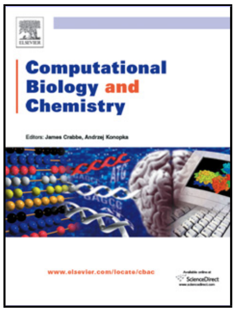

An Improved Chemical Reaction Optimization Algorithm for Solving the Shortest Common Supersequence Problem

Fei Luo, Cheng Chen, Joel Fuentes

PII: S1476-9271(19)31146-6

DOI: https://doi.org/10.1016/j.compbiolchem.2020.107327

Reference: CBAC 107327

To appear in: *Computational Biology and Chemistry*

Received Date: 22 December 2019

Revised Date: 14 June 2020

Accepted Date: 28 June 2020

Please cite this article as: { doi: https://doi.org/

An Improved Chemical Reaction Optimization Algorithm for Solving the Shortest Common Supersequence Problem

Fei Luo*

School of Information and Engineering, East China University of Science and Technology, Shanghai, China

Cheng Chen

School of Information and Engineering, East China University of Science and Technology, Shanghai, China

Joel Fuentes

Department of Computer Science and Information Technologies, Universidad del Bío-Bío. Chillán, Chile

Abstract

The shortest common supersequence (SCS) problem is a classical NP-hard problem, which is normally solved by heuristic algorithms. One important heuristic that is inspired by the process of chemical reactions in nature is the chemical reaction optimization (CRO) and its algorithm known as CRO_SCS. In this paper we propose a novel CRO algorithm, dubbed IMCRO, to solve the SCS problem efficiently. Two new operators are introduced in two of the four reactions of the CRO: a new circular shift operator is added to the decomposition reaction, and a new two-step crossover operator is included in the inter-molecular ineffective collision reaction. Experimental results show that IMCRO achieves better performance on random and real sequences than well-known heuristic algorithms such as the ant colony optimization, deposition and reduction, enhanced beam search, and CRO_SCS. Additionally, it outperforms its baseline CRO_SCS for DNA instances, averaging a SCS length reduction of 1.02, with a maximum

*Corresponding author
Email address: luof@ecust.edu.cn (Fei Luo)

length reduction of up to 2.1.

Keywords: chemical reaction optimization; shortest common supersequence; heuristic algorithm; NP-hard

1. Introduction

The shortest common supersequence (SCS) is a well-known NP-hard problem [1] which has been widely studied for several years. It can be formalized as follows: Let Σ be an alphabet set and a string S_i be a set of zero or more characters from Σ. S=$\{S_1$,S_2,...,$S_n\}$ is a set with n strings and $1 \leq i \leq n$. The length of S_i with q characters from Σ is denoted as $\left|S_i\right|$. The string C is called as a supersequence of the string S_i, if $\left|S_i\right| < \left|C\right|$ and S_i can be embedded in C. If the string CS is the supersequence of all strings in the set S, CS is defined as the common supersequence of the set S. Consider a set $\{CS_1$,CS_2,...,$CS_m\}$ with m strings, and CS_j is a common supersequence of S, where $1 \leq j \leq m$. The SCS of S, shorten as SCS(S), can be defined as in Formula 1, subjected to $1 \leq j \leq m$ and $l_j = \left|CS_j\right|$, which is the common supersequence with the minimum length .

$$SCS(S) = min(l_j) \tag{1}$$

The SCS problem occurs often in real life and has been subject of study in the last decades due to its various applications in many fields. Deoxyribonucleic acid (DNA) sequencing [2], data compression [3], artificial intelligence (AI) planning [4], query optimization in databases[5], and multiple sequence alignment problems [6] are some examples where the SCS problem is applied.

In order to solve the SCS problem and find the optimal solution, different approaches have been proposed. Important proposals found in the literature are greedy methods [7], ant colony optimization (ACO) [8], artificial bee colony (ABC) [9], enhanced beam search (IBS) [10], deposition and reduction (DR) [11], and the chemical reaction optimization (CRO) algorithm known as CRO_SCS [12]. Previous studies have proven that CRO_SCS achieves in average better performance than the other heuristic algorithms [12].

To boost the performance of CRO-based algorithms, one approach is changing one or more operators to improve the capability of the local or global search. One example is RMCRO [13], which merges the idea of the repellent-attractant rule and convergence acceleration to create a fusion chemical reaction optimization based on random molecules. Other methods combine CRO with other heuristics which results in hybrids algorithms. Some examples are the hybrid algorithm based on particle swarm and CRO (HP-CRO) [14], the hybrid chemical reaction with employed bee operator EBCRO [15] and the bat-mutation CRO algorithm BMCRO [16]. Problem-specific heuristics can easily be incorporated into elementary reactions. One can design a molecule for different attributes that suit the problem to be solved as well as give the flexibility managing different operators. Thus, the present proposal is based on the design of novel operators aimed in order to improve the performance of CRO, which is then employed to solve the SCS problem.

This paper presents IMCRO, a novel CRO-based algorithm to solve the SCS problem. The main contribution of the paper is the extension and enhancement of the CRO_SCS framework with the introduction of two new operators for decomposition and inter-molecular ineffective collisions in two of the four reactions of CRO. Our findings demonstrate that these new operators boost the performance and efficiency of CRO when solving the SCS problem. Experimental results on random and real datasets show that IMCRO outperforms previous CRO-based proposals, such as CRO_SCS, as well as related state-of-the-art heuristic algorithms.

The rest of the paper is organized as follows. The related work is summarized in Section II. A detailed description of the IMCRO design, its framework and new operators for solving the SCS problem are presented in Section III. The performance evaluation is presented in Section IV, where experimental results and a detailed analysis are described. Finally, conclusions and future work are drawn in Section V.

2. Related Work

The SCS problem was first defined by David Maier in 1976 [17], where it was proven to be NP-complete for sequences with alphabet size over 5. Since then, the SCS problem has been widely used in different fields, such as data compression[18], scheduling [19], and bioinformatics [20]. Particularly in bioinformatics, the SCS problem is utilized effectively to generate the guide tree in multiple sequence alignment [21].

