Evolutionary Intelligence
https://doi.org/10.1007/s12065-020-00453-1

RESEARCH PAPER

DNAVS: an algorithm based on DNA-computing and vortex search algorithm for task scheduling problem

Nillofar Jazayeri[1] · **Hedieh Sajedi**[1]

Received: 13 December 2019 / Revised: 11 May 2020 / Accepted: 7 July 2020

Abstract
In this paper, we present DNAVS algorithm for the general job-shop scheduling problem (also known as Flexible JSSP). JSSP is an NP-complete problem, which means there is probably no deterministic or exact algorithm that can find the optimum solution in polynomial time for arbitrary instances of the problem, unless, P = NP. In the DNA computing algorithm, although the hardware is not accessible, in the future, we will have those computers that work based on biological hardware. Their most important advantages over silicon-based computers are their capacity for data storage. DNAVS is an improvement of the DNA computing algorithm by using a metaheuristic search algorithm called vortex search (VS) algorithm. The strongness of DNA computing is its parallelization. In the unavailability of DNA computers, the DNAVS reduces the time complexity of DNA computing by employing the VS algorithm. The efficiency of the DNAVS has been tested with standard test instances of job-shop scheduling problems. Our implementation results show that the DNAVS algorithm works effectively even for large scale instances on currently silicon-based computers.

Keywords Job shop scheduling · Vortex search · DNA computing

1 Introduction

The job-shop scheduling problem is one of the most studied combinatorial optimization problems. In a job-shop scheduling problem, n jobs have to be processed by *m* dedicated machines. Based on the literature, there are 14 classes of JSSP includes Deterministic JSSP, General JSSP, static JSSP, dynamic JSSP, no-wait JSSP, large-scale JSSP and etc. [1]. In this paper, we consider two well-known versions of JSSP, a standard version called Simple JSSP and a generic version or JSSP (GJSSP). In the standard version, each job just has one operation, but in GJSSP, each job has n operation that each one should be done by one machine, and we know which operation should be done by which machine (given an n × m matrix called priority matrix). Each job has to visit all the machines in a specific order. The time each job requires to be executed in each machine is given (an *n***m* input matrix called data matrix). The jobs cannot overlap in the machines, and no job can be processed simultaneously by more than one machine. The goal is to schedule the jobs so as to minimize the makespan, which is the maximum of the machines' completion time. From a complexity point of view, the JSSP problem is NP-hard [2]. Even some 'simplified versions' of the problem are NP-Hard [3, 4]. Some particular cases of the job-shop scheduling problem can be solved in polynomial time, such as the problem with two machines and no more than two operations per job [5] and the problem with two machines and unitary processing times [6]. A Dynamic Programming approach was proposed in [7] for the classical job-shop scheduling problem, with complexity proven to be exponentially lower than exhaustive enumeration.

Most of Metaheuristic algorithms were inspired from nature. In the past two decades, several meta-heuristic algorithms have been proposed like Genetic algorithm, Differential Evolution algorithm, Simulated Annealing, Ant colony optimization, Partial swarm optimization, Artificial bee colony, etc. [8–15]. The vortex search algorithm is a metaheuristic algorithm that was inspired from the vortical flow of the stirred fluids [16]. The main feature of the VS algorithm is the adaptive distance step, which significantly improves the performance of the search mechanism. The VS algorithm balances the weak location of

✉ Hedieh Sajedi
hhsajedi@ut.ac.ir

[1] School of Mathematics, Statistics and Computer Science, College of Science, University of Tehran, Tehran, Iran

Published online: 16 July 2020

the neighborhood and the strong location to determine the optimal solution. In addition, the algorithm converges almost at the optimal point, and therefore, responds with an exploitation method to adjust the updated solution to the desired result. Hence, the radius of the neighborhood decreases with each iteration. The VS algorithm definitively places a solution at specified low and upper limits and converts it into an optimal global point for optimization [17].

Although the solutions provided by metaheuristics may not be optimal solutions, they are widespread mostly because of their simplicity and flexibility means that they can adapt to different problems by finding a balance between exploration and exploitation. They also provide fast and robust solutions.

