosen between two bounds: $S \in$ [Number of factories, Number of jobs/2]. These S jobs are then removed in the order in which they were chosen. It is important to note that we choose one job from each factory and if $S > F$ we randomly select the remaining (S - F) jobs from the unselected jobs. The construction procedure then consists in applying a heuristic in order to sequentially reinsert the removed jobs. In our implementation, we use the three latter steps of the NEH2 algorithm with the accelerations of Taillard (1990) until a complete scheduling of all n jobs is obtained. Note that we apply the above mentioned mechanism to both ω_1 and

ω_2 separately since there is no strict exigencies on the mechanisms of generation of new molecules.

The condition of the intermolecular ineffective collision is given below.

$PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} \geq PE_{\omega'_1} + PE_{\omega'_2}$.(12)

If (12) holds, the kinetic energy of the resultant molecules are calculated as follows.

$KE_{\omega'_1} = (PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'_1} - PE_{\omega'_2}) \times q_6$.(13)

$KE_{\omega'_2} = (PE_{\omega_1} + KE\omega_1$ + PEω_2 + KEω_2 - PE$\omega_1{}'$ - PE$\omega'{}_2$) $\times(1 - q_6)$.(14)

Where q_6 is a random number from the interval [0, 1].

It is important to note that the molecules retain the original ω_1, ω_2, PE_{ω_1}, PE_{ω_2}, KE_{ω_1} and KE_{ω_2} if (12) fails.

Figure 5 outlines this operator. The sequence of jobs for factory 1, 2 and 3 are {8-2-9-10-4}, {3-5}, and {7-1-6}, respectively. In this example we only consider the changes on one molecule ω_1 in order to generate a new molecule $\omega'{}_1$ (the same steps are applied to generate $\omega'{}_2$). In fact, when the molecule ω_1 collides with another molecule from the population it undergoes only subtle changes. Hence, some elements of ω_1 are destructed, i.e., jobs 2, 3 and 6 are sequentially removed. Then, during the construction phase the removed jobs are sequentially reinserted using the NEH2 heuristic until a complete solution of all n jobs is obtained.

5.3.4. Synthesis operator

In this operator, one molecule ω' will be created from two different molecules (ω_1 and ω_2). Synthesis collision gives the effect of diversification (i.e., exploration) and so it allows the algorithm to jump to a distant area in the solution

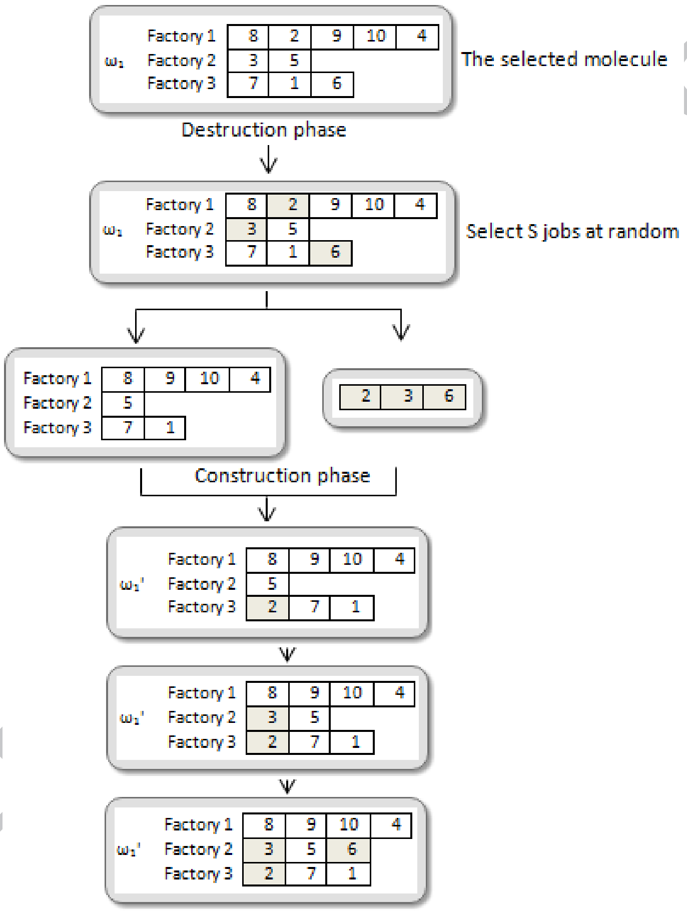


Figure 5: An example of intermolecular ineffective collision

space. In our implementation, we were inspired by the One-Point (OP) crossover operator of genetic algorithm Murata et al. (1996). It consists in selecting separation points randomly for all factories from the first molecule (ω_1). The new molecule (ω') inherits the left-sided set of jobs of (ω_1). The remaining, i.e., not yet scheduled, jobs are sequentially conjoined to the molecule ω' in the same order of appearance as in the molecule (ω_2).

