PLOS ONE

RESEARCH ARTICLE

A DNA algorithm for the job shop scheduling problem based on the Adleman-Lipton model

Xiang Tian[1], **Xiyu Liu**[1]*, **Hongyan Zhang**[1], **Minghe Sun**[2], **Yuzhen Zhao**[1]

1 Business School, Shandong Normal University, Jinan, China, **2** College of Business, The University of Texas at San Antonio, San Antonio, TX, United States of America

* xyliu@sdnu.edu.cn

Check for updates

OPEN ACCESS

Citation: Tian X, Liu X, Zhang H, Sun M, Zhao Y (2020) A DNA algorithm for the job shop scheduling problem based on the Adleman-Lipton model. PLoS ONE 15(12): e0242083. https://doi.org/10.1371/journal.pone.0242083

Editor: Shih-Wei Lin, Chang Gung University, TAIWAN

Received: June 16, 2020

Accepted: October 27, 2020

Published: December 2, 2020

Copyright: © 2020 Tian et al. This is an open access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Data Availability Statement: All relevant data are within the manuscript and its Supporting Information files.

Funding: This work was partly supported by the National Natural Science Foundation of China (Nos. 61876101, 61802234, 61806114), the Social Science Fund Project of Shandong (Nos. 11CGLJ22, 16BGLJ06), the Natural Science Foundation of the Shandong Province (No. ZR2019QF007), the Youth Fund for Humanities and Social Sciences, Ministry of Education (No. 19YJCZH244), the China Postdoctoral Special

Abstract

A DNA (DeoxyriboNucleic Acid) algorithm is proposed to solve the job shop scheduling problem. An encoding scheme for the problem is developed and DNA computing operations are proposed for the algorithm. After an initial solution is constructed, all possible solutions are generated. DNA computing operations are then used to find an optimal schedule. The DNA algorithm is proved to have an $O(n^2)$ complexity and the length of the final strand of the optimal schedule is within appropriate range. Experiment with 58 benchmark instances show that the proposed DNA algorithm outperforms other comparative heuristics.

1. Introduction

It is well known that the traditional silicon-based computers use serial algorithms, so that their computing speed cannot qualitatively leap. It is also well known that optimal solutions of most of the celebrated computationally intractable problems can only be found by an exhaustive search through all possible solutions. However, the insurmountable difficulty lies in the fact that such an exhaustive search is too vast to carry out using currently available computing technology, so that numerous intractable problems cannot be solved effectively. Some visionary remarks were made about new ways of solving such problems through possible miniaturizations. Feynman's view [1] was widely accepted, stating the possibility of establishing "sub-microscopic" computers. Since then, although significant progresses have been made in relation to computer miniaturization, the goal of sub-microscopic computers has not yet been achieved.

As a new interdisciplinary area, DNA computing has received increasing attentions. Massive parallelism and huge storage capacity are two significant advantages of DNA computing. Parallelism means DNA computing can perform billions of operations simultaneously. Furthermore, DNA computers can solve more intractable problems, such as non-deterministic polynomial-time) (NP)-hard problems, in linear time, as compared with conventional electronic computers in exponential time. In addition, the high density of data stored in DNA strands and the ease in duplicating them can make such exhaustive searches possible. Adleman's experiment [2] solved the Hamiltonian Path Problem for a given directed graph, and demonstrated the strong parallel computing power of DNA computing. Lipton's DNA-based solution of the satisfiability problem [3] used some of Adleman's basic operations. Indeed, it

Funding Project (No. 2019T120607), and the China Postdoctoral Science Foundation Funded Project (Nos. 2017M612339, 2018M642695).

Competing interests: The authors have declared that no competing interests exist.

used an exhaustive search that was made computationally feasible by the massive parallelism of the DNA strands. Ouyang et al. [4] turned the maximal clique problem, another NP-complete problem, into the maximum independent set problem, and solved the problem with six vertices in the laboratory by using the parallel overlap assembly technology. Roweis et al. [5] introduced a new DNA computing model, i.e., the sticker model, and used this model to solve the minimal set cover problem and the data encryption problem. Furthermore, the self-assembly model [6], the hairpin model [7] and the surface-based model [8, 9] had already been proposed and built.

Among the many DNA computing models mentioned above, the Adleman-Lipton model and the sticker model are most widely used in solving classical combinatorial optimization problems. There are numerous publications in the literature addressing the combinatorial optimization problems using the Adleman-Lipton model. For example, Xiao et al. [10] solved maximum cut problems using the Adleman-Lipton model with $O(n^2)$ steps. Hsieh et al. [11] solved the graph isomorphism problem with the Adleman-Lipton model with stickers using a polynomial number of basic biological operations. Yang et al. [12] proposed a theoretical DNA algorithm to solve the quadratic assignment problem using the Adleman-Lipton-sticker model, which was executed with an $O(kn^4)$ complexity and could handle the medium-sized cases. Nehi and Hamidi [13] corrected and further improved the DNA model proposed by Yang et al. [12]. Wang et al. [14] solved a traveling salesman problem by a DNA algorithm using the Adleman-Lipton model with an $O(n)$ complexity. Based on the Adleman- Lipton model, Wang et al. [15] proposed a new DNA computing algorithm to tackle the capacitated vehicle routing problem with an $O(n^2)$ complexity.

In accordance with the processing order, the shop scheduling problem can typically be divided into three categories: the flow shop scheduling problem (FSSP), the job shop scheduling problem (JSSP) and the flexible job shop scheduling problem (FJSP). These three categories of problems are all about scheduling n jobs with varying processing times on m machines with varying speeds and capacities. In a JSSP, each job to be processed contains multiple operations, each of which is processed on a specified machine, and each job has a different machining path. In a FSSP, all jobs have the same machining path, i.e., the same operation sequence, without distinction between the operation and the machine. In a FJSP, the machining paths of the jobs are not necessarily the same and a job is allowed to be processed by any machine in a given set of machines. Many heuristic approaches have been developed to solve shop scheduling problems, such as particle swarm optimization (PSO), genetic algorithms (GA), simulated annealing (SA), tabu search (TS), artificial immune (AI), differential evolution algorithm (DEA), and ant colony optimization (ACO), among others, as well as their hybrids [16]. Mohamed Kurdi [17] proposed an effective genetic algorithm with a critical-path-guided Giffler and Thompson crossover operator (GA-CPG-GT) for JSSP. Zhou et al. [18] presented a hybrid social-spider optimization algorithm with a differential mutation (SSO-DM) operator to solve JSSP. Cruz-Chávez et al. [19] proposed a parallel algorithm that generated a set of parallel working threads, where each thread performed a simulated annealing process to solve JSSP. For JSSP, Pongchairerks [20] proposed a new two-level meta-heuristic algorithm composed of an upper-level algorithm and a lower-level algorithm.

However, due to the premature and local convergence of GA, its performance in dealing with complex JSSP is limited [21]. The particle swarm optimization (PSO) algorithm cannot effectively search the entire solution space, and may converge to a local optimal solution prematurely, and thus cannot achieve a good exploration- exploitation balance [22]. The quality of the optimal solution obtained by TS lies on the initial solution [23]. Due to the lack of memory function in SA, it may lead to repeated searches and easily fall into local optima, greatly affecting the effectiveness of SA and causing excessive search time [23]. Due to the dependence

on random natural selection and recombination, the optimization results obtained by classical evolutionary algorithm are still limited [24]. Also due to the stubborn nature of JSSP, a single meta-heuristic method can no longer solve this problem well [18]. In addition, these heuristic approaches do not traverse all possible solutions, and can only find relatively good solutions through operations such as crossover, mutation and iteration. Even if a heuristic finds the optimal solution, the heuristic itself cannot prove that the solution it found is the actual optimal solution.

By contrast, DNA computing may be used to solve the JSSP. As long as appropriate encoding and manipulation are used, all possible solutions to the problem can be produced in one step. Deaton et al. [25] summarized three basic steps in using DNA computing to solve a problem: encoding, interaction and extraction. The first step is the basis of the other two steps, so that the key and the difficult part of DNA computing is to transform the problem into an equivalent DNA computing model by mapping.

Until now, there is not much reported research on solving JSSPs using DNA computing. Yin et al. [26] solved a FSSP using DNA computing by transforming the FSSP problem into a directed graph. Wang et al. [27] proposed a new parallel DNA algorithm to solve the task scheduling problem based on the Adleman-Lipton model, with an enlightening idea.

In this work, an appropriate encoding strategy is developed first to generate all possible solutions in parallel using DNA computing. The advantage of this encoding is that, once a scheduling sequence is determined, the makespan corresponding to each schedule is also determined. Theoretically efficient and parallel DNA algorithms based on Adleman-Lipton model are then proposed for solving the JSSP which can be solved with an $O(n^2)$ complexity. In the experiments, the DNA computing algorithms proposed in this work is simulated through two tool libraries of Python. Simulation experiments with 58 benchmark instances show that the proposed DNA algorithm outperforms other comparative heuristics.