Several heuristic approaches have been proposed for the job-shop scheduling problem, and their main challenge is obtaining reasonable quality solutions. A well-known procedure is the so-called shifting bottleneck procedure by Adams et al. [18]. Van Laarhoven et al. [19] and Steinhofel et al. [20] used Simulated Annealing procedures for obtaining feasible solutions to the problem. Genetic Algorithm (GA)-based scheduling methods were proposed by Croce et al. [21], Dorndorf, and Pesch [22], Wang et al. [23] and Zhang et al. [24]. A tabu search procedure was recommended in the works by Taillard [25] and Nowicki and Smutnicki [26]. Zhang et al. [27] proposed a hybrid approach combining Tabu search and simulated annealing, which is efficient for finding optimal or near-optimal solutions for many benchmark instances of the problem. Rego and Duarte proposed a procedure that combines the basic shifting bottleneck procedure by Adams et al. [18] with a flexible search procedure based on the so-called filter-and-fan method [28]. Recently, an improved Genetic Algorithm for the flexible job-shop scheduling problem with multiple time constraints was proposed [24]. Table 1 summarizes some research works used GA to solve the job scheduling problem.

DNA computing is a branch of biomolecular computing that uses DNA data format instead of the traditional silicon-based computer technologies. The idea that individual biological molecules could be used for computation dates to 1959, when American physicist Richard Feynman presented his ideas on nanotechnology [34]. However, DNA computing was not physically realized until 1994, when American computer scientist Leonard Adleman showed how molecules could be used to solve computational problems [35].

DNA computation is characterized by a broad parallelization given by DNA strands, which means that the first power of DNA computation is the data structure with much more parallelization than neural networks and GA. The second power of DNA computing comes from the complement between the two strands of DNA, which has now been

Table 1 Related works done on job shop and genetic algorithms

Goal	Creating neighbors	Explanation
An improved version of GA is proposed [29]	Crossover and Selection operations are improved, and two methods for mutation are used the first one called "product sequence" picks two genes and replaces them. The second one is called "machine selection mutation," which selects the operation, picks the genes, and then mutes that chromosome based on the opted operation. Also, in the new population, the omitted individuals are replaced by the mutated individual	The initial population is still random, but after calculating the fitness of the first population, the individuals with the lowest fitness will remove
Gathering different versions of hybrid GA algorithms for job shop [30]	Different operators for creating neighbors in this survey have explained and compared	
A complete survey in [31] described genetic algorithms for three major types of task scheduling problems, flow shop, job shop, and open shop	The parents are chosen randomly from the initial population, and main operators for cross over is shifting an individual strand, which is shown by a vector of jobs	LISA package is finally represented for exact and heuristic searches of scheduling problems
Single genetic algorithm (SGA) and parallel genetic algorithm (PGA) were presented [32]	New operators proposed for GA and selecting the population	The results obtained by testing on five benchmark instances
introduced hybrid genetic algorithm [33]	The neighborhood structure is based on reversing operations on a critical path. The chromosome replaced by the best neighbor in the current population	The hill-climbing algorithm has used as genetic algorithm, and details of creating neighbors are described. The implementations are done on a set of problems (the BT set) and some other benchmark instance

successfully proven in the event of a connection, including encryption of an encrypted text and some robotic features [36].

In this paper, the DNAVS algorithm is proposed for solving the job shop scheduling problem. DNAVS algorithm is an improvement in the DNA computing algorithm via the Vortex search algorithm.

The remainder of the paper is organized as follows: In the next section, the background is presented, which includes notation, definitions, and basic properties of the problem. Then, the Vortex Search algorithm and some specifications for JSSP are described. In Sect. 3, the DNAVS algorithm is introduced. Some computational tests performed with the new procedure for some benchmark instances, and the results are presented and discussed in Sect. 4. Section 5 concludes the paper.

2 Definition and notation

2.1 The task scheduling problem

2.1.1 Simple JSSP

Given the time–cost matrix between n jobs and m machines, the problem arranges n jobs to m machines, demanding each job can be allocated to one machine only. The solution to the problem is to find the minimum finished time of the last machine on the task allocation. The problem is mathematically defined as:

$$cost\ function = \min(makespan\ between\ m^n\ possible\ solution)$$

$$makespan = \max\left(sigma_{k=1}^{n}\left(a_{ik}c_{ik}\right)\right)$$

$$a_{ik}\epsilon\{0,1\},\ m \le n$$

where $C=[c_{ik}]$ is time consumption matrix, $A=[a_{ik}]$ is the task assignment matrix that $a_{ik}=1$ means the kth job is assigned to the ith machine. Table 2 shows an example of time cost data ($C[ij]$) with two machines and five jobs. Figure 1a shows an assignment matrix, which also represents a possible solution for the simple JSSP where n = 5 and m = 2) and Fig. 1b displays a graphical version of that solution.