The condition that allows the system to undergo a synthesis reaction is given below.

$PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} \geq PE_{\omega'}$.(15)
Then the KE of the new molecule ω' is given by the following formulation.

$KE_{\omega'} = PE_{\omega_1} + KE_{\omega_1} + PE_{\omega_2} + KE_{\omega_2} - PE_{\omega'}$.(16)
It is noteworthy that if (15) does not hold, the molecules retain the original ω_1, ω_2, PE_{ω_1}, PE_{ω_2}, KE_{ω_1} and KE_{ω_2}.

Figure 6 outlines an example of this operator. The algorithm starts by selecting 3 separation points, one per factory (factory 1 = 3, factory 2 = 1 and factory 3 = 1). The set of jobs in the left side of ω_1, i.e., {8-2-9-3-7}, are inherited from molecule ω_1 to the new molecule ω'. Then, the non-scheduled jobs {10-4-5-1-6} are sequentially conjoined to the molecule ω' in the same order of appearance as in the molecule ω_2. In our example, jobs 5 and 6 are conjoined to the factory 1, job 1 is conjoined to the factory 2, while jobs 10 and 4 are conjoined to the factory 3.

5.3.5. An overview of the proposed CRO for the DPFSP

In this section, we provide a detailed description of the whole process of the proposed CRO (see Algorithm 1). In the initialization, we configure the initial parameters (*PopSize* as the initial number of molecules in the container,

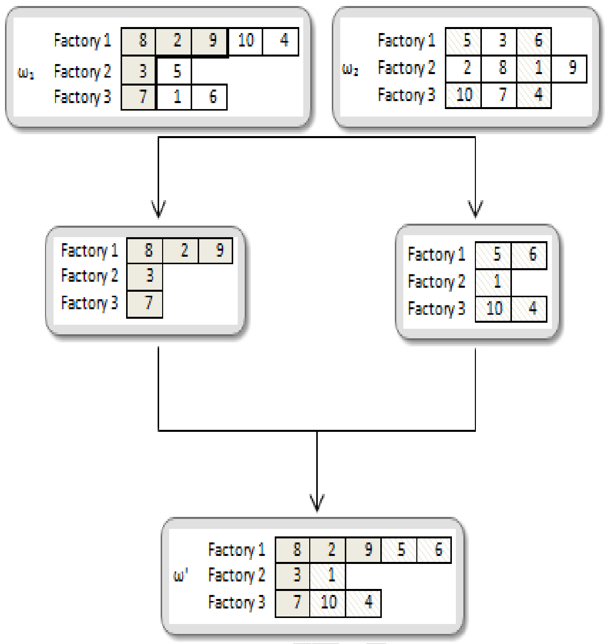


Figure 6: An example of synthesis operator

MoleColl, initial kinetic energy, kinetic energy loss rate, buffer size and the two diversification thresholds (α and β). Then, we create the initial population of molecules. The powerful NEH2 heuristic is combined with random job permutations in order to generate '*PopSize*' solutions with certain quality and diversity. The initial potential energy of each molecule is determined by the corresponding makespan value while its kinetic energy is set to *InitialKE*. Once the initial set of molecules is generated, the CRO algorithm iteratively applies one of the four reactions. Firstly, the algorithm determines whether it is a uni-molecular or inter-molecular collision. This is decided by comparing a random number $b \in [0,1]$ against *MoleColl*.

1. If $b > MoleColl$ or the system only has one molecule, a uni-molecular collision will occur.
   In this case, the system randomly picks one molecule and checks if the decomposition criterion is satisfied.