The remainder of this paper is organized as follows. Section 2 describes the Adleman-Lipton model and describes the JSSP. Section 3 proposes a DNA algorithm for the JSSP and provides a performance analysis of the proposed DNA algorithm. Section 4 reports the experimental results of the proposed DNA algorithm and the comparison with several heuristic algorithms on 58 benchmark instances. Section 5 concludes this work with a summary and future research directions.

2. Preliminaries

This section is composed of two parts. The first part explains the Adleman-Lipton model, and the second part gives a formal description of the JSSP.

2.1 The Adleman-Lipton model

DNA is a polymer which is strung together from monomers called deoxyribonucleotides [28]. A single strand DNA molecule consists of a sequence of nucleotides with four different bases, i.e., adenine, guanine, cytosine and thymine, abbreviated as A, G, C and T, respectively. Every strand, according to its chemical structure, has a 5'-3' direction or a 3'-5' direction, with the 5'-end matching the 3'-end. In the double strand molecule, the two single strands have opposite directions. Using the Watson-Crick complementarity, i.e., the A-T pairing and the G-C pairing, without other possible pairings, a double strand DNA molecule can be formed under appropriate conditions. For instance, the single strand `5'-ACGTTA-3'` and its complement `3'-TGCAAT-5'` can form a double strand, also referred to as a duplex. Assume the upper strand runs from left to right in the 5'-3' direction, and consequently the lower strand runs from left to right in the 3'-5' direction. The complement `3'-TGCAAT-5'` of the strand `5'-`

ACGTTA-3' is denoted by $\overline{\text{ACGTTA}}$. The length of a single strand DNA molecule is the number of nucleotides in the molecule. Thus a single strand consisting of 12 nucleotides is said to be a 12 mer, i.e., a polymer consisting of 12 monomers.

The Adleman-Lipton model. A test tube is a set of molecules of DNA, i.e., a multi-set of finite strings over the alphabets {A, G, C, T}. The following operations can be performed:

(1) *Merge* (N_1,N_2,. . .,N_k): given k test tubes N_1,N_2,. . .,N_k, this operation pours the DNA solution in each of the test tubes N_2,. . .,N_k into test tube N_1. The uniform mixed solution is referred to as N_1.

(2) *Amplify*(N_1,N_2,. . .,N_k): given a test tube N_1, this operation creates copies of N_1 and amplifies them into test tubes N_2,. . .,N_k.

(3) *Separation* (N_1, X, N_2): given a test tube N_1 and a string X, this operation transfers all the single strands containing string X in test tube N_1 to test tube N_2. The single DNA strands removed from test tube N_1 are no longer contained in test tube N_1. If N_1 does not contain X, this operation does nothing.

(4) *Selection* (N_1, L, N_2): given a test tube N_1 and an integer L, this operation filters all DNA strands of length L in N_1 and put them into test tube N_2. Consequently, N_1 no longer contains these filtered DNA strands.

(5) *Append-head*(N,S): given a test tube N and a single strand S, this operation attaches (pastes) S to the front of every strand in N.

(6) *Append-tail* (N,R): given a test tube N and a single strand R, this operation attaches (pastes) R to the end of every strand in N.

(7) *Annealing* (N): given a test tube N with some single strands, this operation derives all possible double strands according to the *Watson-Crick complementarity* pairing principle, leaves them in N, and removes the other single strands from N.

(8) *Denaturation* (N): given a test tube N, this operation separates every double-stranded DNA molecule into two single strands by heating without breaking the phosphodiester bond of each single strand. Briefly, the double-stranded DNA in N is separated as follows

$$\begin{bmatrix} EF \\ \overline{EF} \end{bmatrix} \Rightarrow [EF], [\overline{EF}].$$

(9) *Cutting* (N, $\omega_1\omega_2$): given a test tube N and strings with $\omega_1\omega_2$, this operation divides every strand containing $[\omega_1\omega_2]$ in N to different strands as follows

$$[\cdots\alpha\omega_1\omega_2\beta\omega_1\omega_2\gamma\cdots] \Rightarrow [\cdots\alpha\omega_1], [\omega_2\beta\omega_1], [\omega_2\gamma\cdots],$$

where $\omega_1\omega_2$ corresponds to the recognition site of the cutting operation.

(10) *Discard* (N): given a test tube N, this operation clears all strands in N, that is, emptying N.

(11) *Read* (N): given a test tube N, this operation obtains the precise DNA sequences of all strands in N.

(12) *Sort* (N_1,N_2,N_3): given a test tube N_1 and two empty test tubes N_2 and N_3, this operation chooses the shortest strands in N_1 and puts them in N_2, chooses the longest strands in N_1 and puts them in N_3, and keeps the rest of the strands in N_1.

(13) *Ligation* (N): given a test tube N, this operation links all the DNA molecules (double strands) in N together by enzymes called ligases.

(14) *Detect* (N): given a test tube N, this operation returns "*true*" if N contains at least one DNA strand, and returns "*false*" otherwise.

(15) $T := B(N,\omega)$: given a test tube N and a string ω, this operation produces the test tube T consisting of all strands in N which begin with the string ω.

(16) $L := Length(N,\omega,\Omega)$: given a test tube N, this operation returns the length L of the specific single strand beginning with the string ω and ending with the string Ω in N.

In actual biological experiments, the above operations are feasible and achievable. Take the $Sort(N_1, N_2, N_3)$ operation as an example. In gel electrophoresis, the migration speed of DNA strands is related to its own length. The longer the strand, the slower the migration speed. Therefore, through gel electrophoresis experiments, the longest and shortest DNA strands in the test tube can be obtained. Since all operations mentioned above can be performed in lab within constant biological steps, it is reasonable to assume that the complexity of each operation is $O(1)$. In previous works ([10–12, 14, 15, 27]), many researchers have used this same approach to analyze the complexity of DNA computing algorithms. Therefore, the same approach is used in this study.

2.2 The job shop scheduling problem

The JSSP is already known as a typical NP-hard problem [29]. An $n \times m$ JSSP can be formally described as follows [30]. There are n jobs and m machines denoted as $J = (J_1, J_2, \cdots, Jn)$ and $M = (M_1, M_2, \cdots, M_m)$, respectively. Each job must be processed (or handled) through all m machines to fulfil its processing tasks. The processing of a job is also called an operation. Each job requires m operations. Only one machine is required for each operation, and only one operation can be handled on one of the m machines. Once started on a specified machine, an operation is not allowed to be interrupted until the processing of the job is completed, meaning that each operation begins only when all its previous operations are finished, i.e., preemption is not allowed. The processing time and the sequence of operations, i.e., the machining paths are given in advance. The goal of a JSSP is to find the optimal schedule in order to minimize the maximum makespan. A JSSP with n jobs and m machines has $(n!)^m$ possible solutions.

The notations used for the mathematical description of the JSSP are given below.

(1) n and m denote the numbers of jobs and machines, respectively.

(2) $O_{i,j}$ represents the operation i of job j, where $i \in [1, m]$ and $j \in [1, n]$.

(3) $t_{i,j}$ represents the processing time of $O_{i,j}$.

(4) $TJ_{i,j}$ represents the completion time of $O_{i,j}$, i.e., the cumulative completion time of job j up to operation i.

(5) $TM_{i,j}$ represents the earliest start time of $O_{i,j}$, i.e., the cumulative time (not including $t_{i,j}$) of machine i before job j starts.

The mathematical programming model of the JSSP is given in the following.

$$Min(\underset{1\le j\le n}{Max}(TJ_{m,j})) \tag{1}$$

Subject to

$$TJ_{i-1,j} + t_{i,j} \leq TJ_{i,j}, \text{ for } i \in [1, m] \text{ and } j \in [1, n] \quad (2)$$

$$TM_{i,j} + t_{i,j} \leq TJ_{i,j}, \text{ for } i \in [1, m] \text{ and } j \in [1, n] \quad (3)$$

$$TJ_{ij} \geq 0, \ TM_{ij} \geq 0, \text{ for } i \in [1, m] \text{ and } j \in [1, n]. \quad (4)$$

The objective function minimizes the makespan, i.e., the maximum completion time. Constraint (2) represents precedence relationship between the operations. Constraint (3) means that preemption is not allowed. Constraint (4) gives the domains of the variables.

Example 1. Table 1 shows a $n \times m = 4 \times 2$ FSSP example with 4 jobs J_1, J_2, J_3 and J_4 and 2 machines M_I M_1 and M_2. The jobs have the same operation sequence, i.e., the same machining path, where they first pass through machine 1 (M_1) and then pass through machine 2 (M_2). The time needed by each job on each machine is shown in the table.

Example 2. Table 2 shows a $n \times m = 3 \times 3$ JSSP example with 3 jobs J_1, J_2 and J_3, each with a different machining path, processed on 3 machines, M_1, M_2 and M_3. The machines required are shown in the column M_i and the time needed by each job on each machine is shown in the column $t_{i,j}$. For instance, the 1st operation of job 1 (J_1), i.e., $O_{1,1}$, is processed on machine 3 (M_3) and the processing time corresponding to $O_{1,1}$ is $t_{1,1} = 7$ units. The 2nd operation of job 1 (J_1), i.e., $O_{2,1}$, is processed on machine 1 (M_1) and the processing time corresponding to this operation is $t_{2,1} = 4$ units, and so on.