The SCS problem has been approached from different angles. Main proposals are based on greedy algorithms [22], field programmable gate arrays [23], and some heuristic algorithms such as memetic algorithms [24], ACO [8], ABC [9], IBS [10], DR [11], and CRO [12].

The CRO algorithm is inspired from the process of chemical reactions, and it was first proposed by Lam and Li in 2010 [25]. A chemical reaction undergoes with some sub-reactions, which means a reaction goes through some intermediate states. In every state, the energy of the molecule is lower than the previous state and then the molecule becomes more stable. This phenomenon can be correlated with the step-wise searching of optimization problems.

One important characteristics of the CRO algorithm is that it exploits both the local and the global searches through the reaction operators. High flexibility when designing reaction operators and variable population sizes helps CRO to adapt to different kinds of NP-hard problems [26]. Some examples of these problems are transportation scheduling optimization [27], economic dispatching [28], flow shop scheduling [29], generalized vertex cover problem [30], optimization of protein folding [31], virtual machine placement [32], and next release problem [33]. Moreover, the CRO algorithm is also effective on data mining [34] for word detection [35], and DNA structure prediction [36]. Apart from those applications, CRO can also be applied to solve the SCS problem [12] and the longest common supersequence problem [37]. Overall, it has been reported that the CRO algorithm and its variants achieve good performance when solving the problems mentioned above.

Table 1: Parameters

Parameter	Description
Popsize	Set of all feasible solutions
PE (potential energy)	The objective function value related to a corresponding molecule
KE (kinetic energy)	Numerical value of the amount of tolerance to accept a worse solution
NumHit	Number of collisions by a molecule
KELossRate	Percentage of the upper limit of KE reduction
MoleColl	Threshold to determine the type of chemical reaction: uni-molecule or inter-molecule
Initial KE	Initial value of the kinetic energy assigned to each molecule in the initialization stage
α, β	Threshold values for the intensification and diversification
MinStruct	The molecule structure that has minimum potential
MinHit	The number of hits when a molecule has MinStruct

In general, most of the CRO-based algorithms have a similar framework while their operators for their reactions are distinct. These operators are often adapted to a specific problem, which helps achieving important performance gains. Another possibility to solve specific problems is to extend the traditional CRO algorithm by designing brand-new operators.

Authors in [12] introduced a novel CRO algorithm, named as CRO_SCS, for solving the SCS problem. In particular, they added a new repair function to check and repair the molecule from different iteration stages. When reaction operators jump outside the solution space while searching locally or globally, the repair function takes them back to the solution space. Thus it ensures diversification and intensification properties.

3. IMCRO

3.1. Framework

The general framework used in our proposal corresponds to an improved extension of the CRO-based algorithm CRO_SCS, introduced in [12]. It consists of three stages and they are described in Algorithm 1: initialization, iteration, and the finalization. All the parameters used in the algorithm are presented in Table 1.

The first stage of IMCRO is initialization. In this stage the elements and molecules, such as PopSize, KELossRate, MoleColl, buffer, Initial KE, α, and β (defined in Table 1) are initialized. The molecule energy includes potential energy (PE) and kinetic energy (KE). The potential energy refers to the objective function, as shown in Formula 2, which is the function of the corresponding solution ω. The kinetic energy refers to the amount of tolerance to accept a worse value, and the energy of the surroundings is considered as buffers. It is always important to mention that these chemical reactions follow the energy conservation rule. Energy cannot be created or destroyed rather than it is be transformed from one state to another.

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

Population generation and supersequence representation are also included in initialization stage. The population is generated using random insertion operations. At the beginning, the supersequence C is empty, and then we take each string from a set of strings S. Let us assume that the string taken from S is S_i where $1 \leq i \leq popsize$. Then we take the supersequence from an array, where each character is an element of the array. Now for each symbol of S_i, a particular position is randomly selected from the elements of supersequence. If the similar character is not found, the supersequence C will be appended by inserting that symbol. Otherwise, the process is iterated for the next symbol. Figure 1 displays the process of population generation.

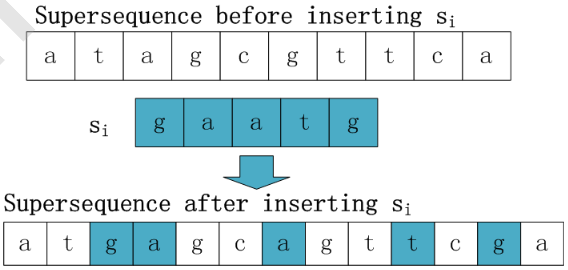
Supersequence before inserting s_i
a t a g c g t t c a
s_i
g a a t g
Supersequence after inserting s_i
a t g a g c a g t t c g a

Figure 1: population generation

Integer numbers are used to represent symbols from the alphabet and the encoding sequence. After generating the population, each supersequence can be encoded by a set of integer values. Then for each symbol in the supersequence, the corresponding integer value represents a solution of that supersequence. For example, $\Sigma = \{a, c, g, t\}$ has an integer encoding as $\{0, 1, 2, 3\}$, then the supersequence $\Sigma = \{a, c, t, g, t, c, g, a\}$ can be represented as $\{0, 1, 2, 3, 2, 1, 3, 0\}$, as shown in Figure 2.

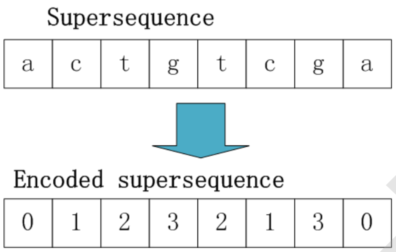

Figure 2: solution representation

The second stage, iteration, is divided into two subtasks: *reaction* and *repair*, which correspond to lines 3-21 in Algorithm 1. In the reaction step there are four main operators: on-wall ineffective collision, decomposition, inter-molecular ineffective collision and synthesis. They fall into two categories: uni-molecule reaction and inter-molecule reaction. On-wall ineffective collision and decomposition are uni-molecule reactions, while inter-molecular ineffective collision and synthesis are inter-molecule reactions.