- If (number of hits - minimum hit number $> \alpha$), which means that the selected molecule has undergone α times of local search without obtaining any new local minimum solution. Hence the molecule should experience a decomposition in order to explore a new region of the solution space. In other words, decomposition takes place suitably between local searches when those latter are not prolific anymore.

- If the decomposition criterion is not verified, the selected molecule undergoes an on-wall ineffective collision.

2. if $b \leq \mathit{MoleColl}$ and the system has more than one molecule, an intermolecular collision will occur.
   In this case, the system randomly selects two different molecules from the population and they are tested against the synthesis criterion.

   - If all involved molecules have KE less than or equal to β the system triggers a synthesis collision. This reaction is another way to escape from local optima. Molecules with too low kinetic energy always lose the flexibility of escaping from local minima. Hence, with a significant structural change, the synthesis transforms inactive molecules into an active one and guides the search to other regions of the solution space.

   - If the synthesis criterion is not verified, the selected molecules experience an intermolecular ineffective collision

The proposed algorithm iteratively applies one of four reactions until the stopping criteria is met. At this step the algorithm ends and outputs the so far best solution as the global optimum.

This completes the description of the proposed CRO algorithm. In the next section, we conduct extensive computational experiments to analyze and validate the efficiency of our model.

Algorithm 1 CRO algorithm

```
1: Input: The objective function
2: Assign values to the CRO attributes: PopSize, InitialKE, KELossRate,
   MoleColl, buffer, $\alpha$, $\beta$
3: Construct the initial population of molecules (with size equal to PopSize)
4: while the stopping criterion not met do
5:   Get b randomly in interval [0, 1];
6:   if b $>$ MoleColl or or PopSize == 1 then
7:     Randomly select one molecule $\omega$ from population;
8:     if If NumHit - MinHit $> \alpha$ then then
9:       $(\omega'_1, \omega'_2$, Success) = Decomposition($\omega$, Buffer);
10:      if  Success == True then
11:        Remove $\omega$ from population;
12:        Add $\omega'_1$ and $\omega'_2$ to population;
13:        PopSize $\leftarrow$ PopSize + 1;
14:      end if
15:    else
16:      $(\omega')$ = On-wall-ineffective-collision($\omega$, Buffer);
17:    end if
18:  else
19:    Randomly select two molecules ($\omega_1$ and $\omega_2$) from population;
20:    if KE$\omega_1 \leq \beta$ & KE$\omega_2 \leq \beta$ then
21:      ($\omega'$, Success) = Synthesis($\omega_1$, $\omega_2$);
22:      if  Success == True then
23:        Remove $\omega_1$, $\omega_2$ from population;
24:        Add $\omega'$ to population;
25:        PopSize $\leftarrow$ PopSize - 1;
26:      end if
27:    else
28:      $(\omega'_1, \omega'_2)$ = Inter-Ineff-Collision($\omega_1$, $\omega_2$);
29:    end if
30:  end if
31:  Verify if there is a new best solution;
32: end while
33: Output:The best found schedule and its makespan value
```

6. Experimental analysis

In this section, we investigate the effectiveness of the proposed CRO algorithm on a set of 720 large-sized instances. The combinations of $n * m$ are {20, 50, 100} * 5, {20, 50, 100, 200} * 10 and {20, 50, 100, 200, 500} * 20 and the number of factories F is from {2, 3, 4, 5, 6, 7}. There are 10 different instances for each combination of parameters. The processing times are distributed uniformly in the interval [1, 99]. The benchmark can be downloaded from: http://soa.iti.es.

To evaluate the performance of our CRO algorithm, same as in literature Naderi & Ruiz (2010), we use the Relative Percentage Deviation (RPD) over the best-known solution for each instance.

$$RPD = \frac{ALG_a - BKS}{BKS} \times 100 \tag{17}$$

Where ALG_a corresponds to the makespan obtained by a given algorithm a and BKS represents the best-known solution available at http://soa.iti.es.

6.1. Setting parameters

As it has been mentioned, the proposed CRO algorithm has some parameters that need tweaking to get the best performance. These parameters are *MoleColl*, *buffer*, *InitialKE*, *KELossRate*, α and β. Our objective is to assign parameter values to CRO with relatively good performance. The values for the first two parameters, *MoleColl* and *buffer*, were deduced from the literature:

- *MoleColl*: a parameter which controls the choice of collision type. We remark that a high value means a higher chance to execute an inter-molecular collision while low value of this attribute leads to a higher chance to perform uni-molecular collision. In our implementation we fixed *MoleColl* value to 0,5 as in Li & Pan (2012) in order to give the same probability for executing the two types of collisions.