Table 2 An example of time cost data ($C[ik]$) with two machines and five jobs

	Job_1	Job_2	Job3	Job_4	Job_5
$Machine_1$	5	3	4	6	2
$Machine_2$	2	8	5	3	3

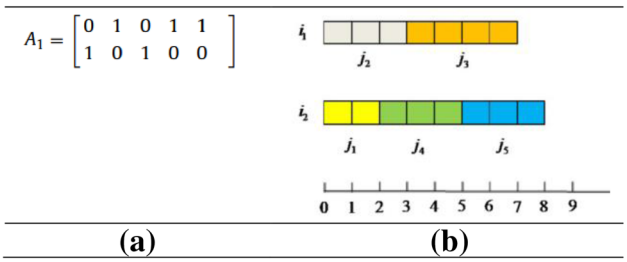


Fig. 1 **a** An assignment matrix, which also represents a possible solution for the simple JSSP where n = 5 and m = 2). **b** A graphical version of that solution

2.1.2 General JSSP

Consider the job-shop scheduling problem as defined in the previous section. Let $J=\{J_1, J_2, \ldots, J_n\}$ denote the set of jobs and $M=\{M_1, M_2, \ldots, M_m\}$ the set of machines. P_{ij} stands for the processing time of job j ∈ J on machine i ∈ M. A job consists of m task operations, each of which is associated with a specific machine. The sequence of operations for each job $j \in J$ is denoted by $\{\pi_j(1), \pi_j(2), \ldots, \pi_j(m)\}$, that is, for job $j \in J$, P_{ij} is the ith machine that jth job has to visit. $O=\{o_1,o_2, \ldots,o_{n\times m}\}$ is the set of operations. The first n operations refer to the first operation of each job (in the order of the jobs), operations o_{n+1}, o_{n+2}, …, o_{2n} concern the second operation of the n jobs, and so on.

Example 1 As an instance, in ft6 [37], it is assumed that m = 6 and n = 6. Tables 3 and 4 display the data and priority matrixes of the ft6 problem.

For every feasible solution for the job-shop scheduling problem, the minimized make span time, there is one and only one sequence of operations [7].

In the case of instance ft06 (by dynamic programming [7]) the following optimal sequence is obtained:

j1 op3 m6 j2 op3 m6 j3 op4 m6 j4 op5 m6 j5 op4 m6 j6 op6 m6
j3 op3 m5 j5 op3 m5 j6 op5 m5 j2 op5 m5 j4 op6 m5 j1 op6 m5
j1 op2 m4 j2 op2 m4 j6 op4 m4 j4 op4 m4 j3 op6 m4 j5 op6 m4
j1 op1 m3 j4 op1 m3 j3 op2 m3 j5 op1 m3 j6 op3 m3 j2 op6 m3
j3 op1 m2 j6 op1 m2 j2 op1 m2 j4 op3 m2 j5 op2 m2 j1 op5 m2
j4 op2 m1 j6 op2 m1 j1 op4 m1 j2 op4 m1 j3 op5 m1 j5 op5 m1

Figure 2 displays the Gantt chart of the optimal solution of ft06.

The cost function of General JSSP is the same as Simple JSSP. The cost function still is the maximum of the last-finished time between machines, but the makespan is defined and computed differently just because the operations have priority over each other, and the machines cannot run individually. In the simple version, each machine executes its

Table 3 Data [i, j]: = shows the time cost of *the j*th operation of the *i*th job

DATA	Op1	Op2	Op3	Op4	Op5	Op6
Machine 1	1	8	5	5	9	3
Machine 2	3	5	4	5	3	3
Machine 3	6	10	8	5	5	9
Machine 4	7	10	9	3	4	10
Machine 5	3	10	1	8	3	4
Machine 6	6	4	7	9	1	1

Table 4 Priority [i, j]: = shows the number of the machine that assigned to the *j*th operation of *i*th job

Priority	Op1	Op2	Op3	Op4	Op5	Op6
Job1	1	2	3	4	6	5
Job2	6	5	4	1	2	3
Job3	2	5	6	3	1	4
Job4	2	3	5	4	6	1
Job5	4	6	5	3	2	1
Job6	5	4	3	6	1	2