From the two examples above, it is intuitive that the JSSP is an extension of the FSSP. The biggest difference between a JSSP and a FSSP lies in the machining paths of the jobs, as shown in Tables 1 and 2. As compared with the FSSP as shown in Table 1, the machining paths of the jobs are different from each other in a JSSP as shown in Table 2. If all the jobs have the same machining path, i.e., each job needs the same operations, without distinction between the operations and the machines, the JSSP becomes a FSSP.

As compared with FSSP, JSSP is much more complicated and closer to the practical problems in production. In a FSSP, only one time matrix is needed. However, in a JSSP, two matrices are required, one represents the processing time and the other represents the machines needed by the jobs. Therefore, this work focuses on the more practical and representative JSSP for an in-depth study.

3. A DNA algorithm for the job shop scheduling problem

Encoding is the key and difficult part of solving the combinatorial optimization problem with DNA computing. Therefore, this section starts with the coding scheme and then gives an overview of the proposed algorithm. The detailed algorithm is finally presented.

Table 1. A $n \times m = 4 \times 2$ FSSP.

Machine	Job			
	J_1	J_2	J_3	J_4
M_1	15	8	6	12
M_2	4	10	5	7

https://doi.org/10.1371/journal.pone.0242083.t001

Table 2. A $n \times m = 3 \times 3$ JSSP.

i	J_1		J_2		J_3	
	M_i	$t_{i,1}$	M_i	$t_{i,2}$	M_i	$t_{i,3}$
1	3	7	2	5	2	4
2	1	4	3	6	1	2
3	2	2	1	3	3	3

https://doi.org/10.1371/journal.pone.0242083.t002

3.1 Encoding

A schedule, or scheduling sequence, of an $n \times m$ JSSP can be denoted by $OP_1{-}OP_2{-}\cdots OP_{n\times m}$, where $OP_i \in [1, n]$ indicates a job number. In this schedule, the i^{th} appearance of job j indicates operation i of job j, i.e., $O_{i,j}$. Each number OP_i appears exactly m times.

Take a scheduling sequence 1-3-2-2-1-3-3-1-2 of a 3 × 3 JSSP as an example. The first number '1' indicates operation 1 of job 1; the second number '3' indicated operation 1 of job 3. The fourth number '2' (the 2nd appearance of job 2) indicates operation 2 of job 2. Similarly, the seventh number '3' (the 3rd appearance of job 3) indicates operation 3 of job 3. Obviously, once a scheduling sequence is determined, the makespan corresponding to this schedule is uniquely determined. For example, referring to the data in Table 2 in Example 2, the completion times or makespans of the 3 jobs can be easily calculated. The completion times of jobs 1, 2 and 3 are 13, 18 and 18, respectively. Consequently, the makespan (completion time) for this schedule is 18. Fig 1 in the following shows this schedule as a Gantt chart.

Effective encoding needs to be used to reasonably map real problems to DNA molecular computing models, and to generate all possible solutions in parallel in one step. In the following, p, E_i, q, F_j are used to represent different DNA single strands with the same length, e.g., u mer, with u as a positive integer. The notations p and q are used for DNA ligation, as defined in Section 2.2, and E_i and F_j represent the single strands for operation i and job j, respectively. The single strand pE_iqF_j can be used to indicate operation i of job j, i.e., $O_{i,j}$. In this way, all $(n!)^m$ possible solutions can be easily generated.

Example 3. Table 3 shows a $n \times m = 5 \times 6$ JSSP example with 5 jobs, each with a different machining path, processed on 6 machines. Data for each job are shown in two columns in the table. The first column shows the corresponding machine number and the second column shows the time needed by the job on each machine.

Fig 2 in the following is an optimal scheduling sequence of this 5×6 JSSP. It means that the jobs are processed in the order of 5-4-2-1-3 for the 1st operation, in the order of 5-2-1-4-3 for the 2nd operation, and so on, where the number '0' in the middle represents a separator. The same job processed by different machines are identified with the same color. A DNA encoding method based on scheduling sequences is proposed below.

In accordance with *Algorithm* 1, the DNA strands $\{pE_1qF_5pE_1qF_4pE_1q\text{-}\ F_2pE_1qF_1pE_1qF_3\}$ will be generated to denote that the jobs are processed in the order of 5-4-2-1-3 in the 1st

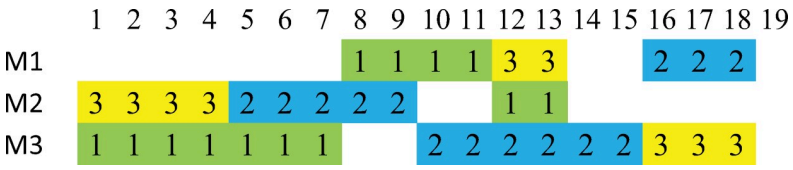

Fig 1. A Gantt chart for the schedule 1-3-2-2-1-3-3-1-2 of Example 2.

https://doi.org/10.1371/journal.pone.0242083.g001

Table 3. A $n \times m = 5 \times 6$ JSSP.

i	J_1		J_2		J_3		J_4		J_5	
	M_i	$t_{i,1}$	M_i	$t_{i,2}$	M_i	$t_{i,3}$	M_i	$t_{i,4}$	M_i	$t_{i,5}$
1	3	3	2	6	3	1	4	7	5	6
2	1	10	3	8	4	5	1	4	2	10
3	2	9	5	1	6	5	3	4	3	7
4	4	5	6	5	1	5	2	3	6	8
5	6	3	1	3	2	9	5	1	1	5
6	5	10	4	3	5	1	6	3	4	4

https://doi.org/10.1371/journal.pone.0242083.t003

operation. In this way, DNA strands can be obtained to denote all n jobs in every operation. By encoding the jobs in this manner, all $(n!)^m$ possible schedules will be obtained.

Fig 3 in the following is the optimal schedule presented as a Gantt chart corresponding to the optimal scheduling sequence given above in Fig 2, where the maximum completion time, i.e., the makespan, of this schedule is 45. The result 45 is calculated using the data in Table 3.

In JSSP, different jobs may require the same machine in some operations, so that some jobs may have to wait for others to finish before being processed and machines may become idle while having to wait for jobs to come. Different schedules have different job and machine waiting times and might have different makespans. The advantage of this encoding is that, once a scheduling sequence, i.e., a schedule, is determined, the makespan corresponding to this schedule is also uniquely determined. However, it should be noted that the makespan is, but the corresponding scheduling sequences may not be, unique.

3.2 An outline of the algorithm

The basic idea of this DNA algorithm for solving the JSSP is to find an optimal solution by checking all possible solution candidates. This brute force approach is realized through DNA computing. Specifically, this proposed algorithm consists of four steps.

(1) Generate the initial solution space in test tube N_0 for the JSSP;

(2) Screen the DNA strands representing the feasible schedules and discard the ones representing infeasible schedules;

(3) Append time information strands at the end of the strands representing feasible schedules and calculate the completion time of each feasible schedule;

(4) Select the strands corresponding to the optimal schedule that minimizes the maximum completion time, i.e., the makespan.

The flowchart of the algorithm is shown in Fig 4 as follows.

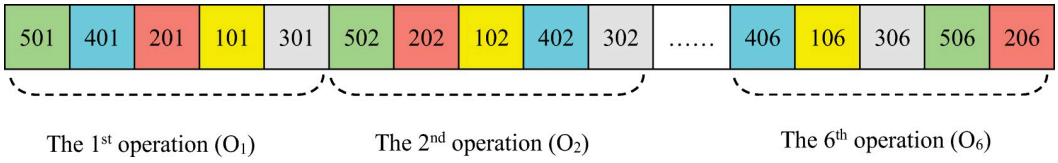

Fig 2. An optimal schedule of the 5×6 JSSP in Example 3.