- *Buffer*: the initial *buffer* size and the initial kinetic energy determine the total energy in the system before the CRO starts. As explained in section

6.3.1, after on wall ineffective collision a portion of kinetic energy of the involved molecule is withdrawn to the *buffer*. When a molecule cannot improve its search quality (trapped into local minimum), the *buffer* will help it by performing decomposition. In other words, the *buffer* accumulates energy from antecedent on wall ineffective collision then uses it to increase the chance of having a decomposition completed when the energy conservation is not satisfied, i.e., $PE_{\omega} + KE_{\omega} < PE_{\omega'_1} + PE_{\omega'_2}$. However, a decomposition that occurs at the early stage of the search process may hinder the convergence of the algorithm particularly when it involves a molecule that is still evolving (not stuck in local minimum). Therefore, the molecule will switch to a distant region instead of focusing the search in promising local regions. Hence, we assign zero to the initial *buffer* size in order to suppress decomposition reactions in the early stage of the search James et al. (2015).

To investigate in the influence of the four remaining parameters on the performance of the CRO, there are several designs including full factorial experiment and Taguchi experiment. In this paper, the Taguchi method of design of experiments Montgomery (2008) is used in order to decrease the consuming time and the required tests. We used a moderate-sized instance I_5_16_3_4 such as in Wang et al. (2013). Table 3 lists the combinations of different values of these parameters. The number of parameters is 4 and the number of factor levels is 4, too. Since both the number of factor levels and the number of parameters are 4, we employ the orthogonal array $L_{16}(4^4)$. The total number of treatments is 16 and for each combination the CRO is run 20 times independently. The average makespan value is used to calculate the Relative Percentage Deviation (RPD) as presented in Table 4. The response value and the significance rank of each parameter are listed in Table 5.

From Table 5, we can remark that the 'decomposition threshold' α is the most significant of the 4 parameters. In fact, if we have too many decompositions, the number of molecules will increase and the number of function

Table 3: Combinations of parameter values

Parameters	Factor level			
	1	2	3	4
InitialKE	2	4	6	8
KELossRate	0.1	0.2	0.3	0.4
α	100	200	300	400
β	2	4	6	8

Table 4: Orthogonal array and (RPD) values

Experiment no.	Factor				
	InitialKE	KELossRate	α	β	RPD
1	1	1	1	1	1.72
2	1	2	2	2	0.22
3	1	3	3	3	0.58
4	1	4	4	4	0.88
5	2	1	2	3	1.20
6	2	2	1	4	0.85
7	2	3	4	1	0.64
8	2	4	3	2	0.65
9	3	1	3	4	0.74
10	3	2	4	3	1.01
11	3	3	1	2	0.95
12	3	4	2	1	0.12
13	4	1	4	2	0.17
14	4	2	3	1	0.32
15	4	3	2	4	0.36
16	4	4	1	3	0.78

evaluations will be exhausted (a large amount of computational budget) without focusing on the local potential regions. In addition, *InitialKE* ranks the second, which implies that the kinetic energy assigned to the initial population of molecules is also important. Hence, the choice of proper values for *InitialKE* and α may affect the convergence of the algorithm and the possibility of finding the global minimum. Besides, the significant rank of the *KELossRate* is the third. It represents the energy loss rate; it controls the rate of transforming Kinetic energy to the *buffer* during the on-wall ineffective collisions. In fact, when a molecule undergoes an on-wall reaction a portion of its *KE* will be divided into two parts. One part becomes the *KE* of the newly created molecule (ω') while the remaining energy will be transferred to the central energy *buffer* (following the law of conservation energy). The energy in the *buffer* is used to increase the chance of having a decomposition completed, in other words when a molecule has not enough energy to endow the *PE* of the resultant two molecules, it draws a portion of *KE* from the *buffer*. Hence, the value of *KELossRate* affects the Kinetic Energy a molecule possesses after the on-wall collision which also impacts the performance and the convergence of the algorithm. Consequently, we can say that *InitialKE* and *KELossRate* are two important parameters. They cooperate with each other to boost the optimization process. A large value of *KE* may lead to situation in which molecule keeps accepting solutions with higher function value over better ones. Whereas, small *KE* may potentially makes the molecule unable to transform to a new one with higher function value and gets stuck in local minimum. Similar conclusion can be drawn for the 'synthesis threshold' β which ranks fourth. It can be interpreted as the minimum kinetic energy a molecule should have.