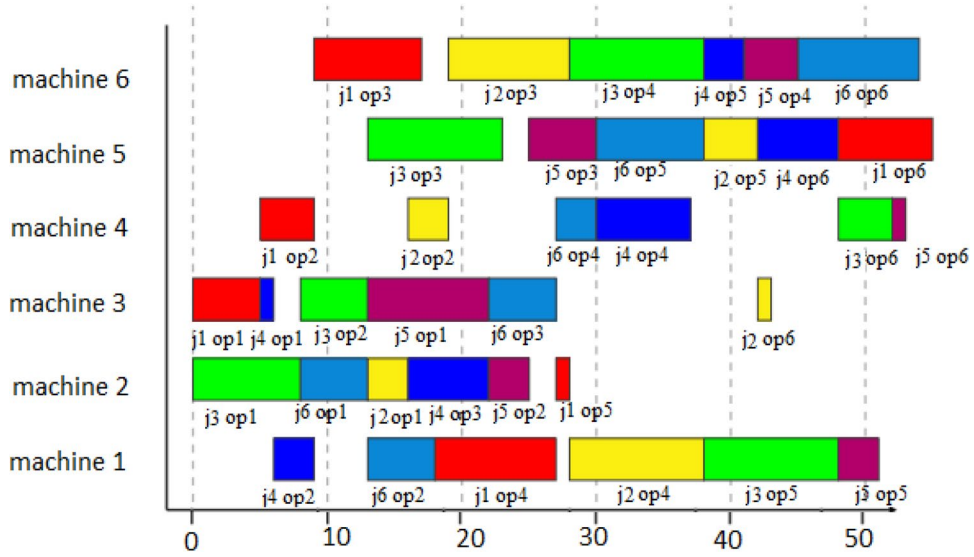


Fig. 2 This is the Gantt chart of the optimal solution of ft06; this solution has a well-known optimal value of 55 [26]

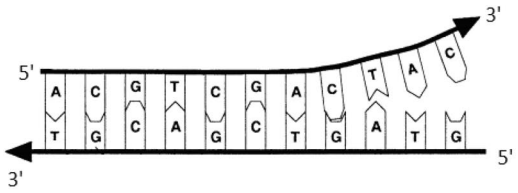
list of the job with no pause, but here each operation of a specific job that assigned to a machine should wait until the previous operation of the same job finishes.

2.2 DNA computing

DNA computing uses DNA, biochemistry, and molecular biology hardware rather than silicon-based computer technologies. DNA computing is a sort of parallel calculations that can be performed using many different DNA molecules at the same time [38]. DNA computing is based on the idea that molecular biology processes can be used to perform arithmetic and logic operations on information encoded as DNA strands. The DNA computing data representation will elaborate more in the following paragraphs.

According to DNA *structure*, DNA *is* made up of four different types of nucleotides; the bases are adenine (A), guanine (G), cytosine (C), and thymine (T). They have lied on a sugar-phosphate backbone. There are chemical bounds between two different DNA single strands called Watson = Crick (W = C) complementary. As Fig. 3 shows there are two main W–C bounds, A can bind to T on opposing single strand.

Most of the existing models of DNA computing have their formal basis in the theory of computing [7]. Computational power is measured in terms of what types of functions a given machine can perform. The most general



$5'CCCCCTCGACCCCC3'$ $5'AAAAAGCGCAAAAA3'$
$3'GGGGGAGCTGGGGG5'$ $3'TTTTTCGCGTTTTT5'$

Fig. 3 A DNA double-strand [7]. DNA Strands have directions from 3′ to 5′

abstract model of computing, and the one with the highest computational power, is known as the Turing machine. Alan Turing proposed the Turing machine as an abstract device capable of computing any computable function. Thus, to prove that a new model of computing achieves the highest level of computational power, it suffices to prove that it is equivalent to a Turing machine. It has been proved that DNA-computing is equal to the Turing model.

The main functions are:

Merge(), *Copy*(), *Separation*(), *Discard*(), *Read*(), *Append-Tail*(), *Append-head*(), *cleavage*(), *annealing*(), *Denaturation*(), *Sort*()

The functionality of the functions is entirely described in [39]. This DNA computing algorithm [39] solves the task scheduling problem in four significant steps, and the time complexity is O(n^2) (Which is not O(n^2). We show the real complexity in the implementation section.

2.3 Vortex search algorithm

Vortex search algorithm searches the possible solution's area using an adaptive manner; modeled as vortex pattern, which is explained in more detail in the following pseudo-code, the beneficial feature of the algorithm is that it finds the best exploration and exploitation for different problems if the parameters were selected correctly. The pseudo-code of the Vortex search is shown in Algorithm 1.