https://doi.org/10.1371/journal.pone.0242083.g002

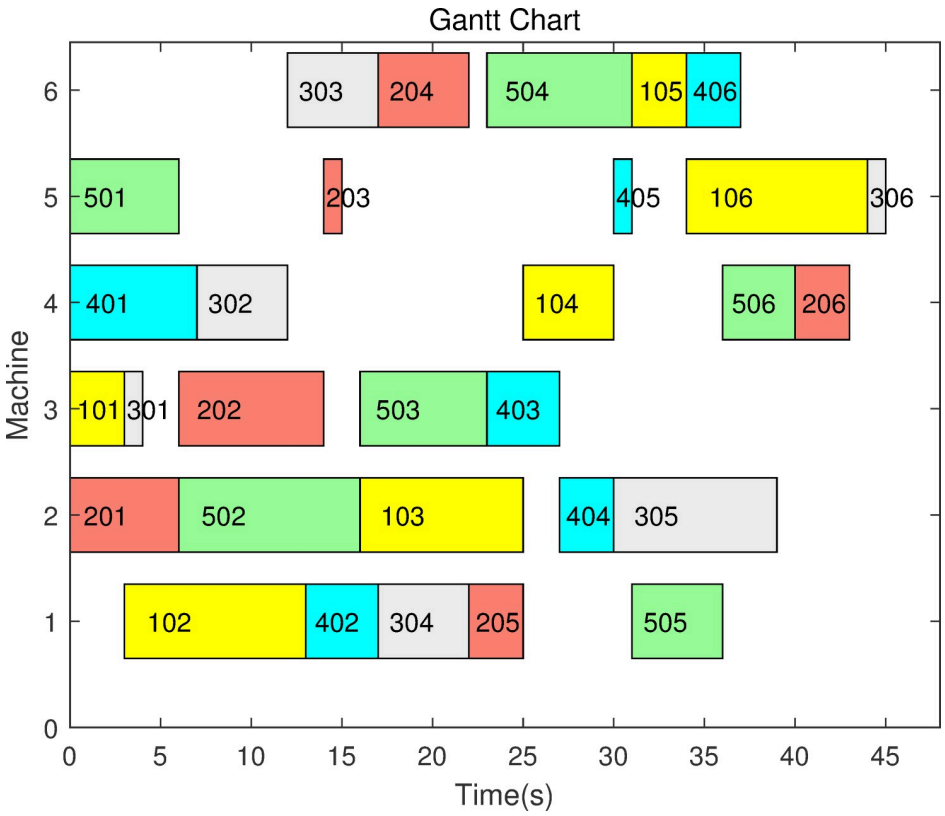

Fig 3. Gantt chart of an optimal schedule of the 5×6 JSSP in Example 3.

https://doi.org/10.1371/journal.pone.0242083.g003

3.3 Detailed algorithm

3.3.1 Initialization of the sequence of the *n* jobs for each operation. Initial test tube:

$$N_i = \{pE_iq\} \text{ for } 1 \le i \le m$$

$$Q = \{F_1, F_2, \cdots, F_n, \overline{qF_1}, \overline{qF_2}, \cdots, \overline{qF_n}, \overline{qF_1p}, \overline{qF_2p}, \cdots, \overline{qF_np}\}$$

Algorithm 1. Initialization of the sequence of the *n* jobs for each operation

```
Begin
1:  for i = 1 to m do
2:     Merge(N_i, Q);
3:    Annealing(N_i);
4:    Ligation(N_i);
5:    Denaturation(N_i);
6:    Selection(N_i, 4nu, V_i);
7:    Discard(N_i);
8:    V_i: = B(T_i, pE_iq);
9:    for j = 1 to j = n do
10:      Separation(V_i, qF_jp, U_i);
11:      Discard(V_i);
12:      Amplify(U_i, V_i);
13:      Discard(U_i);
14:    end for
15:    Amplify(V_i, N_i);
16:    Discard(V_i);
```

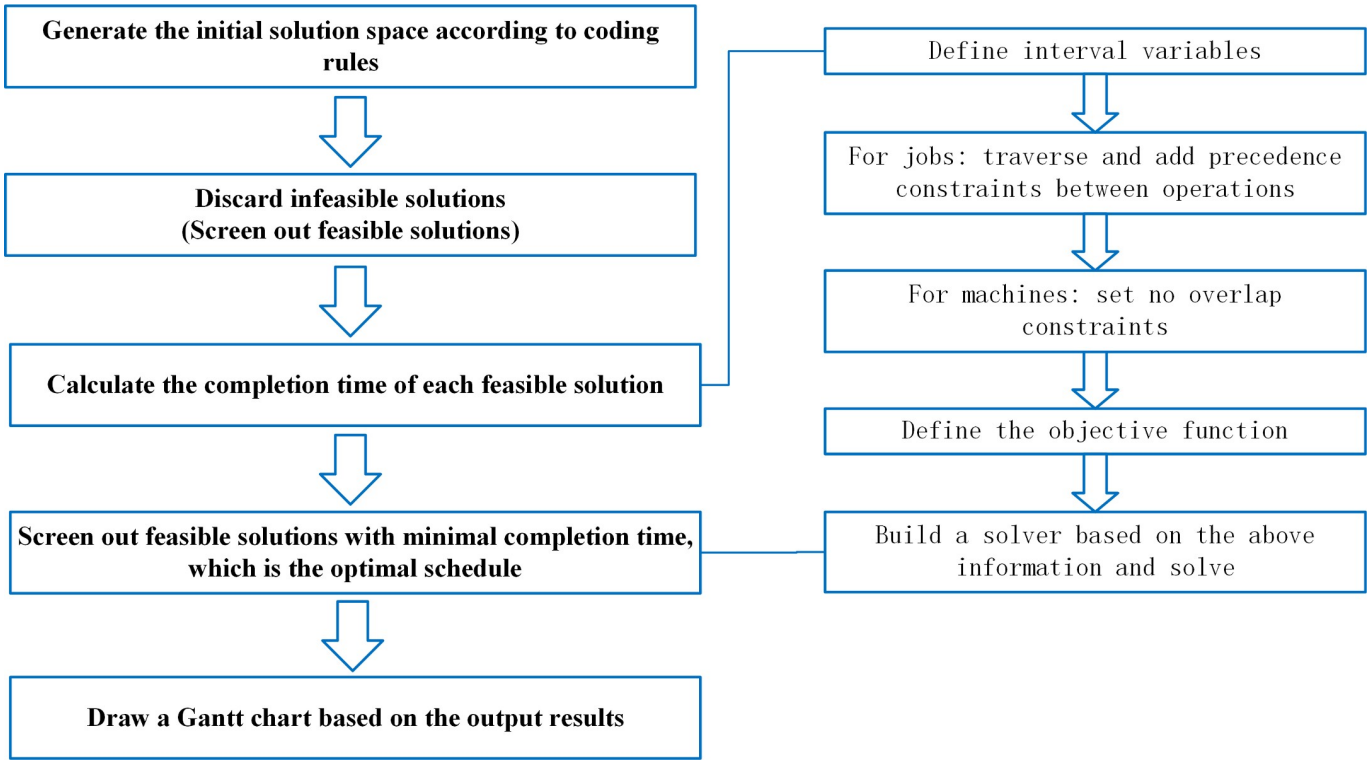

Fig 4. Algorithm flow chart.

https://doi.org/10.1371/journal.pone.0242083.g004

```
17:    Append-tail(N_i, a_i2S);
18:    Append-head(N_i, a_i1);
19:  end for
End
```

Algorithm 1 produces DNA strands representing all n jobs for every possible operation, for instance,

$$\{a_{11}pE_1qF_2pE_1qF_4pE_1qF_5pE_1qF_1pE_1qF_3a_{12}S\}$$
$$\text{and } \{a_{21}pE_2qF_2pE_2qF_1pE_2qF_5pE_2qF_4pE_2qF_3a_{22}S\},$$

and so on. The lengths of the single strands $a_{i,j}$ and S are also u mer. The strands $a_{i,j}$ and S are used for connection in the following algorithm.

3.3.2 Generation of all possible strands for the JSSP. Initial test tube N_0:

$$N_0 = \{\overline{a_{12}Sa_{21}}, \overline{a_{22}Sa_{31}}, \overline{a_{32}Sa_{41}}, \cdots, \overline{a_{m-1,2}Sa_{m1}}\}$$

Algorithm 2. Generation of all possible strands for the JSSP

```
Begin
1:  Merge (N_0, N_1, N_2,..., N_m);
2:  Annealing (N_0);
3:  Denaturation (N_0);
4:  N_0: = B (N_0, a_11pE_1q);
5:  Selection (N, (4n+3)mu, N_0);
6:  for i = 1 to i = m do
7:    Separation (N_0, a_i2S, N_1);
8:    Discard (N_0);
```

```
9:    Amplify (N1, N0);
10:    Discard (N1);
11:  end for
End
```

After executing Algorithm 2, all possible DNA strands representing all possible solutions of the JSSP can be obtained as shown below

$$\{a_{11}pE_1qF_{j_1}\cdots pE_1qF_{j_k}\cdots pE_1qF_{j_n}a_{12}Sa_{21}pE_2qF_{j_1}\cdots pE_2qF_{j_k}\cdots pE_2qF_{j_n}a_{22}S\cdots\},$$

where the subscript j_k of F_{j_k} is uniquely determined by the value of k for $1 \leq k \leq n$, and $F_{j_k} \in \{F_1, F_2, \cdots, F_n\}$, i.e., the sequence $j_1, \ldots, j_k, \ldots, j_n$ is an arbitrary out-of-order combination of the sequence 1,. . .,n.

3.3.3 Computation of the final completion time of each job for every strand. In Algorithm 3, as explained in Section 2.2, TJ_{i,j_k} denotes the completion time of job j_k in operation i, TM_{i,j_k} is the cumulative time (not including t_{i,j_k}) of the required machine corresponding to job j_k in operation i, and t_{i,j_k} is the corresponding processing time of job j_k in operation i. The value of j_k is also uniquely determined by the value of k for $1 \leq k \leq n$, where $j_k \in \{1,2,\cdots,n\}$. The final completion time of the n jobs are stored separately in n test tubes. The single strand Ψ, also with a length of u mer, in Algorithm 3 denotes one unit of time.