At the beginning of the main iteration in Algorithm 1, a parameter t is randomly generated. It determines which type of reaction, e.g. uni-molecule or inter-molecule reaction, will be triggered. If $t > MoleColl$, uni-molecule reactions are triggered; otherwise, inter-molecule reactions are trigged. In the uni-molecule reaction, the parameter α determines which type of reaction occurs. If $(NumHit - MinHit) > \alpha$, decomposition occurs; otherwise, on-wall ineffective collision occurs. In the same way, in the inter-molecule reaction, the parameter β determines the type of reaction. If $KE \leq \beta$, synthesis occurs; otherwise, inter-molecular ineffective collision occurs.

Algorithm 1 IMCRO algorithm

Input: population and parameter values.

```
Initialization: PopSize, KELossRate, MoleColl, buffer, Initial KE, $\alpha$ ,and $\beta$.
Create PopSize number of molecules
while the stopping criteria is not met do
    Generate $t \in [0, 1]$
    if $(NumHit - MinHit) > \alpha$ then
        Randomly select one molecule m
        if $(NumHit - MinHit) > \alpha$ then
            Trigger Decomposition
        else
            Trigger On-wall Ineffective Collision
        end if
    else
        Randomly select two molecules $m_1$ and $m_2$
        if $KE \leq \beta$ then
            Trigger Synthesis
        else
            Trigger Inter-molecular ineffective collision
        end if
    end if
    Check for any new better solution
end while
```

Output: the best solution from the population.

When the algorithm obtains a solution it is validated: if it cannot satisfy the requirement of the problem, a *repair* algorithm mends the obtained solution until it fits the termination condition. Afterwards, the algorithm enters the final stage. If the obtained solution matches the stopping criteria, it is reported as the final solution; otherwise, the algorithm continues the iteration again and repeats the reactions.

Typical stopping criteria include the maximum amount of the CPU time, the maximum number of function evaluations performed, and obtaining an objective function value less than a predefined threshold, among others.

In the final stage, IMCRO simply outputs the best solution found with its objective value and terminates the procedure.

3.2. Operators

The operators used by IMCRO in the reaction stage, i.e. on-wall ineffective collision, inter-molecular ineffective collision, decomposition, and synthesis, are described in the following subsections.

3.2.1. On-wall ineffective collision

In a chemical reaction, when a molecule collides with the wall of the container the structure of this molecule changes. We exploit the one-difference operator [12] to change the structure of molecule. The process begins by selecting one element $m[i]$ from molecule m randomly, and subsequently changing its value. The expression $rand(V_{low}, V_{upper})$ defines the candidate range for the random function of the alphabet in the SCS. If $(m[i] + j) \leq V_{upper}$, $m[i]$ is replaced by $(m[i] + j)$. Otherwise, $(m[i] - j)$ replaces $m[i]$. After this collision, a new molecule m' is obtained.

One-difference operator changes a character in the supersequence and helps to find a neighbor solution in the space. It does not shorten the length of the supersequence, but it helps to find a solution near to the initial population and expand its possibilities to become the SCS. Figure 3 illustrates the process of on-wall ineffective collision and its pseudocode procedure is shown in Algorithm 2.

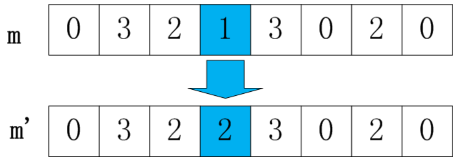

Figure 3: On-wall ineffective collision (one-change operator)

Algorithm 2 On-wall ineffective collision

```
Input: solution $m[0, 1, 2, ..., n-1]$, $V_{low}$,$V_{upper}$.
$m'$ is duplicated from m.
var i = rand(0, n-1)
var j=rand($V_{low}$,$V_{upper}$)
if $m[i] + j \le V_{upper}$ then
    $m'[i] \leftarrow m[i] + j$
else
    $m'[i] \leftarrow m[i] - j$
end if
Output: solution $m'[0, 1, 2, ..., n-1]$.
```

3.2.2. Inter-molecular ineffective collision

This operator takes two molecules m_1 and m_2 randomly from the population, and uses the crossover operator to produce two new solutions m'_1 and m'_2.

In order to improve the capability of local search and avoid falling into local optimization, a great change of the structure of molecules should be considered. Therefore, we design a new operator named "two-step crossover". It is a two-step process: the first step is to crossover between two molecules, and the second step is to crossover inside the molecule itself.

The first step is similar to the two-point crossover operator [12]. It selects two molecules m_1 and m_2, and then two random points n_1 and n_2 are selected, where $n_2 > n_1$. The odd parts in m_1 and the even part of m_2 are merged to form m_1'. The even parts in m_1 and the odd part in m_2 are merged to form m_2'. After producing the molecules m_1' and m_2', crossover occurs between these two different molecules.

In the second step, crossover is applied inside the molecule m_1' and m_2' to produce molecule m_1'' and m_2''. Different from the first step, the crossover in this step occurs inside the molecule m_1' and m_2' themselves. It exchanges two odd parts of m_1' to get molecule m_1'', and it exchanges two odd parts of m_2' to get molecule m_2''. Algorithm 3 illustrates the process of this operator and an example is shown in Figure 4. The original molecules are $m_1 = \{0, 1, 2, 3, 1, 3, 1, 0\}$ and $m_2 = \{2, 0, 0, 1, 2, 2, 3, 2\}$. m_1' and m_2' are obtained through the first inter-crossover between m_1 and m_2, and m_1'' and m_2'' are further attained through the second crossover inside m_1' and m_2', respectively.

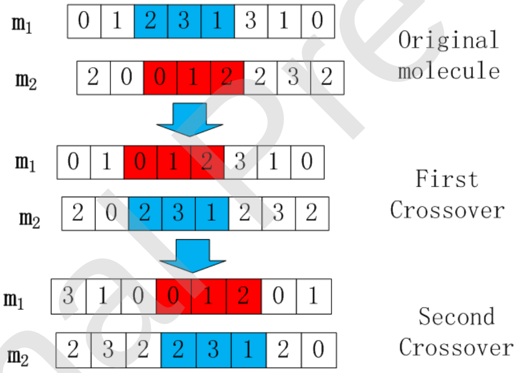

Figure 4: Two-step crossover operator

3.2.3. Decomposition

The decomposition reaction is utilized to allow the system to explore another region of the search space. This is useful because massive changes occur in the molecule structures of the newly produced molecules.