The vortex Search algorithm has four major processes, as follows:

1. The start node will be constructed based on vortex parameters, and it is usually the midpoint of the solution area.
2. All other candidate nodes or candidate solutions in the neighborhood will be generated.
3. The cost or goal function every neighbor will be calculated for each neighbor, and the best-found neighbor will be compared to the best-found solution, and the current node will be updated.
4. The radius of the neighborhood will be changed in each iteration by a descending function.

Algorithm 1. The pseudo-code of Vortex search.

Input:
Centre = C_0
start point = S
radius = r_0
fitness of best found solution = f_{best} = MIN
Current iteration = t=0
Calculate the fitness of start point := $f\ (S_{best})$
Iter =1;

While (maximum number of iterations is reached)
Candidate solution around S in t_th iteration:= C_t (s)
Generate C_t (S) by Gaussian distribution around center C_0
S' = Best solution in C_t (S)
If f(S') < f(S_{best})
$S_{best} = S'$
$f\ (S_{best}) = f(s')$
Else
#decrease the radius
$r\ = Radius\ (iter)$
#increment the iteration
$iter\ =\ iter + 1$
End while
#best solution is Sbest
return S_{best}

Figure 4 shows the search boundaries of the vortex algorithm after several iterations. Although the Vortex search algorithm is shown to be suitable for specific optimization problems, it has some downsides. According to the VS algorithm, candidate solutions are constructed around the central node by using a Gaussian distribution, and the next best-found solution is more bounded in the local area near the current node as the iteration goes on. Furthermore, the radius decreases each time. Consequently, the search area shrinks by the passage of time. Thus we need to get out of locality toward global optimal (if there are any global optimal solution) as quickly as possible. In this regard, some extensions of the VS algorithm have been proposed [40, 41]. One of the methods is used several start points instead

of just one at each iteration; it enhances the exploration in the solution area. Modified VS algorithm (MVS) is a bit more complicated in computation, although the implementation results show improvement in the final results of the algorithm in MVS. For more overview of convergence and efficiency studies of metaheuristic algorithms [35]. Figure 5 shows multiple vortices of MVS.

2.4 DNA computing algorithm

We firstly generate all possible combinations for task location assignment in the data pool. Also, for the task scheduling problem, each task should be assigned to only one machine. In the second step, we append the first machine's weight strands at the end of the previous ones in order to find the best result. In the third step, we successively compare other machine's time costs to count the last finished time of every assignment.

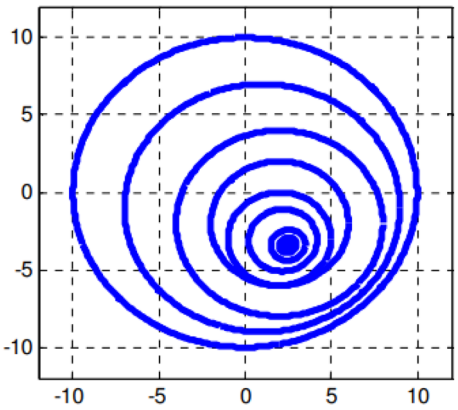


Fig. 4 Search boundaries of the vortex algorithm after several iterations; It is like vortex patterns on fluids [12]

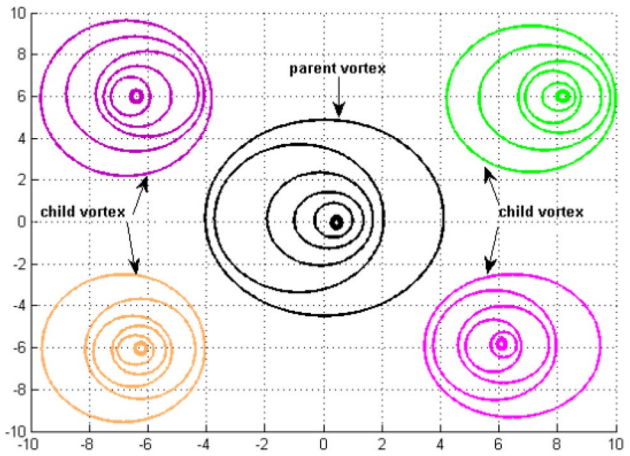


Fig. 5 Selecting multiple vortices instead of considering one vortex [12]

Meanwhile, the solution to the task scheduling problem has minimum time cost in all possible allocations [39].

Step 1 Generating all possible solution m^n: In the implementation, each solution is represented by an array with length $n+1$, which filled by the number of machines if $A[i]=3$ it means that ith -job is assigned to the third machine.