According to the above analysis, a good choice of parameter combination is suggested as: *InitialKE* $= 8$, *KELossRate* $= 0.2$, $\alpha = 200$ and $\beta = 4$. Hence, we use this values in the following comparisons.

Table 5: Response value and significance rank

Level	*InitialKE*	*KELossRate*	α	β
1	0.850	0.957	1.075	0.700
2	0.835	0.600	0.475	0.497
3	0.705	0.632	0.572	0.892
4	0.407	0.607	0.675	0.707
Delta	0.442	0.350	0.502	0.185
Rank	2	3	1	4

6.2. Comparison with other powerful algorithms

In our experimental evaluation, we first compare the CRO with the pioneering variable neighborhood descent VND(a) and the NEH2 algorithms of Naderi & Ruiz (2010). Then, we compare the CRO with the powerful BSIG algorithm of Fernandez-Viagas & Framinan (2015) since it has been shown to outperform other state-of-the-art approaches Komaki et al. (2015).

In order to propose fair comparisons, several requirements should be considered:

- Using the same programming language.
- Running all the algorithms under the same development environment.
- Using the same computer.
- Using the same stopping criterion.

Thus, we perform all experiments on a PC with an Intel Core 2 Duo running at 2.20 GHz with 3.0 GB of RAM memory and we use C# language in Visual Studio 2015. The stopping criterion is a function of the number of jobs n, number of machines m and number of factories F: $n \times m \times F \times 3$ms (as in Fernandez-Viagas & Framinan (2015)). For each algorithm, we perform 5 independent runs and we calculate the average makespan. The only exception is the NEH2, for which only one single run was done since it is a deterministic algorithm.

Table 6: Average relative percentage deviation (RPD) of the 4 algorithms grouped by n and m

n×m	**CRO**	**VND(a)**	**NEH2**	**BSIG**
20×5	4.69	10.39	11.09	4.86
20×10	4.05	8.27	8.83	4.20
20×20	3.46	6.39	7.11	3.66
50×5	0.22	4.12	5.05	0.32
50×10	0.45	4.75	5.73	0.78
50×20	0.44	4.15	5.04	0.76
100×5	-0.02	2.01	2.51	0.07
100×10	0.09	3.00	4.18	0.31
100×20	0.13	3.01	4.08	0.40
200×10	-0.14	1.90	2.59	-0.01
200×20	-0.14	2.11	3.23	0.23
500×20	-0.27	1.35	2.29	0.03
Average	1.08	4.29	5.15	1.30

Average RPD values of the 4 algorithms are shown in Table 6. Results are grouped by the combination of the number of jobs (n) and the number of machines (m). First of all, we confirm that VND(a) obtains better results than NEH2 as was stated in the original paper Naderi & Ruiz (2010). We can also comment on the recent BSIG algorithm of Fernandez-Viagas & Framinan (2015). We can see that BSIG plainly beats the NEH2 and the VND(a), bringing down the RPD to 1.30%.

Comparing the results of CRO with those obtained by BSIG, we can see that they both show a very good performance. However, the proposed algorithm yields better results in 57% of instances. Both algorithms obtain the same performance for 10% of instances and for 33% of instances BSIG gives slightly better results than CRO.

In Table 6 we can also see that, for the groups (100×5), (200×10), (200×20) and (500×20) are minus which means that CRO updates the best-known solutions. More specifically, CRO improve the best-known solutions in 38.47% of instances.

Table 7 shows the average RPD of the four considered algorithms. Results are grouped by the number of factories (F), i.e., each cell of the table contains the average RPD of 120 instances. We can see that the average RPD values are very large for NEH2 and VND(a) algorithms for all instance sizes. It can also be seen that CRO shows the best performance bringing down the average RPD to just 1.08% which is almost 20% better than its challenger BSIG. It was also shown that the proposed algorithm performs the best for every combination. Except in one case with a number of factories = 7 where BSIG provides slightly better solutions. These results support our initial hypothesis that implementing and testing new advanced techniques is a worth while effort.