We introduce a circular shift operator [38] for this decomposition reaction. First, a new solution is obtained by generating an integer within the range $[-n, n]$. Then, two integers are randomly selected. The first one is a negative

Algorithm 3 Inter-molecular

Input: m_1 and m_2.

```
take two points $x_1$, $x_2$ randomly, where $x_2 > x_1$.
for $i \leftarrow 1$ to length of $m_1$ do
    if $i < x_1$ or $i > x_2$ then
        $m'1[i] \leftarrow m1[i]$
        $m'2[i] \leftarrow m2[i]$
    else
        $m'1[i] \leftarrow m2[i]$
        $m'2[i] \leftarrow m1[i]$
    end if
end for
for $j \leftarrow 1$ to length of $m_1$ do
    if $j < x_1$ then
        $m''_1[length - x_1 + 2 + j] \leftarrow m'_1[j]$
        $m''_2[length - x_1 + 2 + j] \leftarrow m'_2[j]$
    else if $x_1 \leq j \leq x_2$ then
        $m''_1[length + 1 - x_1 - x_2 + j] \leftarrow m'_1[j]$
        $m''_2[length + 1 - x_1 - x_2 + j] \leftarrow m'_2[j]$
    else
        $m''_1[j - x_2] \leftarrow m'_1[j]$
        $m''_2[j - x_2] \leftarrow m'_2[j]$
    end if
end for
```

Output: m''_1 and m''_2.

number $-i$, while the other is a positive number j. The negative number $-i$ is used for shifting to the left i steps. The positive number j is used for shifting to the right j steps. Figure 5 shows an example of the circular shift operator. In this example, the two numbers chosen in the range $[-n, n]$ are -2 and 2, respectively. The left molecule m_1 is obtained by shifting to the left two steps, and the right molecule m_2 is obtained by shifting to the right two steps.

This reaction creates two molecules which have different sequence patterns from the initial molecule. It explores another region of the solution space where the global minimum solution might be found. Algorithm 4 shows the pseudocode of the decomposition reaction.

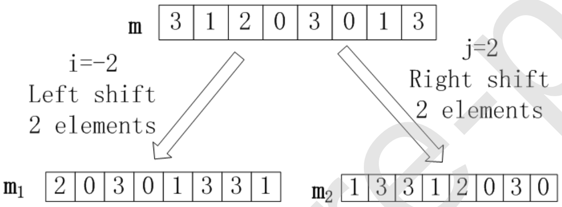

Figure 5: Circular shift operator

3.2.4. *Synthesis*

In this reaction, two molecules (m_1 and m_2) from the population are combined to form a new molecule m', working as the opposite of the decomposition reaction. In this reaction we use a variant of the probabilistic select operator [12].

The operator accelerates the convergence by generating different structures of molecules and allows the system to explore throughout the solution space. This operation maximizes the probability of seeking optimal result and avoids the trap of local optimal solutions. The exploration of the solution spaces increases the chance of finding a better solution than the undergoing reactants.

Figure 6 depicts an example of the synthesis reaction, and its pseudocode procedure is shown in Algorithm 5. Symbols and their frequencies are calculated and kept in *array*1 and *array*2 for m_1 and m_2 respectively. In each iteration,

Algorithm 4 Decomposition

```
Input: $m$
select two numbers $a$ and $b$ randomly.
for $i \leftarrow 1$ to length of $a$ do
    if $i \leq a$ then
        $m_1[length - a + i] \leftarrow m[i]$
    else
        $m_1[i - a] \leftarrow m[i]$
    end if
end for
for $j \leftarrow 1$ to length of $b$ do
    if $j \leq length - b$ then
        $m_2[j + b] \leftarrow m[j]$
    else
        $m_2[j - length + b] \leftarrow m[j]$
    end if
end for
Output: $m_1$ and $m_2$.
```

the symbol with a higher frequency from m_1 or m_2 is appended to m'. Then the frequency of the selected symbol is reduced by one from the array. Here, the frequency is considered to ensure that symbols with more occurrence are selected for the new supersequence. The action may cause deterministic selection, but it gives somehow better result than non-deterministic selection.

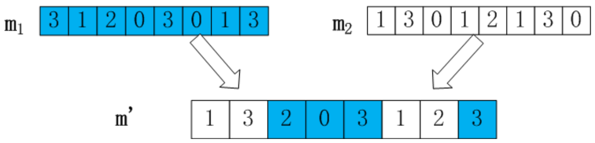

Figure 6: Probabilistic select operator

```
Algorithm 5 Synthesis
Input: $m_1$ and $m_2$.
    Generate $array1$ for the frequencies of the symbols used in $m_1$.
    Generate $array2$ for the frequencies of the symbols used in $m_2$
    for $i \leftarrow 1$ to n do
        if array1$[m_1[i]] \geq$ array2$[m_2[i]]$ then
            $m'[i] \leftarrow m_1[i]$
        else
            $m'[i] \leftarrow m_2[i]$
        end if
    end for
Output: $m'$.
```

3.2.5. Repair function

When IMCRO obtains a solution by one of four reactions, a repair function checks this solution and repairs it if necessary. This repair function contains two phases. The first phase performs a validation of the new supersequence by checking it against the corresponding reaction and every string in S. If no violation is found –violation means the sequence of a string and the sequence of supersequence are not the same–, the new molecule is inserted into the population and the function skips the repair phase. If any violation is found it enters the repair phase, where the number of violations in the supersequence is calculated. A threshold value named *violation threshold* (VT) is defined for this purpose. If the number of violations is more than the threshold, it discards the changes occurred during the chemical reactions. If the number of violations is less than the threshold, the function goes through those strings with mismatches of sequences with the supersequence. Afterwards, every mismatched string is fixed. We exploit the same definition of VT introduced in [12], as shown in Formula 3, where the best result can be achieved if the number of strings n is 200 times of the VT.

$$VT = \begin{cases} \frac{n}{200}, & n \geq 200, \\ \frac{n}{100}, & otherwise \end{cases} \tag{3}$$

4. Experiments and Evaluation

A set of experiments were carried out in order to evaluate the performance and efficiency of IMCRO. In this performance evaluation we compare IMCRO to CRO_SCS and some state-of-the-art heuristic algorithms, such as ACO, IBS and DR.

