An Opposition-Based Learning CRO Algorithm for Solving the Shortest Common Supersequence Problem

Fei Luo 1, Cheng Chen 1, Joel Fuentes 2, Yong Li 1,* and Weichao Ding 1,*

1 School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China; luo.f@ecust.edu.cn (F.L.); chen.cheng@yintech.cn (C.C.)
2 Department of Computer Science and Information Technologies, Universidad del Bío-Bío, Chillán 3780000, Chile; fuentes@ubiobio.cl
* Correspondence: yongli@ecust.edu.cn (Y.L.); weich@ecust.edu.cn (W.D.)

Abstract: As a non-deterministic polynomial hard (NP-hard) problem, the shortest common supersequence (SCS) problem is normally solved by heuristic or metaheuristic algorithms. One type of metaheuristic algorithms that has relatively good performance for solving SCS problems is the chemical reaction optimization (CRO) algorithm. Several CRO-based proposals exist; however, they face such problems as unstable molecular population quality, uneven distribution, and local optimum (premature) solutions. To overcome these problems, we propose a new approach for the search mechanism of CRO-based algorithms. It combines the opposition-based learning (OBL) mechanism with the previously studied improved chemical reaction optimization (IMCRO) algorithm. This upgraded version is dubbed OBLIMCRO. In its initialization phase, the opposite population is constructed from a random population based on OBL; then, the initial population is generated by selecting molecules with the lowest potential energy from the random and opposite populations. In the iterative phase, reaction operators create new molecules, where the final population update is performed. Experiments show that the average running time of OBLIMCRO is more than 50% less than the average running time of CRO_SCS and its baseline algorithm, IMCRO, for the desoxyribonucleic acid (DNA) and protein datasets.

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

1. Introduction
As a well-known non-deterministic polynomial hard (NP-hard) problem [1], the shortest common supersequence (SCS) problem has been widely applied in real life and in many bioinformatics and computer science fields, including pattern finding [2], multiple sequence alignment problems [3], deoxyribonucleic acid (DNA) sequencing [4], data compression [5], and artificial intelligence (AI) planning [6]. To find the optimal solution for the SCS problem many different studies have been proposed, including artificial bee colonies (ABC) [7], multiple sequence alignment problems [3], deoxyribonucleic acid (DNA) sequencing [4], data compression [5], and artificial intelligence (AI) planning [6]. To find the optimal solution for the SCS problem different studies have been proposed, including artificial bee colonies (ABC) [7], ant colony optimization (ACO) [8], enhanced beam search (IBS) [9], deposition and reduction (DR) [10], and the CRO_SCS (chemical reaction optimization (CRO) for the SCS problem) algorithm [11].

Compared to other heuristic algorithms, CRO algorithms have achieved better results in solving the SCS problem. The CRO algorithm is inspired by the process of obtaining low-potential energy molecules from chemical reactions in nature. The algorithm calls the solution of the problem a molecule, which is obtained from an iterative process in which four reaction operators are used to generate new molecules performing local and global search. To solve the SCS problem, Saifullah et al. proposed the CRO_SCS algorithm, a CRO-based proposal with the inclusion of a check and repair function to repair molecules after iterations [11]. The operator checks each character of the generated superstring and...
records the number of illegal characters. If the number of violations exceeds a certain standard, the string is discarded. If the number of violations is less than a predefined value, it is repaired. Compared to AOC, ABC, IBS and DR, CRO_SCS is able to shorten the average length of SCS and reduce the average running time.

In order to further improve the performance of the CRO algorithm in solving the SCS problem, a new algorithm based on CRO_SCS, the IMCRO algorithm, was proposed in [12]. It introduced the circular shift operator and the two-step crossover operator in the decomposition reaction and the intermolecular collision reaction. Compared to CRO_SCS, the average length of the shortest common superstring calculated by IMCRO is shorter by 1.02 characters on average.

Even though these CRO-based algorithms have shown important performance gains, they can only perform iterative searches based on the current solution. On the one hand, they are affected by the randomness of the initial population, while on the other they face the problems of slow convergence and premature convergence [13]. The opposition-based learning (OBL) mechanism has arisen as a powerful technique for overcoming these problems. In OBL, the current solution and the reverse solution are examined at the same time during the search process, which improves the diversity of the population and at the same time expands the search range of the algorithm. By doing this the algorithm has more opportunities to escape from the current search area, improving its global search ability. This can effectively solve the premature optimization problem of CRO-based algorithms and achieve convergence in fewer iterative steps, which accelerates the convergence speed [14].

This article introduces the OBL into IMCRO, proposing an opposition-based learning CRO algorithm dubbed OBLIMCRO. The main contribution of this paper is as follows:

- Proposing an OBL function to solve the SCS problem by generating an opposite solution corresponding to the original one. When a new solution is produced, its reverse solution will be generated by the OBL function.
- Introducing the OBL mechanism into the construction of the population by mixing random molecules and opposite molecules, which can be used to optimize the quality of the population and enrich the diversity of the population.
- Proposing an improved CRO-based algorithm with the OBL mechanism in the initialization stage and the iteration stage, resulting in speeding up the convergence and solving the premature trap.