Algorithm 3. Computation of the final completion time of each job for every strand

```
Begin
1:  Amplify (N0,N1,N2, ...,Nn);
2:  Discard (N0);
3:  for i = 1 to m do
4:    for k = 1 to n do
5:      if i>1 then
6:        Separation (N_jk,Sω, U_jk);
7:        Discard (N_jk);
8:        Cutting (U_jk,Sω);
9:        N_jk: = B (U_jk,a11pE1q);
10:        Discard (U_jk);
11:      else
12:        Continue
13:      if TJ_{i-1,jk} > TM_{i,jk} (when i = 1, both initial values are 0) then
14:        Append-tail (N_jk, ω ΨΨ···ΨΨ Ω);
                                  TJ_{i-1,jk}+t_{i,jk}
15:      else
16:        Append-tail (N_jk, ω ΨΨ···ΨΨ Ω);
                                  TM_{i,jk}+t_{i,jk}
17:      TJ_{i,jk}: = Length(N_jk, ω, Ω);
18:      TM_{i,jk}: = Length(N_jk, ω, Ω);
19:    end for
20:  end for
End
```

3.3.4 DNA optimization. This algorithm finds the optimal schedule that minimizes the maximum completion time, i.e., the makespan,

$$Min(\underset{1\leq j\leq n}{Max}(TJ_{m,j})),$$

where $TJ_{m,j}$ is the final completion time of job j in the last operation.

Algorithm 4. DNA optimization

```
Begin
1:  for i = 1 to n do
2:    Sort (N_i, V_1, V_2);
3:  end for
4:  Sort (V_2, V_0, V_3);
5:  Cutting (V_0, Sω);
6:  T_0: = B (V_0, a_11pE_1q);
7:  Read (V_0)
End
```

Theorem in the following is obtained by inspecting Algorithms 1–4 line by line.

Theorem. *Without loss of generality, $n \geq m$ is assumed. The solutions of a n×m JSSP has an $O(n^2)$ complexity using DNA computing.*

Proof. The total complexity of the four algorithms is as follows

$$O(Algorithm\ 1) = O(m(7+4n+4)) \approx O(4n^2+11n) = O(n^2);$$

$$O(Algorithm\ 2) = O(5+4m) \approx O(4n+5) = O(n);$$

$$O(Algorithm\ 3) = O(10mn+2) \approx O(10n^2+2) = O(10n^2) = O(n^2);$$

$$O(Algorithm\ 4) = O(n+4) = O(n);$$

$$\begin{aligned} O &= O(Algorithm\ 1) + O(Algorithm\ 2) + O(Algorithm\ 3) + O(Algorithm\ 4) \\ &= O(n^2) + O(n) + O(n^2) + O(n) = O(n^2) \end{aligned}$$

In conclusion, the optimal schedule of a JSSP can be found with an $O(n^2)$ complexity.

Summary. The solution of the JSSP can be represented by a strand with a polynomial length.

Explanation. Suppose the length of the different strands is

$$||E_i|| = ||F_j|| = ||a_{ij}|| = ||S|| = ||\Psi|| = ||\Omega|| = ||\omega|| = ||p|| = ||q|| = u \text{ mer, for } i \in [1,\ m] \text{ and } j \in [1,\ n].$$

Let $l = \Sigma\Sigma t_{ij}$, and also assume $m \leq n$. The length of DNA strand L corresponding to the optimal schedule in Algorithm 4 is as follows.

$$||L|| = n\sum_{i=1}^{m}||E_i|| + m\sum_{j=1}^{n}||F_j|| + mn(||p|| + ||q||) + ||\Omega|| + ||\omega||$$

$$+\underbrace{||\Psi|| + \cdots + ||\Psi||}_{Min(Max_{1\leq j\leq n}(TJ_{m,j}))<<l} + \sum_{i=1}^{m}(||a_{i1}|| + ||a_{i2}||) + m||S||$$

$$< mnu + mnu + mn2u + lu + 2mu + mu$$

$$< n^2u + n^2u + 2n^2u + lu + 2nu + nu$$

$$= (4n^2 + 3n + l)u$$

The final solution strand in Algorithm 4 is within appropriate length. The optimal solution can then be found and determined.

4. Experiment and comparison

The algorithm proposed in this study is simulated in Python. Two important tool libraries, i.e., Biopython and DOcplex, are used to simulate and implement the four algorithms, as components of the proposed algorithm, in this work. Biopython, a Python tool for computational molecular biology, is used to encode problems and construct solution spaces. DOcplex, a Python tool library for solving constraint programming problems, is used to simulate the constraints in Algorithm 3 and the objective function in Algorithm 4. The computer used for computation has an i5-4210H processor with a 2.90GHz clock speed and 12G of RAM.

The algorithm is first compared with four state-of-the-art heuristics on 43 JSSP benchmark instances (see Table 4). The results show that, except for instance LA29, the proposed algorithm found the best known solutions for the remaining 42 instances, and has the same or better performance than the four comparative heuristics.

The 43 JSSP benchmark instances are selected from the OR Library [31], which contains 3 instances (FT06, FT10, FT20) designed by Fisher and Thompson [32] and 40 instances (LA01~LA40) designed by Lawrence [33]. The four comparative heuristics used for comparison are MAGATS [21], NIMGA [34], aLSGA [35] and WW [36].

Table 4 shows the results obtained by the proposed algorithm and the four comparative heuristics for the 43 instances. These results include the names of the instances, the sizes of the instances represented by $n\times m$, the best known solutions (BKS) and the best solutions obtained by the proposed algorithm and the four comparative heuristics. Results of the comparative heuristics are from the original respective publications [21, 34–36].

In order to visualize the scheduling results of the proposed algorithm, the Gantt charts of the optimal schedules of the instances FT20, LA20 and LA36 are presented in Figs 5–7, respectively. Figs 5–7 show that the optimal makepans of instances FT20, LA20 and LA36 are 1165, 902 and 1268 units of time, respectively. Not all four comparative heuristics could find the best known solution for instance FT20 and none of these heuristics could find the best known solutions for instances LA20 and LA36.

Table 5 shows a comparative analysis with four more heuristics on four instances of different sizes. The four heuristics are PSO [37], IGA [38], DE [39] and SSO-DM [18]. The table gives the statistical results, i.e., the best, worst, mean and standard deviation (Std.), of 20 independent runs. Except for instance YN4, the proposed algorithm found the best known

Table 4. Results obtained by the proposed algorithm and four comparative heuristics for the 43 instances.

Instances	Size	BKS	Proposed	MAGATS	NIMGA	aLSGA	WW
FT06	6×6	55	**55**	55	55	55	55
FT10	10×10	930	**930**	930	930	930	930
FT20	20×5	1165	**1165**	1165	**1173**	1165	1165
LA01	10×5	666	**666**	666	666	666	666
LA02	10×5	655	**655**	655	655	655	655
LA03	10×5	597	**597**	597	597	**606**	597
LA04	10×5	590	**590**	590	590	**593**	590
LA05	10×5	593	**593**	593	593	593	593
LA06	15×5	926	**926**	926	926	926	926
LA07	15×5	890	**890**	890	890	890	890
LA08	15×5	863	**863**	863	863	863	863
LA09	15×5	951	**951**	951	951	951	951
LA10	15×5	958	**958**	958	958	958	958
LA11	20×5	1222	**1222**	1222	1222	1222	1222
LA12	20×5	1039	**1039**	1039	1039	1039	1039
LA13	20×5	1150	**1150**	1150	1150	1150	1150
LA14	20×5	1292	**1292**	1292	1292	1292	1292
LA15	20×5	1207	**1207**	1207	1207	1207	1207
LA16	10×10	945	**945**	945	945	**946**	945
LA17	10×10	784	**784**	784	784	784	784
LA18	10×10	848	**848**	848	848	848	848
LA19	10×10	842	**842**	842	842	**852**	842
LA20	10×10	902	**902**	**907**	**907**	**907**	**907**
LA21	15×10	1046	**1046**	1046	**1058**	**1068**	**1046**
LA22	15×10	927	**927**	927	**937**	**956**	**935**
LA23	15×10	1032	**1032**	1032	1032	1032	1032
LA24	15×10	935	**935**	935	**947**	**966**	**937**
LA25	15×10	977	**977**	977	**989**	**1002**	977
LA26	20×10	1218	**1218**	1218	1218	**1223**	1218
LA27	20×10	1235	**1235**	1235	**1269**	**1281**	**1236**
LA28	20×10	1216	**1216**	1216	**1247**	**1245**	1216
LA29	20×10	1152	**1176**	**1164**	**1241**	**1230**	**1160**
LA30	20×10	1355	**1355**	1355	1355	1355	1355
LA31	30×10	1784	**1784**	1784	1784	1784	1784
LA32	30×10	1850	**1850**	1850	1850	1850	1850
LA33	30×10	1719	**1719**	1719	1719	1719	1719
LA34	30×10	1721	**1721**	1721	1721	1721	1721
LA35	30×10	1888	**1888**	1888	1888	1888	1888
LA36	15×15	1268	**1268**	**1281**	**1293**	–	**1279**
LA37	15×15	1397	**1397**	1397	**1432**	–	**1407**
LA38	15×15	1196	**1196**	**1198**	**1222**	–	1196
LA39	15×15	1233	**1233**	1233	**1251**	–	**1242**
LA40	15×15	1222	**1222**	**1228**	**1246**	–	**1229**

https://doi.org/10.1371/journal.pone.0242083.t004

solutions for the remaining three instances. The results of the comparative heuristics are from Zhou et al. [18]. Figs 8–11 show the box plots of these five algorithms on the four instances.