4.1. Configuration of Experiments

All algorithms were implemented in Java and executed in a computer machine with Intel Core i5-4210U CPU at 2.40GHz, 4.00GB RAM and Windows 7 (64 bits). Important parameters used in this performance evaluation are described in Table 2.

Two types of datasets were used in the experiments, they were taken from the online repository BioMedCentral[1,2]. The first dataset corresponds to a random

[1] `http://www.biomedcentral.com/content/supplementary/1471-2105-7-S4-S12-S1.zip`

[2] `http://www.biomedcentral.com/content/supplementary/1471-2105-7-S4-S12-S2.zip`

Table 2: Parameters used by IMCRO

Parameter	Value
PE	Length(m)
KE	-
PopSize	20
KELossRate	0.6
MoleColl	0.2
α	Rand[10,100]
β	Rand[10,100]
NumHit	0
MinHit	0

DRM for DNA sequences with 15 instances ($\Sigma = 4$), while the second one is a real dataset DRL with 11 instances ($\Sigma = 20$). Strings in each instance have equal length, and in DRL six instances are DNA sequences while five instances are protein sequences. Some parameters used for this performance evaluation are described below, and the details of the datasets are shown in Table 3 and 4.

- n: number of strings in each instance
- k: length of each string.
- L: average SCS length of the algorithm. For the specialized algorithm alg and the instance ins, L is specialized as $L_{alg}(ins)$. Therein, $alg \in \{ACO, IBS, DR, CRO\_SCS, IMCRO\}$, and $ins \in DRM \cup DRL$.
- T: average execution time of the algorithm. In the same way, T is specialized as $T_{alg}(ins)$ for a concrete alg and ins.
- SD: standard deviation. It indicates the stability of the algorithm, and it is specilized as $SD_{alg}(ins)$ for a concrete alg and ins.

Table 3: Random dataset ✓

No.	n	k
1	5	10
2	10	10
3	50	10
4	100	10
5	5	100
6	10	100
7	50	100
8	100	100
9	500	100
10	1000	100
11	5000	100
12	100	1000
13	500	1000
14	1000	1000
15	5000	1000

Table 4: Real dataset

No.	n	k
DNA-1	100	500
DNA-2	500	500
DNA-3	100	1000
DNA-4	500	1000
DNA-5	100	100
DNA-6	500	100
PROT-1	100	500
PROT-2	500	500
PROT-3	1000	500
PROT-4	100	100
PROT-5	500	100

In the experiments, each set was executed with the mentioned algorithms 20 times. We obtained 200 different results after testing one instance (including the length of SCS and the execution time). Then the average SCS length and average execution time were calculated for these 200 results. Afterwards, we calculated the average SCS length and the average execution time for each instance. After repeating the process described above, the average SCS length and average execution time for all instances were obtained. In the experiments there are two stopping criteria defined in the CRO framework: the maximum number of iterations for the CRO operations, and the potential energy (Formula 2) exceeding the threshold. If one of the two is satisfied, the final solution will be output, as depicted in Algorithm 1. In particular, the maximum number of iterations was set to 500, while the threshold is related to the structure of each instance.

4.2. Results and Analysis

The average SCS length L and average execution time T are used to evaluate the performance of every algorithm. The best algorithm should have the shortest average SCS length and shortest average execution time. The standard deviation SD is used to determine the stability of the algorithms. The lower the value of SD for the algorithm is, the more stable the algorithm is. Tables 5, 6, 7 and 8 show the base performance of the algorithms with the datasets. Standard deviations SD of the SCS length are displayed in Table 5 and 7. Best results are emboldened in each Table.

Table 5: Average SCS Length (Standard Deviation) in random datasets

n	k	L (SD)				
		ACO	DR	IBS	CRO_SCS	IMCRO
5	10	22.5 (1,50)	21.2 (1.30)	19.9 (1.27)	20.2 (0.69)	**19.7 (0.62)**
10	10	26.7 (2.06)	25.1 (1.45)	25.2 (0.89)	25.3 (0.70)	**24.9 (0.65)**
50	10	31.5 (0.58)	31.1 (0.71)	30.0 (0)	29.3 (0.62)	**28.6 (0.48)**
100	10	33.0 (0)	32.5 (0.54)	32.0 (0)	32.1 (0.90)	**31.5 (0.50)**
5	100	207.4 (11.35)	198.2 (1.95)	184.0 (0)	181.6 (0.98)	**180.5 (0.66)**
10	100	233.7 (1.49)	226.2 (2.25)	210.0 (0)	209.4 (0.88)	**208.3 (0.62)**
50	100	263.7 (0.88)	262.0 (1.99)	252.0 (0)	244.4 (1.23)	**243.8 (0.72)**
100	100	270.1 (1.73)	269.2 (1.60)	261.1 (1.08)	252.1 (1.50)	**251.0 (0.74)**
500	100	277.2 (0.88)	277.8 (1.66)	273.6 (1.61)	267.6 (1.22)	**266.7 (1.21)**
1000	100	281.7 (1.25)	278.5 (1.37)	276.8 (1.31)	270.1 (1.45)	**269.7 (1.21)**
5000	100	282.9 (0.67)	282.9 (0.94)	281.5 (1,50)	271.9 (1.34)	**270.6 (0.70)**
100	1000	2535.6 (8.30)	2531.6 (3.06)	2466.7 (2.58)	2443.2 (1.47)	**2442.1 (0.66)**
500	1000	2565.6 (2.74)	2578.8 (2.72)	2540.2 (2.69)	2532.1 (1.45)	**2530.5 (0.61)**
1000	1000	2570.8 (8.62)	2581.4 (1.32)	2555.5 (1.30)	2535.6 (1.23)	**2533.9 (1.05)**
5000	1000	2590.6 (3.65)	2586.9 (3.32)	2571.6 (3.25)	2562.9 (1.45)	**2561.7 (0.70)**