The rest of the paper is organized as follows: related works are summarized in Section 2; the design details of OBLIMCRO are presented in Section 3, where the OBL mechanism for population generation and updating is depicted; to illustrate the performance of OBLIMCRO, an evaluation strategy is defined and the resulting experiments and analysis are shown in Section 3.2.3; finally, the conclusions of the paper are drawn in Section 5.

2. Related Work

David Maier first defined the SCS problem in 1976 [15], which was proved to be NP-complete for sequences over an alphabet of size five. It has been widely used in bioinformatics and computer science. Different algorithms have been proposed to solve the SCS problem, including heuristic algorithms such as [8], ABC [7], IBS [9], DR [10], and CRO_SCS [11].

The CRO algorithm was originally proposed by Lam and Li in 2010 [16], and was inspired by the process of chemical reactions. In short, it is based on the fact that a chemical reaction undergoes sub-reactions where certain intermediate states correspond to a reaction. The molecule becomes more stable when the molecule’s energy is lower than it was in the prior state.

Following the CRO principle, the CRO_SCS algorithm effectively solves the SCS problem. Compared to other heuristic algorithms, the average length of the shortest common superstring obtained by CRO_SCS is smaller under the same conditions. However, as a heuristic algorithm it faces the problems of slower convergence speed and premature maturity. The IMCRO algorithm [12] is an improved algorithm based on CRO_SCS. It introduces
the two-step crossover and circular shift operators for the intermolecular collision and the decomposition reactions, respectively. The use of the reaction operator expands the degree of molecular structure change, improving overall algorithm performance. The premature problem is addressed by the algorithm’s local search and global search capabilities. Compared to CRO_SCS, the average length of the shortest common superstring output by IMCRO is reduced by 1.02 characters.

By combining the CRO with other heuristic algorithms, many novel CRO-based algorithms have been proposed, including random molecule-based chemical reaction optimization (RMCRO) [17], particle swarm and CRO (HP-CRO) [18], employed bee operator-based CRO (EBCRO) [19] and bat mutation-based CRO (BMCRO) [20].

OBL is an optimization strategy used in machine learning that was proposed by Tizhoosh in 2005. The main idea of the algorithm is to simultaneously evaluate both the current solution and its opposite solution during the search process. The algorithm then selects the solution closer to the optimal solution from among both the current and opposite solutions. It then performs subsequent searches, reaching the optimal solution in fewer iterations and speeding up the search process.

Meta-heuristic algorithms use OBL to obtain better results in terms of optimizing the population quality and accelerating the speed of convergence. The work presented in [21] combines the harmony search algorithm (HS) and the OBL mechanism to construct the opposition-based learning global harmony search algorithm (OLGHS). The OBL mechanism participates in the establishment of the initial harmony library, and is considered in the initialization process. For forward and reverse solutions, it chooses the proper solution to establish the initial harmony library. This is in order to ensure that the quality of the initial harmony library is appropriate as well as to solve the premature problem by optimizing the quality of the harmony library.

Several novel CRO algorithms combined with OBL have been proposed. Examples include the quasi-opposition learning chemical reaction optimization algorithm (QOCRO) [13], used to solve the reaction energy scheduling problem, and the elite opposition-based learning chemical reaction difference (EOLCRDE) optimization algorithm [22], used to solve the power dispatch problem.

To the best of our knowledge, the OBL mechanism has not previously been used to solve SCS problems. This work introduces OBL in IMCRO (our previous work) to solve the aforementioned problem.

3. OBLIMCRO

3.1. Opposition-Based Learning

The OBL strategy included in IMCRO is intended to generate “high-quality” solutions to accelerate the iteration speed. In each iteration of the heuristic algorithm, when a new solution (e.g., the forward solution) is produced its reverse solution is provided by OBL as well. The forward solution and the reverse solution are compared and the better one joins the population. This selection process is performed before starting the next iteration, and is equivalent to expanding the search range.

Notice that by applying the OBL mechanism the quality of the population is optimized and the diversity of the population is enriched. This results in improving the algorithm’s global search ability, allowing it to avoid falling into the local optimum trap.

3.2. OBLIMCRO Framework

The same as IMCRO, OBLIMCRO has three stages, initialization, iteration, and finalization, as shown in Figure 1. The general procedure is described in Algorithm 1, while the stages of the framework are detailed in the following subsections.
Figure 1. Framework of OBLIMCRO.
Algorithm 1 Framework of OBLIMCRO

Input: Sets of populations.
1: Initialization of parameters, such as buffer, Initial KE, PopSize, KElossRate, MoleColl, \( \beta \) and \( \alpha \).
2: Create PopSize number of random molecules
3: Calculate opposite molecules of random molecules
4: Mix random molecules and opposite molecules
5: Select PopSize molecules with lowest potential energy from the mix set
6: while the stopping criteria is not met do
7: Generate \( t \in [0, 1] \)
8: if \((\text{NumHit} - \text{MinHit}) > \alpha\) then
9: Randomly select one molecule \( m \)
10: if \((\text{NumHit} - \text{MinHit}) > \alpha\) then
11: Trigger Decomposition
12: else
13: Trigger On-wall Ineffective Collision
14: end if
15: else
16: Randomly select two molecules \( m_1 \) and \( m_2 \)
17: if \( \text{KE} \leq \beta \) then
18: Trigger Synthesis
19: else
20: Trigger Inter-molecular ineffective collision
21: end if
22: end if
23: Check for new better solution
24: Calculate opposite molecule
25: Choose the solution with lower PE from the solution and its opposite molecule
26: Update population with the chosen solution
27: end while

Output: Best solution.