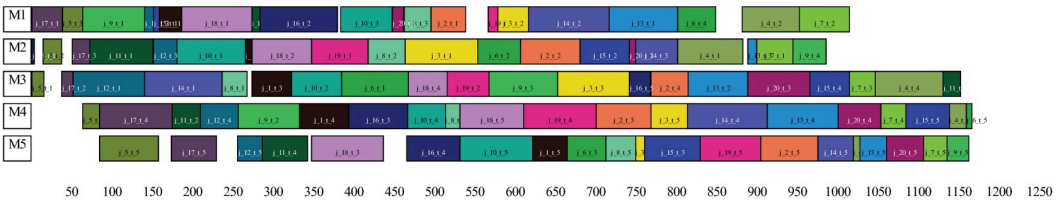

Fig 5. Gantt chart of an optimal schedule of instance FT20.

https://doi.org/10.1371/journal.pone.0242083.g005

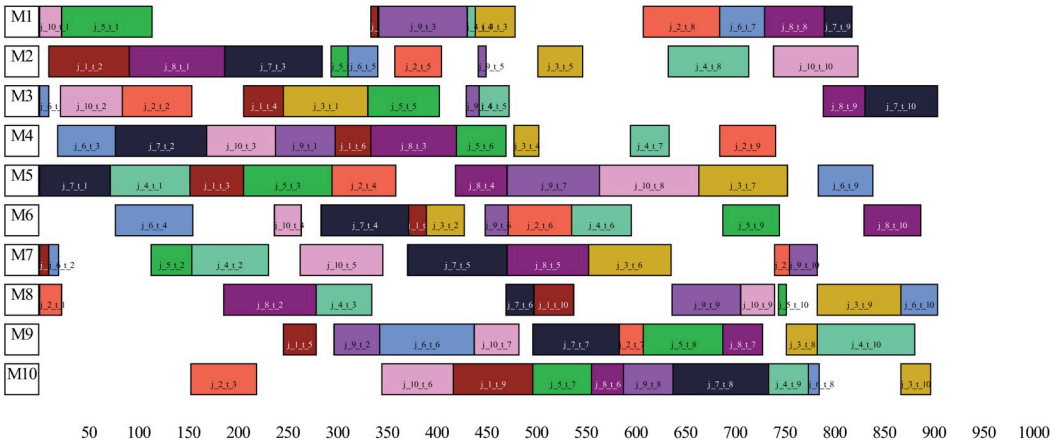

Fig 6. Gantt chart of an optimal schedule of instance LA20.

https://doi.org/10.1371/journal.pone.0242083.g006

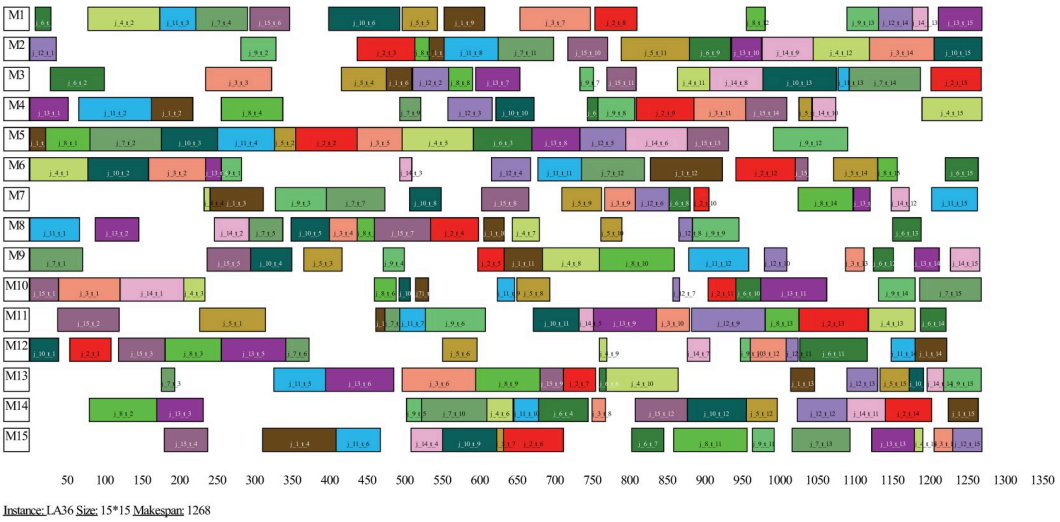

Fig 7. Gantt chart of an optimal schedule of instance LA36.

https://doi.org/10.1371/journal.pone.0242083.g007

Table 5. Statistical results of five algorithms on four instances.

Instances	Size	BKS	Algorithm	Best	Worst	Mean	Std.
FT20	20×5	1165	PSO	1374.00	1521.00	1442.50	42.02
			IGA	1744.00	2527.00	2025.50	198.95
			DE	1456.00	1554.00	1506.00	27.64
			SSO-DM	1374.00	1374.00	1374.00	0
			Proposed	1165.00	1165.00	1165.00	0
LA40	15×15	1222	PSO	1498.00	1732.00	1576.05	59.79
			IGA	2154.00	2803.00	2340.25	155.90
			DE	1691.00	1824.00	1767.05	36.46
			SSO-DM	1528.00	1528.00	1528.00	0
			Proposed	1222.00	1222.00	1222.00	0
ORB10	10×10	944	PSO	1039.00	1263.00	1150.05	48.84
			IGA	1431.00	2121.00	1761.25	158.12
			DE	1190.00	1293.00	1244.40	25.04
			SSO-DM	1114.00	1114.00	1114.00	0
			Proposed	944.00	944.00	944.00	0
YN4	20×20	968	PSO	1340.00	1607.00	1425.15	64.84
			IGA	1826.00	2192.00	1997.90	116.48
			DE	1486.00	1601.00	1570.75	26.15
			SSO-DM	1492.00	1492.00	1492.00	0
			Proposed	979.00	996.00	987.43	6.04

https://doi.org/10.1371/journal.pone.0242083.t005

Table 6 shows the experimental results of the proposed algorithm on the instances of Yamada and Nakano [40] (YN1~YN4) and Storer et al. [41] (SWV01~SWV10). In the experiment, the running time limit of the algorithm is set to 2000 seconds (Sec.), and the relative

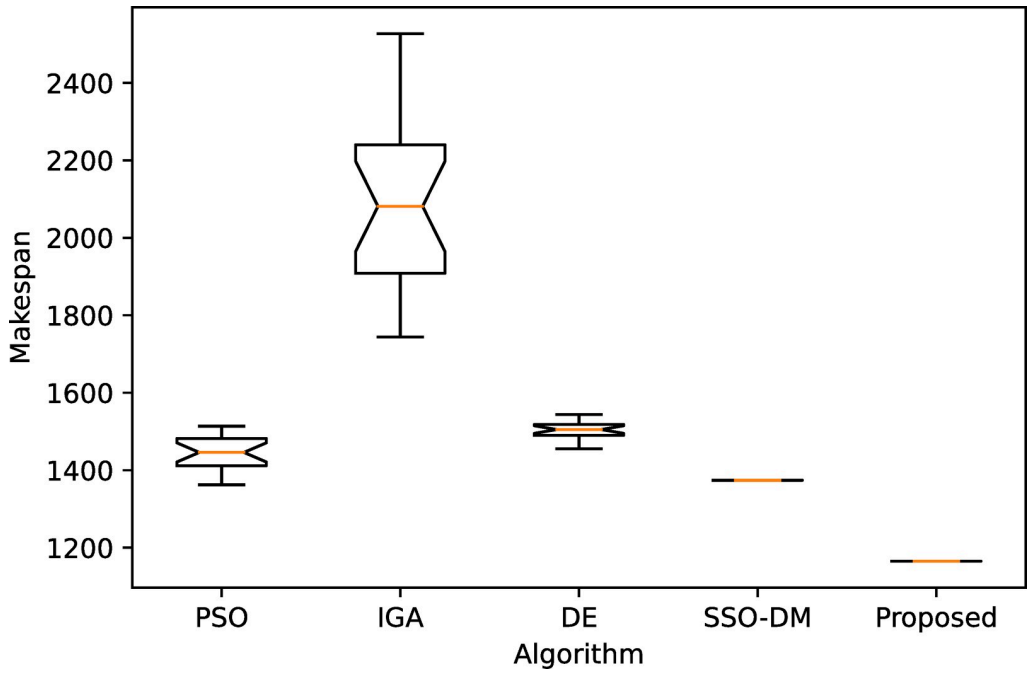

Fig 8. The box plot for FT20.