Example 2 In the following, the symbols Ai, Bj, (i ∈ {1, 2,…, m}, j ∈ {1, 2,…, n}) indicate different DNA singled strands having the same integer length. If $m=2$, $n=5$ each individual is represented like below:

$$[2,1,2,2,1,0] := A2B1A1B2A2B3A2B4A1B5$$

There are 2 machines and 5 jobs. $AiBj$ represents that j-job is assigned to i-machine.

The last empty index in the array will fill with the strand cost, which is the maximum of the last finished time of m machines.

Step 2 Appending corresponding time cost for the start point.

Step 3 Similarly, we get the second individual time cost. Again the functioned that mentioned in Step 2 does this step.

Step 4 Select the best one with minimum time cost as the solution in that recursive function. It stores every solution last finished time and updates it if the cost is less and returns the solution with the minimum last finished time.

3 DNAVS algorithm

Based on the original algorithm, the whole operation is done by bio operations in the lab. In the first stage, the population is initialized, which is exactly with the same size as the Test tube. The upper bound and lower bound of each element in operation will be set in this step. In the next stage, when the population is defined, and the n-dimension solution area represented, then we shall start searching from the first possible solution known as first. DNA strand, it had better selection because it affects the time of convergence, and the better start points will reach to best optimum sooner, but here we set it the mean of the area. Furthermore, in an improved version, the start point could be the best node among a small population from all over the area, or we could have several start points. The size of the radius around the current location is equal to the value of a linearly decreasing function in that iteration. At first, the radius is n (the whole area is considered), then it reaches to 0 in the convergence point, when the search is also finished. Figure 6 shows the flow chart of the DNAVS algorithm.

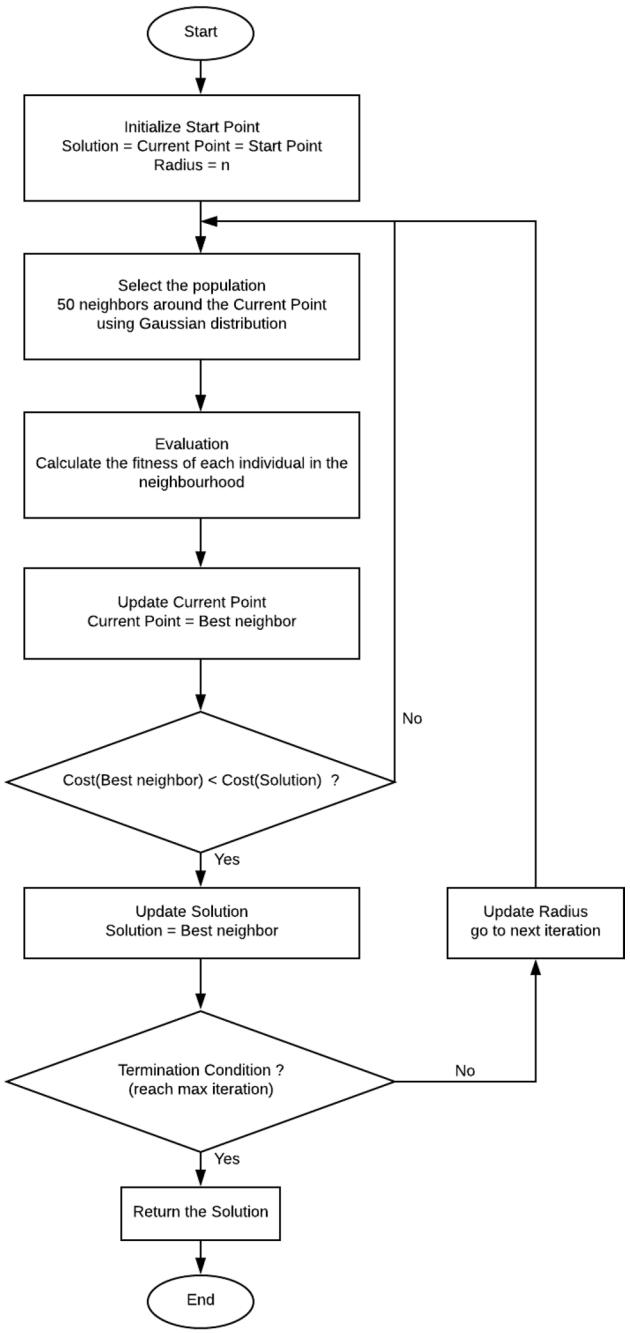


Fig. 6 The flow chart of the DNAVS algorithm

Definition The neighborhood of strand = T in radius = r is a set of DNA strands that are different in less than r elements in their vector feature. Here we chose the 70 percent of the neighborhood population in the radius and 30 percent for searching out of the radius to improve the exploration and prevent to stick in local optima.