3.2.1. Initialization Stage

Initialization is the first stage of OBLIMCRO, where the molecules are initialized. Initial values of molecules include PopSize, KElossRate, MoleColl, buffer, Initial KE, \( \alpha \), and \( \beta \). Therein, PopSize is the number of all feasible solutions, KElossRate is the maximum percentage of KE reduction, MoleColl is used to determine whether the chemical reaction is inter-molecule or uni-molecule, KE is the initial kinetic energy of the molecule, and \( \alpha \) and \( \beta \) are the thresholds for intensification and diversification.

Molecules in the initial population group are generated with OBL. First, the initial molecule group, \( RX = \{x_1, x_2, \ldots, x_n\} \), is randomly generated; then, its opposite group, \( OX = \{ox_1, ox_2, \ldots, ox_n \mid ox_i = \text{obl}(x_i)\} \), is generated with the function \( \text{obl} \) based on the OBL mechanism. Afterwards, the random and opposite group are mixed as \( MX \), where \( MX = RX \cup OX \). The individuals in \( MX \) are sorted according to their potential energy and the PopSize molecules with the lowest potential energy are selected to form the initial population, denoted as \( IX \). Figure 2 shows an example of the initialization process, where the number of molecules in \( RX \), \( OX \), \( MX \), and \( IX \) is PopSize.

The construction of \( RX \), \( OX \), \( MX \), and \( IX \) takes place in the initialization stage. \( RX \) is constructed with a random inserting operation, as depicted in Figure 3. Therein, the initial superstring \( C \) is randomly generated as an array the elements of which are composed of letters in the string alphabet. The element of \( S_i \) is randomly inserted into \( C \), where \( S_i \) is one string of the input string set \( S \) and \( 1 \leq i \leq \text{PopSize} \). This random inserting operation is performed on all strings in \( S \) to obtain the initial population with PopSize molecules. For further processing, each supersequence is encoded as a set of integer values. For instance, \( \Sigma \) is an alphabet set and \( \Sigma = \{a, c, g, t\} \). It is encoded as \( I = \{0, 1, 2, 3\} \), where each element
of $\Sigma$ is encoded as an integer in $I$ in the corresponding position of the set. Then, the set \{0, 1, 2, 3, 2, 1, 3, 0\} can represent the supersequence $E = \{a, c, t, g, t, c, g, a\}$, as shown in Figure 4.

OX is obtained by the OBL mechanism after the initialization of RX. For each molecule $X$ with $n$ elements in RX, where $X = \{x_1, x_2, \ldots, x_n\}$, its opposite molecule OX is obtained using the function $obl()$, as defined in Formula (1). Therein, the function $o(x)$ obtains the inverse number of the integer $x$. For a real number $x$ the feasible region of which is $[a,b]$, its inverse number $ox$ is defined in Formula (2). Taking a molecule $X = \{0, 1, 2, 3, 2, 1, 3, 0\}$ with a feasible region $[0,3]$ as an example, according to Formula (1), the opposite molecule is $OX = \{3, 2, 1, 0, 1, 2, 0, 3\}$.

$$OX = obl(X) = obl(\{x_1, x_2, ..., x_n\}) = \{o(x_1), o(x_2), ..., o(x_n)\}$$  \hspace{1cm} (1)

$$ox = o(x) = a + b - x$$ \hspace{1cm} (2)

After obtaining RX and OX, MX is obtained by performing the collective union operation on RX and OX, where $MX = RX \cup OX = \{x_1, x_2, \ldots, x_n, ox_1, ox_2, \ldots, ox_n\}$ with the scale as $2^{*}\text{PopSize}$. The molecules are sorted according to the molecular potential energy of the molecules in MX. Half of the molecules in MX with the lowest potential energy are selected to construct IX, which is the initial molecule population.

![Diagram](attachment:image.png)

**Figure 2.** Initialization based on OBL.
3.2.2. Iteration Stage

The iteration stage consists of two subtasks, reaction and repair. There are four main operators used in the action subtask: on-wall ineffective collision, decomposition, inter-molecular ineffective collision, and synthesis. On-wall ineffective collision is realized by the collision of a single molecule. Then, one element of the molecule is randomly selected and changed to form a new molecule. The operator is used for local search. Decomposition refers to a molecule colliding with the wall and splitting into two or more molecules, and is usually used for global search. Inter-molecular ineffective collision is similar to on-wall ineffective collision. Synthesis is the combination of two molecules into a new molecule, which is equivalent to the inverse reaction of decomposition.