https://doi.org/10.1371/journal.pone.0242083.g008

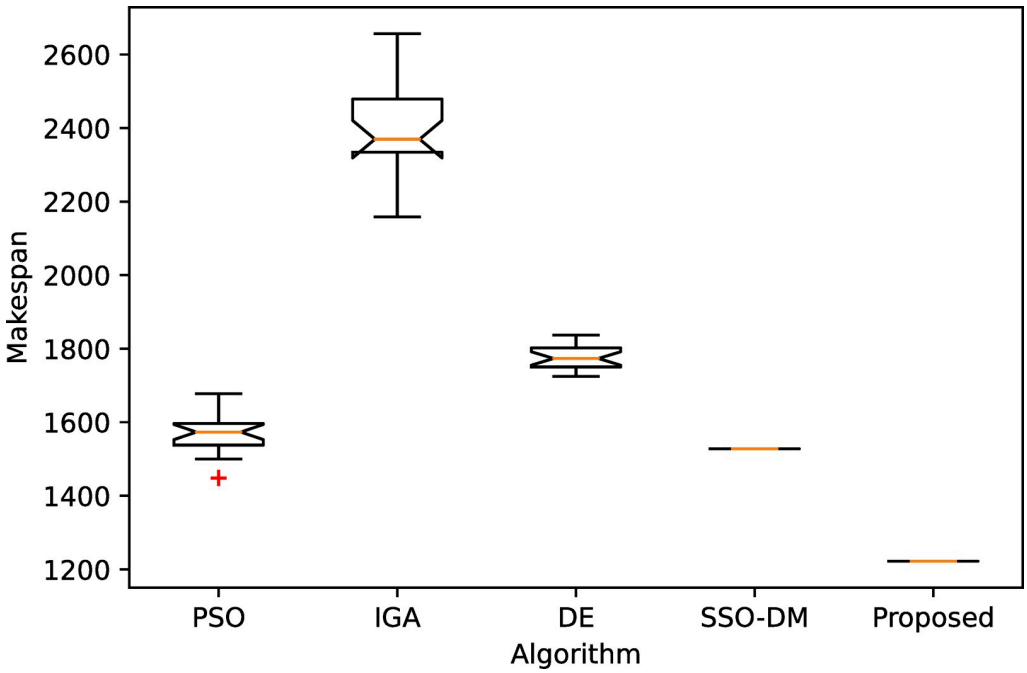

Fig 9. The box plot for LA40.

https://doi.org/10.1371/journal.pone.0242083.g009

error (RE), i.e., the error between the obtained and the best know solutions defined as a percentage of the best known solution, is introduced as a criterion. The third column BKS/UB in the table represents the best known solutions (BKS) or the known upper bounds (UB) when the BKS is unknown. The last column t shows the running time taken by the algorithm in

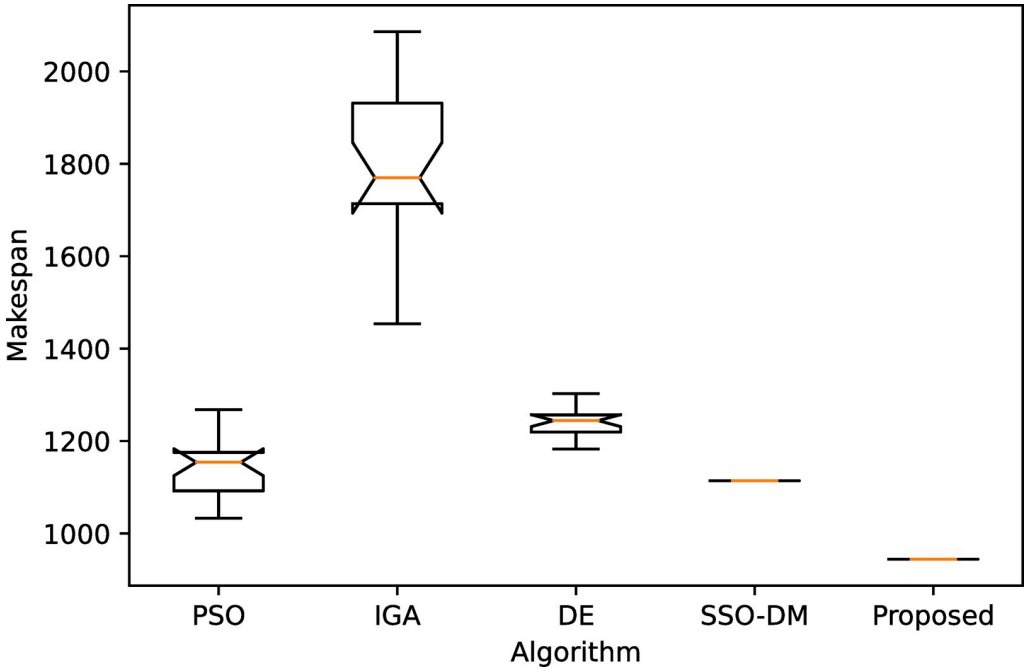

Fig 10. The box plot for ORB10.

https://doi.org/10.1371/journal.pone.0242083.g010

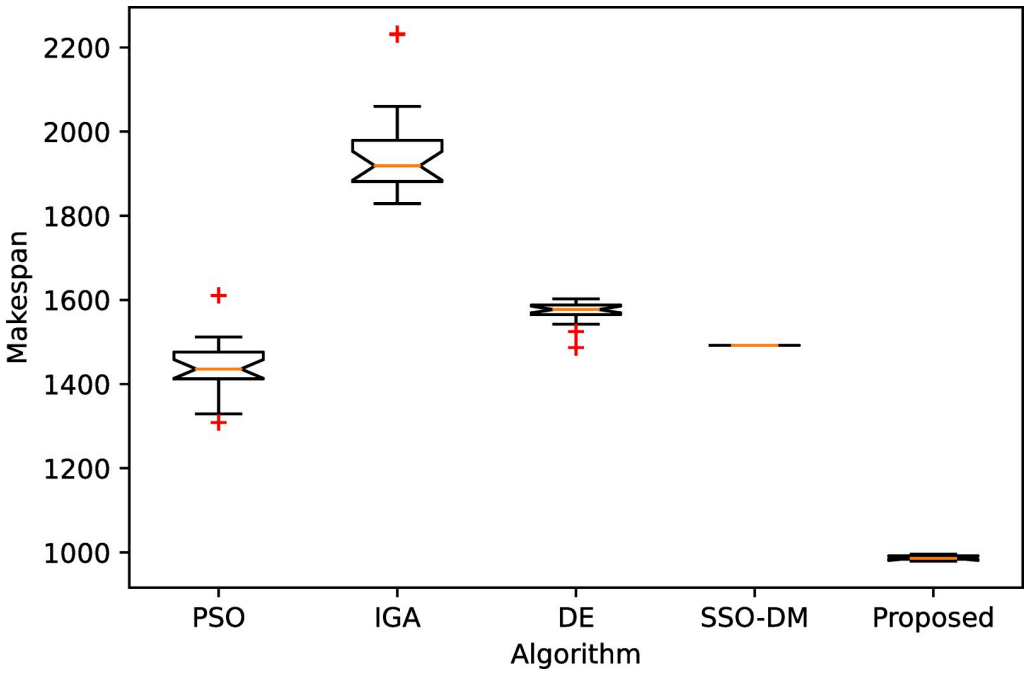

Fig 11. The box plot for YN4.

https://doi.org/10.1371/journal.pone.0242083.g011

seconds. The results show that the proposed algorithm can find the best known solutions for the three instances SWV01~SWV03 in a short time, but cannot find the best known solutions for the remaining 11 instances within the running time limit of 2000 seconds. The maximum RE value for these instances do not exceed 6%.

5. Conclusions

Based on the DNA operations of the Adleman-Lipton model, an appropriate encoding strategy is developed first to generate all possible solutions in parallel using DNA computing in one

Table 6. Simulation test on examples YN01~YN04 and SWV01~SWV10.

Instances	Size	BKS/UB	Best	RE(%)	t (Sec.)
SWV01	20×10	1407	1407	0	1021.06
SWV02	20×10	1475	1475	0	468.86
SWV03	20×10	1398	1398	0	612.78
SWV04	20×10	1474	1505	2.10	2000.00
SWV05	20×10	1424	1506	5.75	2000.00
SWV06	20×15	1678	1746	4.05	2000.00
SWV07	20×15	1600	1630	1.86	2000.00
SWV08	20×15	1763	1798	1.99	2000.00
SWV09	20×15	1661	1724	3.79	2000.00
SWV10	20×15	1767	1795	1.58	2000.00
YN1	20×20	885	896	1.24	2000.00
YN2	20×20	909	912	0.30	2000.00
YN3	20×20	892	905	1.46	2000.00
YN4	20×20	968	979	1.14	2000.00

https://doi.org/10.1371/journal.pone.0242083.t006

step. Then four highly efficient and parallel DNA algorithms as components of the proposed algorithm are proposed for JSSP. The proposed algorithm is simulated and compared with several heuristics using 58 JSSP benchmark instances from the literature, and the proposed algorithm found the best known solutions for 46 instances. The results show that the proposed algorithm performs better than the comparative heuristics.