Table 7: Average relative percentage deviation (RPD) of the four algorithms grouped by F

F	**CRO**	**VND(a)**	**NEH2**	**BSIG**
2	0.63	3.55	4.58	1.27
3	0.59	3.84	4.82	1.08
4	0.74	4.00	5.00	0.97
5	0.94	4.26	5.03	1.00
6	1.33	4.70	5.40	1.37
7	2.25	5.38	6.04	2.12
Average	1.08	4.29	5.15	1.30

To show that the differences in Table 6 and Table 7 accomplished by the different algorithms are statistically significant, the paired samples t-test is carried out with all the 720 instances. Note that for this test, we consider the average makespan of the five runs. The statistical results of this test are listed in Table 8; the test yields $Sig = 0.000 < 0.05$, which implies that the results of the CRO statistically improves each other algorithm.

Table 8: Paired samples t-test

Algorithm	Mean	Std. Error Mean	Lower	Upper	t	Sig.
VND(a)-CRO	3.205	0.071	3.066	3.344	45.296	0.000
NEH2-CRO	4.069	0.075	3.916	4.209	54.413	0.000
BSIG-CRO	0.218	0.036	0.147	0.290	5.997	0.000

To sum up, we can confirm that the chemical reaction optimization algorithm outperforms other popular algorithms in solving the DPFSP. In fact, there are two important reasons for the rapid convergence and the good scheduling solutions of the proposed algorithm. The first is due the variety of change operators, which offers a good trade-off between intensification and diversification. The second is the capability to escape from local minimum and quickly guide the search to other better solution regions thanks to the conservation of energy law.

7. Conclusions and outlook

This study proposed a chemical reaction optimization algorithm to solve the distributed permutation flowshop scheduling problem with makespan criterion. Firstly, the representation of the molecule structure and its different attributes were defined. Secondly, the effective NEH2 algorithm was combined with random initialization to create the initial population of molecules with certain quality and diversity. Moreover, four operators were carefully designed to implement the local and global searches. Furthermore, the One-Point (OP) crossover and greedy strategy were integrated into synthesis and intermolecular ineffective collisions respectively in order to enhance the convergence of the algorithm and ameliorate the solution quality. The intensive experiment has been performed to evaluate our novel algorithm. Results proved that the effectiveness of the proposed CRO, is able to find new best-known solutions. Besides, in order to evaluate the performance of CRO, we compare it with several states of art algorithms with the use of the instances presented by Naderi & Ruiz (2010). Comparisons have proved that CRO can usually produce better solutions than other algorithms.

Even though this study proposed an effective CRO algorithm for the DPFSP, there are some impediments that should be improved in future research work. In fact, this study have just considered the basic framework of the CRO metaheuristic. However, there is a new adaptive version of CRO (ACRO) James et al. (2015) that relieve the effort in tuning parameters. The new adaptive CRO reduces the number of parameters to just three which allows us to focus our effort on the conception of the algorithm operators. Moreover, this study considers only the maximum completion time objective which contrasts with the real-life production since scheduling situations multi-criteria should be considered simultaneously to satisfy the multiple scheduling requirements.

On the basis of the satisfactory results obtained by this work and the limitations presented above, we plan to explore the following issues:

- Refining the settings for the proposed algorithm by using the new adaptive scheme ACRO and comparing the two variants of the CRO metaheuristic.
- Extending the idea of combining the proposed CRO algorithm with other swarm intelligence algorithms, such as Particle Swarm Optimization to explore the search space and improve the solution.
- Considering important real-world constraints such as setup times and assembly operations.
- Investigate in the applicability of the proposed approach to the multi-objective DPFSP.

References

Ahmadizar, F., & Barzinpour, F. (2010). A hybrid algorithm to minimize makespan for the permutation flow shop scheduling problem. *International Journal of Computational Intelligence Systems*, *3*, 853–861.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. John Wiley & Sons.

Bargaoui, H., & Driss, O. B. (2014). Multi-agent model based on tabu search for the permutation flow shop scheduling problem. In *Distributed Computing and Artificial Intelligence, 11th International Conference* (pp. 519–527). Springer.