The four operators fall into two types, uni-molecule reaction and inter-molecule reaction. Therein, uni-molecule reactions include on-wall ineffective collision and decomposition, while inter-molecule reactions include inter-molecular ineffective collision and synthesis. The four operators in this stage are the same as those defined in [12].

At the start of each iteration step, a parameter \( t \) is randomly generated. This parameter determines whether uni-molecule reactions or inter-molecule reactions are triggered. In particular, uni-molecule reactions are triggered when \( t > \text{MoleColl} \); otherwise, inter-molecule reactions are triggered. The specific operator is further determined according to values of \( \alpha \).
and $\beta$. In uni-molecule reactions, decomposition is triggered if $(\text{NumHit} - \text{MinHit}) > \alpha$; otherwise, on-wall ineffective collision is triggered. In inter-molecule reactions, synthesis is triggered if $KE \leq \beta$; otherwise, inter-molecular ineffective collision is triggered.

Through the processing of the chemical operators a new molecule, $X_i'$, is created based on the molecule $X_i$ and an updating process is triggered, as shown in Algorithm 2. First, the opposite molecule, $OX_i'$, is created according to Formula (1). The potential energy of $X_i'$ and $OX_i'$ are compared and the molecule with lower potential energy is chosen to be returned to the initial population for updating.

**Algorithm 2** Update based on OBL.

**Input:** new molecule $X_i'$.

1. $OX_i' = \text{obl}(X_i')$;
2. $\text{PE}(X_i') = f(X_i')$;
3. $\text{PE}(OX_i') = f(OX_i')$;
4. if $\text{PE}(X_i') < \text{PE}(OX_i')$ then;
5. $UX_i' = X_i'$;
6. else
7. $UX_i' = OX_i'$;
8. end if
9. if $UX_i'$ matches stopping criteria then
10. output $UX_i'$;
11. else
12. put $UX_i'$ into IX to update the initial group;
13. end if

Similar to that in IMCRO, a repair function to check this solution is triggered when a solution is obtained. If the requirement of the problem is satisfied, the obtained solution is repaired by a repair algorithm until the termination condition is met. Then, it enters the final stage, where the best solution is output and returned when the stopping criteria are met; otherwise, iteration is triggered again and the reactions are repeated.

3.2.3. Finalization

At the end of each iteration, termination conditions are checked and a new molecule is obtained through the OBL-based updating process (Algorithm 2). If the conditions are met, either a new molecule is output as the result or the iteration continues. There are two termination conditions. One is that the potential energy of the new molecule is less than the specified threshold, and the other is that the number of iterations reaches a certain upper limit. If either of the two conditions is met, it is deemed to have met the termination condition.

4. Experiments and Evaluation

In order to evaluate the performance and efficiency of OBLIMCRO for solving the SCS problem, we carried out a set of experiments where OBLIMCRO was compared to IMCRO and CRO_SCS, as well as to heuristic algorithms such as ACO, IBS, and DR.

4.1. Configuration of Experiments

The execution environment for the compared algorithms was a Windows-based personal computer with an i5-4210U CPU and 4.0GB RAM. We adopted the same parameters for the experiments as those in IMCRO [12]. In particular, MoleColl was 0.2, KELossRate was 0.6, and PopSize was 20.

The datasets were the same as those used in IMCRO [12], including a random DRM dataset with fifteen instances and a real DRL dataset with eleven instances. There were fifteen DNA sequences in the random DRM dataset, with each instance containing four different types of characters ($\Sigma = 4$). There were six DNA sequences and five protein
sequences in the real DRL dataset. There were twenty different types of characters ($\Sigma = 20$) in each protein sequence.

In the experiment, each compared algorithm was run twenty times with the different datasets. For each instance we obtained 200 different results, including the length of SCS and the execution time. The average length and the average execution time for the SCS were obtained from these 200 results. After repeating the process described above, results such as the average SCS length, represented as $L$, and the average execution time, represented as $T$, were obtained. The value of $L$ of the specialized algorithm $\text{alg}$ for the instance $\text{ins}$ is denoted as $L_{\text{alg}}(\text{ins})$. Meanwhile, the value of $T$ of the specialized algorithm $\text{alg}$ for the instance $\text{ins}$ is denoted as $T_{\text{alg}}(\text{ins})$; $\text{alg}$ is one of the compared algorithms, e.g., $\text{alg} \in \{\text{DR, IBS, ACO, CRO-SCS, IMCRO, OBLIMCRO}\}$, and $\text{ins}$ is one type of the datasets, e.g., $\text{ins} \in \text{DRM} \cup \text{DRL}$.

The maximum number of iterations for the CRO operations and the current potential energy exceeding the threshold were the two stopping criteria for the execution of the compared algorithms. Specifically, the maximum number of iterations was 500 and the threshold of the potential energy was specialized according to the structure of the particular instance. As depicted in Algorithm 1, if one of the two criteria were satisfied, the algorithm was stopped and the final solution was output.

4.2. Results and Analysis

The objective of the compared algorithms was to obtain the shortest average SCS length and the minimum average execution time. Tables 1–4 show the results of the algorithms with these two types of datasets. Therein, $n$ is the instance’s string number and $k$ is the string length. The emboldened values in each table are the best results.