One direction of future works is to explore the possibility of solving the FJSP by using viable biological computational models including the sticker model among others. In addition, for larger-scale benchmarks such as the SWVs and the YNs, multi-threaded computing will be considered for simulation implementation in the future.

Supporting information

S1 File. Code, 58 benchmark instances and their data descriptions, solution results and operations guides (Readme.doc file) related to the python source program.
(ZIP)

Author Contributions

Conceptualization: Xiang Tian, Xiyu Liu.

Formal analysis: Hongyan Zhang.

Funding acquisition: Xiyu Liu.

Investigation: Xiang Tian.

Methodology: Xiang Tian, Xiyu Liu, Hongyan Zhang.

Software: Xiang Tian.

Supervision: Xiyu Liu, Minghe Sun.

Validation: Hongyan Zhang, Minghe Sun, Yuzhen Zhao.

Visualization: Xiang Tian.

Writing – original draft: Xiang Tian.

Writing – review & editing: Xiang Tian, Minghe Sun, Yuzhen Zhao.

References

1. Feynman R, Gilbert D. Miniaturization. Reinhold, New York. 1961: 282–96.
2. Adleman LM. Molecular computation of solutions to combinatorial problems. Science. 1994; 266: 1021–4. https://doi.org/10.1126/science.7973651 PMID: 7973651
3. Lipton RJ. DNA solution of HARD computational problems. Science.1995; 268: 542–5. https://doi.org/10.1126/science.7725098 PMID: 7725098
4. Ouyang Q, Kaplan PD, Liu S, Libchaber A. DNA solution of the maximal clique problem. Science. 1997; 278(5337): 446–9. https://doi.org/10.1126/science.278.5337.446 PMID: 9334300
5. Roweis S, Winfree E, Burgoyne R, Chelyapov NV, Goodman MF, Rothemund PW, et al. A sticker-based model for DNA computation. Journal of Computational Biology. 1998; 5(4): 615–29. https://doi.org/10.1089/cmb.1998.5.615 PMID: 10072080
6. Winfree E, Liu F, Wenzler LA, Seeman NC. Design and self-assembly of two-dimensional DNA crystals. Nature. 1998; 394(6693): 539–44. https://doi.org/10.1038/28998 PMID: 9707114
7. Sakamoto K, Gouzu H, Komiya K, Kiga D, Yokoyama S, Yokomori T, et al. Molecular computation by DNA hairpin formation. Science. 2000; 288(5469): 1223–6. https://doi.org/10.1126/science.288.5469.1223 PMID: 10817993
8. Liu Q, Wang L, Frutos AG, Condon AE, Corn RM, Smith LM. DNA computing on surfaces. Nature. 2000; 403(6766): 175–9. https://doi.org/10.1038/35003155 PMID: 10646598

9. Smith LM, Corn RM, Condon AE, Lagally MG, Frutos AG, Liu Q, et al. A surface-based approach to DNA computation. Journal of computational biology. 1998; 5(2): 255–67. https://doi.org/10.1089/cmb.1998.5.255 PMID: 9672831

10. Xiao D, Li W, Zhang Z, He L. Solving maximum cut problems in the Adleman- Lipton model. Biosystems. 2005; 82(3): 203–7. https://doi.org/10.1016/j.biosystems.2005.06.009 PMID: 16236426

11. Hsieh SY, Chen MY. A DNA-based solution to the graph isomorphism problem using Adleman–Lipton model with stickers. Applied Mathematics and Computation. 2008; 197(2): 672–86.

12. Yang X, Lu Q, Li C, Liao X. Biological computation of the solution to the quadratic assignment problem. Applied Mathematics and Computation. 2008; 200(1): 369–77.

13. Nehi HM, Hamidi F. A comment on "Biological computation of the solution to the quadratic assignment problem". Applied Mathematics and Computation. 2012; 218(21): 10759–61.

14. Wang Z, Zhang Y, Zhou W, Liu H. Solving traveling salesman problem in the Adleman–Lipton model. Applied Mathematics and Computation. 2012; 219(4): 2267–70.

15. Wang Z, Ren X, Ji Z, Huang W, Wu T. A novel bio-heuristic computing algorithm to solve the capacitated vehicle routing problem based on Adleman–Lipton model. Biosystems. 2019; 184:103997. https://doi.org/10.1016/j.biosystems.2019.103997 PMID: 31369836

16. Pellerin R, Perrier N, Berthaut F. A survey of hybrid meta-heuristics for the resource constrained project scheduling problem. European Journal of Operational Research. 2020; 280(2): 395–416.

17. Kurdi M, An effective genetic algorithm with a critical-path-guided Giffler and Thompson crossover operator for job shop scheduling problem. International Journal of Intelligent Systems and Applications in Engineering, 2019; 7(1): 13–18.

18. Zhou G, Zhou Y, Zhao R, Hybrid social spider optimization algorithm with differential mutation operator for the job-shop scheduling problem. Journal of Industrial & Management Optimization, 2019; 13(5): 1–16.

19. Cruz-Chávez MA, Peralta-Abarca JdC, Cruz-Rosales MH, Cooperative threads with effective-address in simulated annealing algorithm to job shop scheduling problems. Applied Sciences, 2019; 9(16): 3360.

20. Pongchairerks P A two-level meta-heuristic algorithm for the job-shop scheduling problem. Complexity, 2019; 1–11.

21. Peng C, Wu G, Liao TW, Wang H. Research on multi-agent genetic algorithm based on tabu search for the job shop scheduling problem. PLoS One. 2019; 14(9): e0223182. https://doi.org/10.1371/journal.pone.0223182 PMID: 31560722

22. Abdel-Kader RF, An improved PSO algorithm with genetic and neighborhood-based diversity operators for the job shop scheduling problem. Applied Artificial Intelligence, 2018; 32(5): 433–462.

23. Zhang CY, Li P, Rao Y, Guan Z, A very fast TS/SA algorithm for the job shop scheduling problem. Computers & Operations Research, 2008; 35(1): 282–294.

24. Xing LN, Chen YW, Wang P, Zhao QS, Xiong J, A knowledge-based ant colony optimization for flexible job shop scheduling problems. Applied Soft Computing, 2010; 10(3): 888–896.

25. Deaton R, Garzon M, Rose J, Franceschetti D, Stevens S Jr. DNA computing: A review. Fundamenta Informaticae. 1997; 30: 23–41.

26. Zhixiang Y, Jianzhong C, Yan Y, Ying M. Job shop scheduling problem based on DNA computing. Journal of Systems Engineering and Electronics. 2006; 17(3): 654–9.

27. Wang Z, Ji Z, Wang X, Wu T, Huang W. A new parallel DNA algorithm to solve the task scheduling problem based on inspired computational model. Biosystems. 2017; 162: 59–65. https://doi.org/10.1016/j.biosystems.2017.09.001 PMID: 28890344

28. Păun G, Rozenberg G, Salomaa A. DNA computing: new computing paradigms: Springer; 1998.

29. Mohan J, Lanka K, Rao AN. A review of dynamic job shop scheduling techniques. Procedia Manufacturing. 2019; 30: 34–9.

30. Wang SJ, Tsai CW, Chiang MC. A high performance search algorithm for job-shop scheduling problem. Procedia Computer Science. 2018; 141: 119–26.

31. Beasley J. E. O-L. Distributing test problems by electronic mail. Journal of the Operational Research Society. 1990; 41(11): 1069–72.

32. Fisher H, Thompson GL. Probabilistic learning combinations of local job-shop scheduling rules. Industrial Scheduling. 1963; 225–51.

33. Lawrence S. Supplement to resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques. Graduate School of Industrial Administration. 1984; 4(7): 4411–7.

34. Kurdi M. An effective new island model genetic algorithm for job shop scheduling problem. Comput Oper Res. 2016; 67:132–42.

35. Asadzadeh L. A local search genetic algorithm for the job shop scheduling problem with intelligent agents. Comput Ind Eng. 2015; 85: 376–83.

36. Cheng L, Zhang QZ, Tao F, Ni K, Cheng Y. A novel search algorithm based on waterweeds reproduction principle for job shop scheduling problem. Int J Adv Manuf Tech. 2016; 84(1–4): 405–24.

37. Lin TL, Horng SJ, Kao TW, Chen YH, Run RS, Chen RJ, et al., An efficient job shop scheduling algorithm based on particle swarm optimization. Expert Systems with Applications, 2010; 37(3): 2629–2636.

38. Kurdi M, A new hybrid island model genetic algorithm for job shop scheduling problem. Computers & Industrial Engineering, 2015; 88: 273–283.

39. Zobolas GI, Tarantilis CD, Ioannou G, A hybrid evolutionary algorithm for the job shop scheduling problem. Journal of the Operational Research Society, 2009; 60(2): 221–235.

40. Yamada T, Nakano R. A genetic algorithm applicable to large-scale job shop problems. In: Manner R, Manderick B, editors. Proceedings of the second international workshop on parallel problem solving from nature (PPSN'2). Belgium; 1992; 281–90.

41. Storer RH, Wu SD, Vaccari R. New search spaces for sequencing problems with applications to job-shop scheduling. Management Science, 1992; 38(10): 1495–509.