Table 5 shows that for each instance $ins = DRM$ and algorithm $alg \in \{ACO, IBS, DR, CRO\_SCS\}$, $L_{IMCRO}(ins) < L_{alg}(ins)$. Table 6 shows that

Table 6: Average execution time in random datasets

n	k	T/s				
		ACO	DR	IBS	CRO_SCS	IMCRO
5	10	0.8	0.018	0.03	**0.008**	**0.008**
10	10	1.00	0.033	**0.03**	**0.03**	**0.03**
50	10	2.3	0.1	0.07	0.08	**0.065**
100	10	3.5	0.15	0.12	0.08	**0.055**
5	100	5.9	0.6	0.14	0.02	**0.013**
10	100	8.6	1.18	0.22	0.14	**0.12**
50	100	16.3	4.07	0.46	0.37	**0.26**
100	100	23.5	7.28	0.91	0.65	**0.52**
500	100	65.5	27.3	3.06	1.69	**0.92**
1000	100	127.9	69.2	6.45	2.66	**1.95**
5000	100	706.6	339.4	41.65	**5.01**	**5.01**
100	1000	207.7	420.6	6.33	5.75	**5.5**
500	1000	651.1	1205.3	37.93	15.85	**14.12**
1000	1000	1296.5	2116.8	61.67	39.9	**22.0**
5000	1000	3101.6	3761.4	487.16	480.02	**480.01**

Table 7: Average SCS Length (Standard Deviation) in real datasets

name	n	k	L (SD)				
			ACO	DR	IBS	CRO_SCS	IMCRO
DNA-1	100	500	1346.9 (16.24)	1332.6 (5.02)	1280.7 (4.74)	1271.4 **(0.76)**	**1271.0** (0.82)
DNA-2	500	500	1520.0 (2.05)	1404.6 (2.87)	1352.7 (2.69)	1351.8 **(0.40)**	**1350.8** (0.87)
DNA-3	100	1000	2712.2 (18,76)	2670.1 (7.61)	2542.9 (8.52)	2442.5 (1.28)	**2440.9 (0.81)**
DNA-4	500	1000	3092.1 (8.31)	2782.7 (8.22)	2664.4 (23.16)	2532.4 (1.11)	**2530.3 (1.10)**
DNA-5	100	100	297.8 (10.42)	285.4 (1.67)	272.3 (2.0)	252.1 (1.45)	**251.5 (1.15)**
DNA-6	500	100	405.2 (27.70)	291.5 (1.30)	288.3 (2.16)	267.2 (1.39)	**266.3 (1.21)**
PROT-1	100	500	6908.2 (6.6)	4851.4 (9.3)	4349.7 (5.7)	4312.4 **(0.86)**	**4311.6** (1.04)
PROT-2	500	500	8910.4 (10.3)	5545.2 (15.2)	5229.3 (8.8)	5041.1 **(0.88)**	**5040.8** (1.05)
PROT-3	1000	500	11086 (6.7)	5748.7 (11.2)	5395.6 (7.9)	**5301.7 (1.31)**	5301.9 (1.37)
PROT-4	100	100	1303.5 (5.5)	1005.6 (6.2)	913.3 (1.3)	**920.9** (0.85)	921.2 **(0.75)**
PROT-5	500	100	1776.2 (4.6)	1205.1 (5.4)	1107.6 (2.1)	1126.0 **(0.71)**	**1125.9** (0.79)

Table 8: Average execution time in real datasets

name	n	k	T/s				
			ACO	DR	IBS	CRO_SCS	IMCRO
DNA-1	100	500	151.3	349.4	3.00	**1.91**	**1.91**
DNA-2	500	500	613.1	540.6	17.14	8.35	**7.83**
DNA-3	100	1000	334.4	483.6	10.31	5.85	**5.48**
DNA-4	500	1000	1514.1	1156.5	42.4	15.7	**15.46**
DNA-5	100	100	41.97	7.87	0.92	0.56	**0.49**
DNA-6	500	100	92.0	37.55	2.95	0.81	**0.73**
PROT-1	100	500	560.3	1125.3	92.4	31.98	**21.18**
PROT-2	500	500	1450.2	1905.4	307.8	52.29	**50.50**
PROT-3	1000	500	3205.4	4002.7	1905.6	116.12	**110.76**
PROT-4	100	100	16.4	31.2	13.5	1.66	**1.54**
PROT-5	500	100	123.5	95.7	65.3	4.63	**4.53**

for each instance $ins \in DRM$ and algorithm $alg \in \{ACO, IBS, DR, CRO\_SCS\}$, $T_{IMCRO}(ins) \leq T_{alg}(ins)$. These results show that IMCRO achieved the minimum average SCS length and minimum average execution time in comparison with the other four algorithms. They also indicate that IMCRO has the best performance for all instances in the random set among the five tested algorithms. Notice that the SD values are also displayed in Table 5 within parenthesis, and these values manifest that IMCRO is more stable than the other algorithms for the random datasets.

Table 7 shows that in the real dataset experiment evaluation, $L_{IMCRO}(ins)$ achieved the minimum values of the average SCS length except when $ins \in \{PROT{-}3, PROT{-}4\}$. Although $L_{CRO\_SCS}(PROT{-}3)$ and $L_{CRO\_SCS}(PROT{-}4)$ got the minimum values, $L_{IMCRO}(PROT-3)$ and $L_{IMCRO}(PROT-4)$ are closest to $L_{CRO\_SCS}(PROT-3)$ and $L_{CRO\_SCS}(PROT-4)$, respectively, which are much smaller than the other average SCS length. Table 8 shows that for each instance $ins = DRL$, $T_{IMCRO}(ins) \leq T_{alg}(ins)$, where $alg \in \{ACO, IBS, DR, CRO\_SCS\}$.