Table 1. Average SCS Length in random datasets.

| n   | k  | ACO | DR  | IBS | CRO-SCS | IMCRO | OBLIMCRO |
|-----|----|-----|-----|-----|---------|-------|----------|
| 5   | 10 | 22.5| 21.2| 19.9| 20.2    | 19.7  | 19.6     |
| 10  | 10 | 26.7| 25.1| 25.2| 25.3    | 24.9  | 24.8     |
| 50  | 10 | 31.5| 31.1| 30.0| 29.3    | 28.6  | 28.6     |
| 100 | 10 | 33.0| 32.5| 32.0| 32.1    | 31.5  | 30.5     |
| 5   | 100| 207.4|198.2|184.0|181.6   |180.5  |179.8     |
| 10  | 100| 233.7|226.2|210.0|209.4   |208.3  |207.6     |
| 50  | 100| 263.7|262.0|252.0|244.4   |243.8  |241.4     |
| 100 | 100| 270.1|269.2|261.1|252.1   |251.0  |247.3     |
| 500 | 100| 277.2|277.8|273.6|267.6   |266.7  |266.5     |
| 1000| 100| 281.7|278.5|276.8|270.1   |269.7  |269.7     |
| 5000| 100| 282.9|282.9|281.5|271.9   |270.6  |269.9     |
| 1000| 1000|2535.6|2531.6|2466.7|2443.2  |2442.1 |2440.1    |
| 5000| 1000|2565.6|2578.8|2540.2|2532.1  |2530.5 |2530.2    |
| 1000| 1000|2570.8|2581.4|2555.5|2535.6  |2533.9 |2531.4    |
| 5000| 1000|2590.6|2586.9|2571.6|2562.9  |2561.8 |2558.7    |
Table 2. Average execution time in random datasets.

| n   | k   | T/s   | ACO | DR   | IBS | CRO_SCS | IMCRO | OBLIMCRO |
|-----|-----|-------|-----|------|-----|---------|-------|----------|
| 5   | 10  | 0.8   | 0.018 | 0.03 | 0.008 | 0.008   | 0.008  | 0.005    |
| 10  | 10  | 1.00  | 0.033 | 0.03 | 0.03  | 0.03    | 0.03   | 0.02     |
| 50  | 10  | 2.3   | 0.1   | 0.07 | 0.08  | 0.065   | 0.034  | 0.034    |
| 100 | 10  | 3.5   | 0.15  | 0.12 | 0.08  | 0.055   | 0.021  | 0.021    |
| 5   | 100 | 5.9   | 0.16  | 0.14 | 0.02  | 0.013   | 0.005  | 0.005    |
| 10  | 100 | 8.6   | 1.18  | 0.22 | 0.14  | 0.12    | 0.05   | 0.05     |
| 50  | 100 | 16.3  | 4.07  | 0.46 | 0.37  | 0.26    | 0.10   | 0.10     |
| 100 | 100 | 23.5  | 7.28  | 0.91 | 0.65  | 0.52    | 0.30   | 0.30     |
| 500 | 100 | 65.5  | 27.3  | 3.06 | 1.69  | 0.92    | 0.28   | 0.28     |
| 1000| 100 | 127.9 | 69.2  | 6.45 | 2.66  | 1.95    | 0.56   | 0.56     |
| 5000| 100 | 230   | 109   | 9.09 | 4.21  | 2.93    | 0.68   | 0.68     |

Table 3. Average SCS Length in real datasets.

| Name   | n    | k    | L   | ACO | DR   | IBS | CRO_SCS | IMCRO | OBLIMCRO |
|--------|------|------|-----|-----|------|-----|---------|-------|----------|
| DNA-1  | 100  | 500  | 1346.9 | 1332.6 | 1280.7 | 1271.4 | 1271.0 | 1270.0   |
| DNA-2  | 500  | 500  | 1520.0 | 1404.6 | 1352.7 | 1351.8 | 1350.8 | 1349.7   |
| DNA-3  | 100  | 1000 | 2712.2 | 2670.1 | 2542.9 | 2442.5 | 2440.8 | 2438.9   |
| DNA-4  | 500  | 1000 | 3092.1 | 2782.7 | 2664.4 | 2532.4 | 2530.3 | 2530.2   |
| DNA-5  | 100  | 100  | 297.8  | 285.4  | 272.3  | 252.1  | 251.5  | 250.6    |
| DNA-6  | 500  | 100  | 405.2  | 291.5  | 288.3  | 267.2  | 266.3  | 266.2    |
| PROT-1 | 100  | 500  | 6908.2 | 4851.4 | 4349.7 | 4312.4 | 4311.6 | 4310.5   |
| PROT-2 | 500  | 500  | 8910.4 | 5545.2 | 5229.3 | 5041.1 | 5040.8 | 5028.2   |
| PROT-3 | 1000 | 500  | 11086  | 5748.7 | 5395.6 | 5301.7 | 5301.9 | 5302.1   |
| PROT-4 | 100  | 1000 | 1303.5 | 1005.6 | 913.3  | 920.9  | 921.2  | 921.8    |
| PROT-5 | 500  | 1000 | 1776.2 | 1205.1 | 1126.7 | 1125.9 | 1126.7 |          |