Campbell, H. G., Dudek, R. A., & Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. *Management science*, *16*, B–630.

Črepinšek, M., Liu, S.-H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, *45*, 35.

Fernandez-Viagas, V., & Framinan, J. M. (2015). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, *53*, 1111–1123.

Fernandez-Viagas, V., Ruiz, R., & Framinan, J. M. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, *257*, 707–721.

Gao, J., & Chen, R. (2011). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems*, *4*, 497–508.

Gao, J., Chen, R., & Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, *51*, 641–651.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, *5*, 287–326.

Gupta, J. N., & Stafford, E. F. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research*, *169*, 699–711.

James, J., Lam, A. Y., & Li, V. O. (2014). Chemical reaction optimization for the set covering problem. In *Evolutionary Computation (CEC), 2014 IEEE Congress on* (pp. 512–519). IEEE.

James, J., Lam, A. Y., & Li, V. O. (2015). Adaptive chemical reaction optimization for global numerical optimization. In *Evolutionary Computation (CEC), 2015 IEEE Congress on* (pp. 3192–3199). IEEE.

Jemni, M., Ladhari, T. et al. (2011). Solving the permutation flow shop problem with makespan criterion using grids. *International Journal of Grid and Distributed Computing*, *4*, 53–64.

Johnson, S. M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics (NRL)*, *1*, 61–68.

Kahn, K. B. (2012). *The PDMA handbook of new product development*. John Wiley & Sons.

Komaki, G., Mobin, S., Teymourian, E., & Sheikh, S. (2015). A general variable neighborhood search algorithm to minimize makespan of the distributed permutation flowshop scheduling problem. *World Academy of Science, Engineering and Technology, International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering*, *9*, 2701–2708.

Lam, A. Y., & Li, V. O. (2010). Chemical-reaction-inspired metaheuristic for optimization. *IEEE Transactions on Evolutionary Computation*, *14*, 381–399.

Lam, A. Y., & Li, V. O. (2012). Chemical reaction optimization: A tutorial. *Memetic Computing*, *4*, 3–17.

Lam, A. Y., Li, V. O., & Xu, J. (2013). On the convergence of chemical reaction optimization for combinatorial optimization. *IEEE Transactions on Evolutionary Computation*, *17*, 605–620.

Li, J.-q., & Pan, Q.-k. (2012). Chemical-reaction optimization for flexible job-shop scheduling problems with maintenance activity. *Applied soft computing*, *12*, 2896–2912.

Lin, S.-W., Ying, K.-C., & Huang, C.-Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research*, *51*, 5029–5038.

Liu, H., & Gao, L. (2010). A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem. In *Manufacturing Automation (ICMA), 2010 International Conference on* (pp. 156–163). IEEE.

Liu, Y.-F., & Liu, S.-Y. (2013). A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. *Applied Soft Computing*, *13*, 1459–1463.

Marzouki, B., & Driss, O. B. (2015). Multi agent model based on chemical reaction optimization for flexible job shop problem. In *Computational Collective Intelligence* (pp. 29–38). Springer.

Montgomery, D. C. (2008). *Design and analysis of experiments*. John Wiley & Sons.

Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, *30*, 1061–1071.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, *37*, 754–768.

Naderi, B., & Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research*, *239*, 323–334.

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*, 91–95.

Ruiz, R., & Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, *165*, 479–494.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*, 2033–2049.

Shao, W., & Pi, D. (2016). A self-guided differential evolution with neighborhood search for permutation flow shop scheduling. *Expert Systems with Applications*, *51*, 161–176.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European journal of Operational research*, *47*, 65–74.

Truong, T. K., Li, K., & Xu, Y. (2013). Chemical reaction optimization with greedy strategy for the 0–1 knapsack problem. *Applied Soft Computing*, *13*, 1774–1780.

Wang, S.-y., Wang, L., Liu, M., & Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, *145*, 387–396.

Highlights

- This is the first time that CRO is applied to solve the distributed permutation flow shop scheduling problem.
- The proposed CRO combines iterated greedy strategy and crossover mechanism to improve the optimization process.
- The proposed algorithm improves more than 200 best-known solutions.
- The effectiveness of the proposed CRO has been statistically proved.