These results indicate that in real datasets, IMCRO has the best performance for DNA instances both in average SCS length and average execution time. For protein instances, although IMCRO and CRO_SCS overwhelm the other algorithms, IMCRO does not reduce the average SCS length considerably in comparison with CRO_SCS. Additionally, results above also indicate that IMCRO can reduce L for DNA instances in comparison with CRO_SCS. Specially, for the DNA instances, the average reduction RC for L can be obtained from Formula 4 and the maximum reduction RC_{max} can be obtained from Formula 5. In DRM, RC is 1.02 and RC_{max} is 1.7, where num = 15. On the other hand, in DRL RC is 1.10 and RC_{max} is 2.1, where num = 6.

Table 7 also shows that for each instance $ins = DRL$, $SD_{CRO\_SCS}(ins)$ and $SD_{IMCRO}(ins)$ are much smaller than $SD_{ACO}(ins)$, $SD_{IBS}(ins)$ and $SD_{DR}(ins)$. It means that IMCRO and CRO_SCS are more stable than ACO, IBS and DR. However, which of $\{CRO\_SCS, IMCRO\}$ is more stable cannot be distinguished.

$$RC = \frac{\sum_{i=1}^{num}(L_{IMCRO}(DNA-i) - L_{CRO\_SCS}(DNA-i))}{num} \tag{4}$$

$$RC_{max} = \max_{i=1}^{num}(L_{IMCRO}(DNA-i) - L_{CRO\_SCS}(DNA-i)) \tag{5}$$

Furthermore, we applied a statistical significance test of L_{IMCRO} on L_{CRO_SCS} by using T-Test [39]. In random dataset (Table 5), the p-value is 3.58×10^{-7}, while in real datasets (Table 7), the p-value is 0.025. Both p-values are smaller than 0.05, which means the difference on the average SCS length between IMCRO and CRO_SCS is statistically significant. Also the PE threshold for each instance and the corresponding iterations for the final solution in the CRO framework are shown in Table 9 and 10. These results suggest that the number of iterations for the CRO operations is less than the maximum iterations (500), and the number of iterations of IMCRO approaches the iterations of CRO_SCS. It indicates that IMCRO converges invariantly with the introduction of the additional operators.

5. Conclusions

In this paper we proposed an improved CRO algorithm, abbreviated IMCRO, for the SCS problem. IMCRO consists of three stages: initialization, iteration, and the finalization. We introduced two new operators: two-step crossover operator with an inter-molecular reaction, and the circular shift operator for decomposition. Experimental results show that similar to CRO_SCS, IMCRO overwhelms well-known heuristic algorithms such as ACO, DR and IBS in reduction on both the average SCS length and the average execution time when solving the SCS problem. Additionally, when comparing to CRO_SCS, IMCRO improves and the performance by reducing the average SCS length for DNA sequences.

Table 9: Iteration in random datasets

n	k	threshold	CRO_SCS	IMCRO
5	10	10	13	11
10	10	25	27	26
50	10	3	5	4
100	10	1	3	2
5	100	40	41	41
10	100	40	41	41
50	100	10	11	11
100	100	7	10	8
500	100	1	2	2
1000	100	1	2	2
5000	100	1	3	2
100	1000	55	57	56
500	1000	10	11	11
1000	1000	4	6	5
5000	1000	1	2	1

Table 10: Iterations in real datasets

name	n	k	threshold	CRO_SCS	IMCRO
DNA-1	100	500	20	205	201
DNA-2	500	500	6	61	61
DNA-3	100	1000	30	301	301
DNA-4	500	1000	16	161	161
DNA-5	100	100	3	6	4
DNA-6	500	100	41	45	41
PROT-1	100	500	40	402	401
PROT-2	500	500	8	82	81
PROT-3	1000	500	4	43	41
PROT-4	100	100	5	52	51
PROT-5	500	100	2	21	21

6. Acknowledgments

This work is partially supported by the project on Educational Teaching Law and Method of East China University of Technology, the Online Education Fund (pervasive education) of Online Education Research Center in Chinese Ministry of Education (No. 2017YB122), and the National Natural Science Foundation of China (NSFC) (No. 61472139).

References

[1] K. Räihä, E. Ukkonen, The shortest common supersequence problem over binary alphabet is np-complete, Theor. Comput. Sci. 16 (1981) 187–198.

[2] K. Mangal, R. Kumar, A recursive algorithm for generalized constraint scs problem, National Academy Science Letters 39 (4) (2016) 273–276.

[3] V. G. Timkovskii, Complexity of common subsequence and supersequence problems and related problems, Cybernetics 25 (5) (1989) 565–580.

[4] D. E. Foulser, M. Li, Q. Yang, Theory and algorithms for plan merging, Artif. Intell. 57 (2) (1992) 143–181.

[5] J. S. Sim, K. Park, The consensus string problem for a metric is np-complete, J. Discrete Algorithms 1 (1) (2003) 111–117.

[6] R. W. Irving, C. Fraser, On the worst-case beha-viour of some approximation algorithms for the shortest common supersequence of k strings, Proceedings of the 4th Annual Symposium on Combinatorial Pattern Matching (1993) 63–73.

[7] S. Rajendran, C. Rajendran, H. Ziegler, An ant-colony algorithm to transform jobshops into flowshops: A case of shortest-common-supersequence stringology problem, in: Bio-Inspired Models of Network, Information, and Computing Systems - 5th International ICST Conference, BIONETICS 2010, Boston, MA, USA, December 1-3, 2010, Revised Selected Papers, 2010, pp. 413–424.

[8] A. S. Jaradat, M. M. Noaman, Solving shortest common supersequence problem using artificial bee colony algorithm, International Journal of ACM Jordan 2 (1) (2011) 180–185.

[9] S. R. Mousavi, F. Bahri, F. Tabataba, An enhanced beam search algorithm for the shortest common supersequence problem, Eng. Appl. of AI 25 (3) (2012) 457–467.