Table 4. Average execution time in real datasets.

| Name   | n    | k    | T/s   | ACO | DR   | IBS | CRO_SCS | IMCRO | OBLIMCRO |
|--------|------|------|-------|-----|------|-----|---------|-------|----------|
| DNA-1  | 100  | 500  | 151.3 | 349.4 | 3.00  | 1.91 | 1.91    | 0.33  | 0.33     |
| DNA-2  | 500  | 500  | 613.1 | 540.6 | 17.14 | 8.35 | 7.83    | 1.91  | 1.91     |
| DNA-3  | 100  | 1000 | 344.4 | 483.6 | 10.31 | 5.85 | 5.48    | 1.43  | 1.43     |
| DNA-4  | 500  | 1000 | 1514.1| 1156.5| 42.4  | 15.7 | 15.46   | 6.36  | 6.36     |
| DNA-5  | 100  | 100  | 41.97 | 37.55 | 0.92  | 0.56 | 0.49    | 0.23  | 0.23     |
| DNA-6  | 500  | 100  | 92.0  | 37.55 | 2.95  | 0.81 | 0.73    | 0.34  | 0.34     |
| PROT-1 | 100  | 500  | 560.3 | 1125.3| 92.4  | 31.98| 21.18   | 8.90  | 8.90     |
| PROT-2 | 500  | 500  | 1450.2| 1905.4| 307.8 | 52.29| 50.50   | 24.01 | 24.01    |
| PROT-3 | 1000 | 500  | 3205.4| 4002.7| 1905.6| 116.12| 110.76  | 62.46 | 62.46    |
| PROT-4 | 100  | 100  | 16.4  | 31.2  | 13.5  | 1.66 | 1.54    | 0.94  | 0.94     |
| PROT-5 | 500  | 1000 | 123.5 | 95.7  | 65.3  | 4.63 | 4.53    | 2.69  | 2.69     |

Table 1 shows that \( L_{OBLIMCRO}(\text{ins}) < L_{\text{alg}}(\text{ins}) \), where \( \text{ins} \in \text{DRM} \) and \( \text{alg} \in \{\text{DR, IBS, ACO, CRO}_S\text{CS, IMCRO}\} \). Table 2 shows that \( T_{OBLIMCRO}(\text{ins}) \leq T_{\text{alg}}(\text{ins}) \), where \( \text{ins} \in \text{DRM} \) and \( \text{alg} \in \{\text{DR, IBS, ACO, CRO}_S\text{CS, IMCRO}\} \). The results show that when compared with DR, IBS, ACO, CRO_SCS and IMCRO, OBLIMCRO obtained the minimum average SCS length and minimum average execution time.
Table 3 shows that $L_{OBLIMCRO}(ins) < L_{alg}(ins)$, where $ins \in DRL$ except $ins \in \{PROT - 3, PROT - 4, PROT - 5\}$ and $alg \in \{DR, IBS, ACO, CRO\_SCS, IMCRO\}$. Meanwhile, $L_{OBLIMCRO}(PROT - 3), L_{OBLIMCRO}(PROT - 4)$, and $L_{OBLIMCRO}(PROT - 5)$ are closest to $L_{IMCRO}(PROT - 3), L_{IMCRO}(PROT - 4),$ and $L_{CRO\_SCS}(PROT - 5)$, respectively, which are the smallest values. Table 4 shows that $T_{OBLIMCRO}(ins) \leq T_{alg}(ins)$, where $ins \in DRL$ and $alg \in \{DR, IBS, ACO, CRO\_SCS, IMCRO\}$.

The results shown in Tables 1 and 3 indicate that OBLIMCRO obtains the minimum value in both average SCS length and average execution time for DNA instances in real datasets. However, OBLIMCRO does not considerably overwhelm IMCRO and CRO\_SCS for protein instances, while OBLIMCRO, IMCRO and CRO\_SCS achieve smaller values of the average SCS length in comparison with the other algorithms. In particular, in comparison with IMCRO OBLIMCRO can reduce $L$ for DNA instances, as shown in Formula (3), where $RC$ is the average reduced value. In DRM, $RC$ with OBLIMCRO is 2.11, where num = 15. On the other hand, RC with OBLIMCRO is 2.24 in DRL, where num = 6.