3.1 Improved version of data representation

Based on what we described in the form of data representation earlier in Example 2, in DNA computing based job shop scheduling [39], despite of its creativity and compact appearance, it had some drawbacks. Firstly, that input format of data is just dedicated to simple JSSP, and it would not work on GJSSP where each job has its own operations sequence. Unless, a new variable added to show the operation number of a GJSSP problem like what showed in Example 1. However, it does not have any general form of representation. Secondly, any changes in the basic representation (Example 2) do not lead to another possible solution necessarily. This issue also is just related to the improved version of the algorithm or DNAVS. Because we need to create neighbors in each iteration by making some changes in the central point. Not any representation in that form is a possible job assignment to machines because of the main rules in JSSP for instance one of them says "any operation should be done after the previous one for a specific job". Thus, for example, if the changes reach us to a solution where operation 2 and 3 are done by machine 5 in reverse order, then this is an unacceptable solution. Briefly speaking, the space of representation in [39] is much bigger than the actual possible solution area in GJSSP, but it does okay in Simple JSSP (m^n). To overcome previous disadvantages, the following representation is introduced. This version is used for the genetic search of GJSSP in [21].

Example 3 The same solution in Feature 3 and Example 1:

1 3 1 1 3 4 2 5 2 4 6 6 3 6 6 3 2 1
4 2 4 5 4 2 5 4 3 6 5 3 6 1 5 2 1 5

Read it from left to right and when reach a new number go to the job with that number and see which operation had not done yet and pick the one with least number operation from the top of the queue and assign it to the proper machine according to the priority matrix (shown in Table 3).

Based on the instruction, it is easy to prove that the previous representation is equal to exactly one possible solution.

This representation allows the replacement in the numbers and reaches to another possible solution. Therefore, those mentioned disadvantages of data representation in the DNA computing algorithm [39] have been solved.

3.2 Creating neighbors

If the points are strings of numbers with length l then the distance between two points should be computed using

Hamming distance. If the Hamming distance between Point 1 and Point 2 is 3, this means that exactly three positions in them have different values. We used that distance to generate neighbors all over the central point that the distance of them from the current location follows Gaussian distribution. The improved version of the original DNA computing algorithm in [39] is proposed by reducing the computation with a genetic-like-search algorithm (Vortex search) the need of generating from one point to another one is done by as mutation or crossover adopted from genetic algorithms with following instructions. First of all, a distance-vector D is created based on a Gaussian distribution with size 50. Then 50 neighbors generated based on that distance-vector, which means that the jth neighbor has $D(j)$ Hamming distance with the central point.

3.3 Evaluation and fitness function

Fitness function in DNAVS is like the original version in [39], but instead of calculating the whole DNA strands Tube, containing all possible solutions, just the neighbor's cost is calculated. In the implemented function, the fitness function returns two arguments, the solution strand and the last finished time of machines; the last finished time of each solution strand is appended at the end of it. The code also prints the Gantt chart of the solution. For example, for the LA01 benchmark instance, the output after ten runs in less than 1 min with the Intel(R) Core(TM) i5, 2.5 GHz processor.

3.4 Radius function

The radius could be any decreasing function with two main properties, $Radius\ (0)=n$ (the maximum number of containing the whole space) in first iteration and $Radius\ (Iteration\_max)=0$, meaning that in the convergence point it shrinks to a single point. We used a simple linear decreasing function.

$$Radius(iter) := \frac{n(Iteration\_max - iter)}{Iteration\_max(1 + iter)}$$

$$T(Initialization) := O(n + m + 7)$$

The added number 7 is for seven instructions of vortex algorithm for initialization. "$n+m$" is for initialing the first strand or first possible solution, which is a vector with length equal to $n+m$ (according to the improved version of data representation).

$$\begin{aligned} T(Loop) := (&T(Selecting\ population) \\ &+T(Evaluation) + T(Update\ node) \\ &+T(Update\ solution) + T(Update\ radius)) \\ &\times Max\_Iteration \end{aligned}$$

$$m \le n$$

$$\begin{aligned} T(Selecting\ population) &:= O(50 \times (n + m)) \\ &= O(n + m) \le O(2n) = O(n) \end{aligned}$$

Based on [3] *computation for calculating fitness we have:*

$$T(Evaluation) := O(50 \times (5n + 13)) = O(n)$$

$$T(Update\ node) := O(1)$$

$$T(Update\ solution) := O(1)$$

$$T(Update\ radius) := O(1)$$

$$\begin{aligned} T(Loop) := &(O(n) + O(n) + O(1) + O(1) + O(1)) \\ &\times Max\_iteration = O(n) \times Max\_iteration = O(C \times n) \end{aligned}$$

$$T\_final := T(Initialization) + T(Loop) := O(n) + O(C \times n) < O\left(n^2\right)$$

Therefore the complexity is $O(n)$ better than what is proposed in original version of the DNA computing algorithm for JSSP in [3].