[10] K. Ning, H. W. Leong, Towards a better solution to the shortest common supersequence problem: the deposition and reduction algorithm, BMC Bioinformatics 7 (Suppl 4) (2006) S12–S22.

[11] C. M. K. Saifullah, M. R. Islam, Chemical reaction optimization for solving shortest common supersequence problem, Computational Biology and Chemistry 64 (2016) 82–93.

[12] Q. Yang, Z. Yang, G. Hu, W. Du, A new fusion chemical reaction optimization algorithm based on random molecules for multi-rotor uav path

planning in transmission line inspection, Journal of Shanghai Jiaotong University (Science) 23 (5) (2018) 671–677.

[13] Z. Li, T. T. Nguyen, S. Chen, T. K. Truong, A hybrid algorithm based on particle swarm and chemical reaction optimization for multi-object problems, Appl. Soft Comput. 35 (2015) 525–540.

[14] Z. Li, T. Yuan, B. Yang, S. Jiang, Y. Xie, Ebcro: Hybrid chemical reaction with employed bee operator, in: 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), 2017, pp. 192–200.

[15] Z. L. Ransikarn Ngam, Bat-mutation chemical reaction optimization algorithm for conflict optimization problem: Case of bandwidth utilization, Journal of Computational and Theoretical Nanoscience 14 (2017) 5118–5127.

[16] D. Maier, The complexity of some problems on subsequences and supersequences, J. ACM 25 (2) (1978) 322336.

[17] En-hui Yang, Zhen Zhang, The shortest common superstring problem: average case analysis for both exact and approximate matching, IEEE Transactions on Information Theory 45 (6) (1999) 1867–1886.

[18] R. Bhatia, S. Khuller, J. Naor, The loading time scheduling problem, in: Proceedings of IEEE 36th Annual Foundations of Computer Science, 1995, pp. 72–81.

[19] K. Ning, H. K. Ng, H. W. Leong, Finding patterns in biological sequences by longest common subsequencesand shortest common supersequences, in: Sixth IEEE Symposium on BioInformatics and BioEngineering (BIBE'06), 2006, pp. 53–60.

[20] A. Garg, D. Garg, Progressive alignment using shortest common supersequence, in: 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2014, pp. 1113–1117.

[21] M. Buzdalov, F. Tsarev, An evolutionary approach to hard test case generation for shortest common superstring problem, in: 2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence, 2013, pp. 81–85.

[22] F. Schwiegelshohn, M. Hbner, An application scenario for dynamically reconfigurable fpgas, in: 2014 9th International Symposium on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC), 2014, p-p. 1–8.

[23] J. E. Gallardo, C. Cotta, A. J. Fernandez, On the hybridization of memetic algorithms with branch-and-bound techniques, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 37 (1) (2007) 77–83.

[24] A. Y. S. Lam, V. O. K. Li, Chemical reaction optimization: a tutorial - (invited paper), Memetic Computing 4 (1) (2012) 3–17.

[25] M. R. Islam, C. M. K. Saifullah, M. R. Mahmud, Chemical reaction optimization: survey on variants, Evolutionary Intelligence 12 (3) (2019) 395–420.

[26] M. R. Islam, M. R. Mahmud, R. M. Pritom, Transportation scheduling optimization by a collaborative strategy in supply chain management with tpl using chemical reaction optimization, Neural Computing and Applications 32 (2020) 3649–3674.

[27] J. Li, H. Sang, Q. Pan, P. Duan, K. Gao, Solving multi-area environmental/ economic dispatch by pareto-based chemical-reaction optimization algorithm, IEEE CAA J. Autom. Sinica 6 (5) (2019) 1240–1250.

[28] Y. Fu, M. Zhou, X. Guo, L. Qi, Artificial-molecule-based chemical reaction optimization for flow shop scheduling problem with deteriorating and learning effects, IEEE Access 7 (2019) 53429–53440.

[29] M. R. Islam, I. H. Arif, R. H. Shuvo, Generalized vertex cover using chemical reaction optimization, Appl. Intell. 49 (7) (2019) 2546–2566.

[30] M. R. Islam, R. A. Smrity, S. Chatterjee, M. R. Mahmud, Optimization of protein folding using chemical reaction optimization in hp cubic lattice model, Neural Computing and Applications 32 (2020) 3117–3134.

[31] Z. Li, Y. Li, T. Yuan, S. Chen, S. Jiang, Chemical reaction optimization for virtual machine placement in cloud computing, Appl. Intell. 49 (1) (2019) 220–232.

[32] H. Alrezaamiri, A. Ebrahimnejad, H. Motameni, Software requirement optimization using a fuzzy artificial chemical reaction optimization algorithm, Soft Comput. 23 (20) (2019) 9979–9994.

[33] N. P. Gopalan, T. S. Murthy, Association rule hiding using chemical reaction optimization, in: Soft Computing for Problem Solving - SocProS 2017, Volume 1, Bhubaneswar, India, December 23-24, 2017, 2017, pp. 249–255.

[34] C. M. Khaled Saifullah, M. R. Islam, M. R. Mahmud, Chemical reaction optimization algorithm for word detection using pictorial structure, in: Emerging Technologies in Data Mining and Information Security, 2019, pp. 427–440.

[35] M. R. Kabir, F. T. Zahra, M. R. Islam, Rna structure prediction using chemical reaction optimization, in: Emerging Technologies in Data Mining and Information Security, 2019, pp. 587–598.

[36] M. R. Islam, C. M. K. Saifullah, Z. T. Asha, R. Ahamed, Chemical reaction optimization for solving longest common subsequence problem for multiple string, Soft Computing 23 (14) (2019) 5485–5509.

[37] A. Y. S. Lam, V. O. K. Li, Chemical-reaction-inspired metaheuristic for optimization, IEEE Trans. Evolutionary Computation 14 (3) (2010) 381–399.

[38] M. Papapetrou, D. Kugiumtzis, Investigating long range correlation in dna sequences using significance tests of conditional mutual information, Computational Biology and Chemistry 53 (2014) 32–42.