Except for the reduction in average SCS length, the reduction in average running time is better. In the random set, the average running time of the OBLIMCRO algorithm is the lowest among the three CRO algorithms. Compared with the CRO\_SCS algorithm, after using the OBLIMCRO algorithm the average running time is shortened by 52.8%. Compared with IMCRO, after using the OBLIMCRO algorithm the average running time is shortened by 50.9%. In the real set, a substantial improvement in the average running time was found. Compared with CRO\_SCS, the average running time of OBLIMCRO is shortened by 54.3%, and compared with IMCRO, the running time of OBLIMCRO is shortened by 50.3%. It can be seen that whether using experimental data in a random set or using a real set, after using OBLIMCRO the running time can be shortened by more than half when compared with CRO\_SCS and IMCRO. The average reduction in average running time, $\Delta T$, for $T$ can be obtained using Formula (4).

\[
RC = \frac{\sum_{i=1}^{num} (L_{IMCRO/CRO\_SCS}(DNA - i) - L_{OBLIMCRO}(DNA - i))}{num}
\]

\[
T = \frac{T_{IMCRO/CRO\_SCS}(i) - T_{OBLIMCRO}(i)}{T_{IMCRO/CRO\_SCS}(i)}
\]

To prove the credibility of the results above, we applied a statistical significance test to $L_{OBLIMCRO}$ on $L_{CRO\_SCS}$ and $L_{IMCRO}$ using $t$-tests [23]. The $p$-value of the $t$-test was $2.83 \times 10^{-5}$ and 0.0013 for the random datasets (Table 1), respectively, and 0.04 and 0.0164 for the real datasets (Table 3), respectively. All of these $p$-values are smaller than 0.05, which means that the difference in the average SCS length between OBLIMCRO and IMCRO or CRO\_SCS is statistically significant. The results of the $t$-tests statistically verify the credibility of the results, showing that OBLIMCRO can efficiently reduce the average SCS length for most of the datasets in comparison with IMCRO and CRO\_SCS.

Next, we applied a statistical significance test to $T_{OBLIMCRO}$ on $T_{CRO\_SCS}$ and $T_{IMCRO}$ using $t$-tests, as well. For the random dataset (Table 1), the $p$-value of the $t$-test was 0.12 and 0.13 respectively, while for the real datasets (Table 3) it was 0.021 and 0.024 respectively. As all of these $p$-values are smaller than 0.05, the difference in the average running time between OBLIMCRO and IMCRO or CRO\_SCS is statistically significant. The results of the $t$-tests statistically verify the credibility of $t$ results, showing that in comparison with IMCRO and CRO\_SCS, OBLIMCRO can efficiently reduce the running time required to resolve the SCS problem.

The number of iterations required to reach the best solution with the CRO algorithms is shown in Tables 5 and 6, and is related to the PE threshold of each instance. For the three CRO algorithms, the number of CRO operations per iteration is less than 500, which is configured as the maximum iterations. This indicates that the three CRO algorithms converge invariantly.
Table 5. Iterations in random datasets.

| n  | k  | Threshold | CRO, SCS | IMCRO | OBLIMCRO |
|----|----|-----------|----------|-------|----------|
| 5  | 10 | 10        | 13       | 11    | 10       |
| 10 | 10 | 25        | 27       | 26    | 25       |
| 50 | 10 | 3         | 5        | 4     | 4        |
| 100| 10 | 1         | 3        | 2     | 2        |
| 5  | 100| 40        | 41       | 41    | 40       |
| 10 | 100| 40        | 41       | 41    | 41       |
| 50 | 100| 10        | 11       | 11    | 11       |
| 100| 100| 7         | 10       | 8     | 8        |
| 500| 100| 1         | 2        | 2     | 2        |
| 1000|100  | 1     | 2        | 2     | 2        |
| 5000|100  | 55      | 57       | 56    | 56       |
| 1000|1000 | 10     | 11       | 11    | 11       |
| 5000|1000 | 4       | 6        | 5     | 5        |

Table 6. Iterations in real datasets.

| Name  | n     | k     | Threshold | CRO, SCS | IMCRO | OBLIMCRO |
|-------|-------|-------|-----------|----------|-------|----------|
| DNA-1 | 100   | 500   | 20        | 205      | 201   | 150      |
| DNA-2 | 500   | 500   | 6         | 61       | 61    | 50       |
| DNA-3 | 100   | 1000  | 30        | 301      | 301   | 240      |
| DNA-4 | 500   | 1000  | 16        | 161      | 161   | 106      |
| DNA-5 | 100   | 100   | 3         | 6        | 4     | 3        |
| DNA-6 | 500   | 100   | 41        | 45       | 41    | 41       |
| PROT-1| 100   | 500   | 40        | 402      | 401   | 205      |
| PROT-2| 500   | 500   | 8         | 82       | 81    | 71       |
| PROT-3| 1000  | 500   | 4         | 43       | 41    | 33       |
| PROT-4| 100   | 100   | 5         | 52       | 51    | 35       |
| PROT-5| 500   | 100   | 2         | 21       | 21    | 10       |

5. Conclusions

We proposed a novel algorithm by introducing an OBL mechanism into IMCRO to solve the SCS problem. The proposed algorithm, named OBLIMCRO, aims to tackle common problems found in state-of-the-art algorithms for SCS. The OBL mechanism participates in the process of constructing and updating the population, which optimizes the quality of the population that is used to obtain the final solution. Experimental results show that for DNA proteins in both random and real datasets, the average running time is reduced and the average length required to solve the SCS is shortened. The results indicate that with the help of OBL, OBLIMCRO can achieve more optimal solutions with a faster convergence speed.

Author Contributions: Conceptualization, Y.L.; Methodology, W.D.; Supervision, F.L.; Writing—original draft, C.C.; Writing—review editing, J.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the project on the Shanghai Action Plan of Technological Innovation (20DZ1201400, 22ZBR1416500) and sponsored by Shanghai Sailing Program (20YF1410900).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: Yongjun Luo and Yong Li are thanked for expert advice and inspiring discussions.

Conflicts of Interest: The authors declare no conflict of interest.
References

1. Räihä, K.; Ukkonen, E. The Shortest Common Supersequence Problem over Binary Alphabet is NP-Complete. *Theor. Comput. Sci.* 1981, 16, 187–198. [CrossRef]

2. Ning, K.; Ng, H.K.; Leong, H.W. Finding Patterns in Biological Sequences by Longest Common Subsequences and Shortest Common Supersequences. In *Proceedings of the Sixth IEEE Symposium on BioInformatics and BioEngineering (BIBE’06)*, Arlington, VA, USA, 16–18 October 2006; pp. 53–60.

3. Garg, A.; Garg, D. Progressive alignment using Shortest Common Supersequence. In Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, 24–27 September 2014; pp. 1113–1117.

4. Mangal, K.; Kumar, R. A Recursive Algorithm for Generalized Constraint SCS Problem. *Nat. Acad. Sci. Lett.* 2012, 36, 273–276. [PubMed]

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

6. Foulser, D.E.; Li, M.; Yang, Q. Theory and Algorithms for Plan Merging. *Artif. Intell.* 1992, 57, 143–181. [CrossRef]

7. Jaradat, A.S.; Noaman, M.M. Solving Shortest Common Supersequence Problem Using Artificial Bee Colony Algorithm. *Int. J. ACM Jordan* 2011, 2, 180–185.

8. Rajendran, S.; Rajendran, C.; Ziegler, H. 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, Proceedings of the 5th International ICST Conference, BIONETICS 2010, Boston, MA, USA, 1–3 December 2010*; Revised Selected Papers; Springer: Berlin/Heidelberg, Germany, 2010; pp. 413–424.

9. Mousavi, S.R.; Bahri, F.; Tabataba, F. An enhanced beam search algorithm for the Shortest Common Supersequence Problem. *Eng. Appl. AI* 2012, 25, 457–467. [CrossRef] [PubMed]

10. Ning, K.; Leong, H.W. Towards a better solution to the shortest common supersequence problem: The deposition and reduction algorithm. *BMC Bioinform.* 2006, 7, S12–S22. [CrossRef] [PubMed]

11. Saifullah, C.M.K.; Islam, M.R. Chemical reaction optimization for solving shortest common supersequence problem. *Comput. Biol. Chem.* 2016, 64, 82–93. [CrossRef] [PubMed]

12. Luo, F.; Chen, C.; Fuentes, J. An improved chemical reaction optimization algorithm for solving the shortest common supersequence problem. *Comput. Biol. Chem.* 2020, 88, 107327. [CrossRef] [PubMed]

13. Dutta, S.; Paul, S.; Roy, P.K. Optimal allocation of SVC and TCSC using quasi-oppositional chemical reaction optimization for solving multi-objective ORPD problem. *J. Electr. Syst. Inf. Technol.* 2018, 5, 83–98. [CrossRef]

14. Tizhoosh, H. Opposition-Based Learning: A New Scheme for Machine Intelligence. In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC’06), Vienna, Austria, 28–30 November 2005; Volume 1, pp. 695–701.

15. Maier, D. The Complexity of Some Problems on Subsequences and Supersequences. *J. ACM* 1978, 25, 322–336. [CrossRef]

16. Lam, A.Y.; Li, V.O.K. Chemical Reaction Optimization: A tutorial—(Invited paper). *Memetic Comput.* 2012, 4, 3–17. [CrossRef]

17. Yang, Q.; Yang, Z.; Hu, G.; Du, W. A New Fusion Chemical Reaction Optimization Algorithm Based on Random Molecules for Multi-Rotor UAV Path Planning in Transmission Line Inspection. *J. Shanghai Jiaotong Univ. Sci.* 2018, 23, 671–677. [CrossRef]

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

19. Li, Z.; Yuan, T.; Yang, B.; Jiang, S.; Xie, Y. EBCRO: Hybrid chemical reaction with employed bee operator. In *Proceedings of the 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, Guilin, China, 29–31 July 2017; pp. 192–200.

20. Ngam, R.; Li, Z. Bat-Mutation Chemical Reaction Optimization Algorithm for Conflict Optimization Problem: Case of Bandwidth Utilization. *J. Comput. Theor. Nanosci.* 2017, 14, 5118–5127.

21. Zhai, J.; Qin, Y. Opposition-based learning in global harmony search algorithm. *Control. Decis.* 2019, 34, 1449–1455.

22. Fei, H. *The Research of Dynamic Economic Dispatch Integrated with Wind Power System Based on Chemical Reaction Optimization Algorithm*; Hunan University: Changsha, China, 2017.

23. Papapetrou, M.; Kugiumtzis, D. Investigating long range correlation in DNA sequences using significance tests of conditional mutual information. *Comput. Biol. Chem.* 2014, 53, 32–42. [PubMed]