4 Experimental results

In this study, the time cost of the DNA algorithm in [3] for the task scheduling problem has analyzed. Our implementation assured that the efficiency of the proposed algorithm is hugely better than the original DNA algorithm. The input data are based on well-known benchmark table (LA01–LA02 for JSSP), and we used a random uniform generated data (for Simple JSSP).

We explained how finding feasible or non-feasible solutions can make some difficulties and how to affect the convergence point of the algorithm. Then some extensions in both representation and algorithm steps have proposed, and their time-order complexity has analyzed either. In addition, the creating neighbor operator explained in detail and compared it to other related works in this area. We also mentioned that premature convergence is a common problem among them.

Generating all possible solutions means that a big test tube of DNA strands is produced. Our main effort is to search this solution area extremely more cautiously without generating all solutions (feasible or non-feasible) and controlling the distance to the optimum solution. The algorithm terminates after reaching max_iteration (other improved termination conditions have extra criteria like stop after not finding the better solution in, for example, ten iterations or

Table 5 Based on random data generation

Simple JSSP	DNA computation algorithm	DNAVS algorithm
Complexity	O(fitness function × m^n)	O(fitness function × max_iteration × neighbourhood_size)
Number of iterations	1000	1000
Neighborhood_size	1 in each iteration	50 in each iteration
Result of instance—10 machines, 5 jobs	200 (in 20 s)	96 (in 20 s)
Result of instance—10 machines, 10 jobs	240 (in 26 s)	140 (in 26 s)
Result of instance—20 machines, 15 jobs	692 (in 40 s)	136 (in 40 s)

Table 6 Properties of DNA and DNAVSA Algorithms

General JSSP	DNA computation algorithm	DNAVSA algorithm
Complexity	O(fitness function × $n!^m$)	O(fitness function × max_iteration × neighbourhood_size)
Time	5 h	5 h
Number of iterations	1000	1000
Neighborhood_size	1 in each iteration	50 in each iteration

Table 7 Comparison of DNA with DNAVS and with other algorithms

	Best solution	DNA computation algorithm	DNAVSA algorithm	modified clonal selection algorithm [42]	Parallel GA [43]
LA01: 5 machines, 10 jobs	666	2227	666	666	666
LA02: 5 machines, 10 jobs	655	1962	655	655	659

after 1 h). The main advantage over the original version of DNA computing is we are sure that this way of processing data is far better than the first approach DNA algorithm which creates all permutations of assigning jobs to machines; because we now know how to move toward the best solution without sticking to local optima for next several iterations and not blindly try to find the global optimum independent form what have seen in previous iterations.

4.1 Comparing DNA algorithm and DNAVS algorithm

Tables 5 and 6 provide comparison results obtain by implementing both classic DNA-Computing algorithm and the improved version of that algorithm called DNAVS. Table 7 compared the algorithms in solving LA01 and LA02 problems. An illustration of the results of the DNAVS algorithm is shown in Fig. 6. The convergence and exploration and exploitation are depicted better there.

4.2 Other performance curves

The curves in Fig. 7 shows the relation between generations (or iteration numbers) and the best-found neighbour (or best found job-shop solution) of DNAVS algorithm and DNA computing on two benchmark instances (LA01 and LA02). As the diagram shows in both instances, DNAVS can find the optimum solution with acceptable speed.

5 Conclusion

In this paper, we Improved DNA computing algorithm to solve the job shop scheduling problem in General form, where each job also has several operations. In this paper, a new and an effective combination of two metaheuristic algorithms, namely DNA-computing Algorithm and the Vortex search, has been proposed. This hybridization called DNAVS. DNA computing algorithm uses the chemical operations on DNA strands based on Adleman-Lipton model. The vortex search algorithm is inspired from vortical moves in fluid and is a method that helps DNA computing algorithms not to create the whole space or the whole possible solution area. All the operations in the DNAVS algorithm are accessible with today's molecular biotechnology. Our complexity analysis and computer implementations assured that with this new computing algorithm, it is possible to get a solution from a very small initial

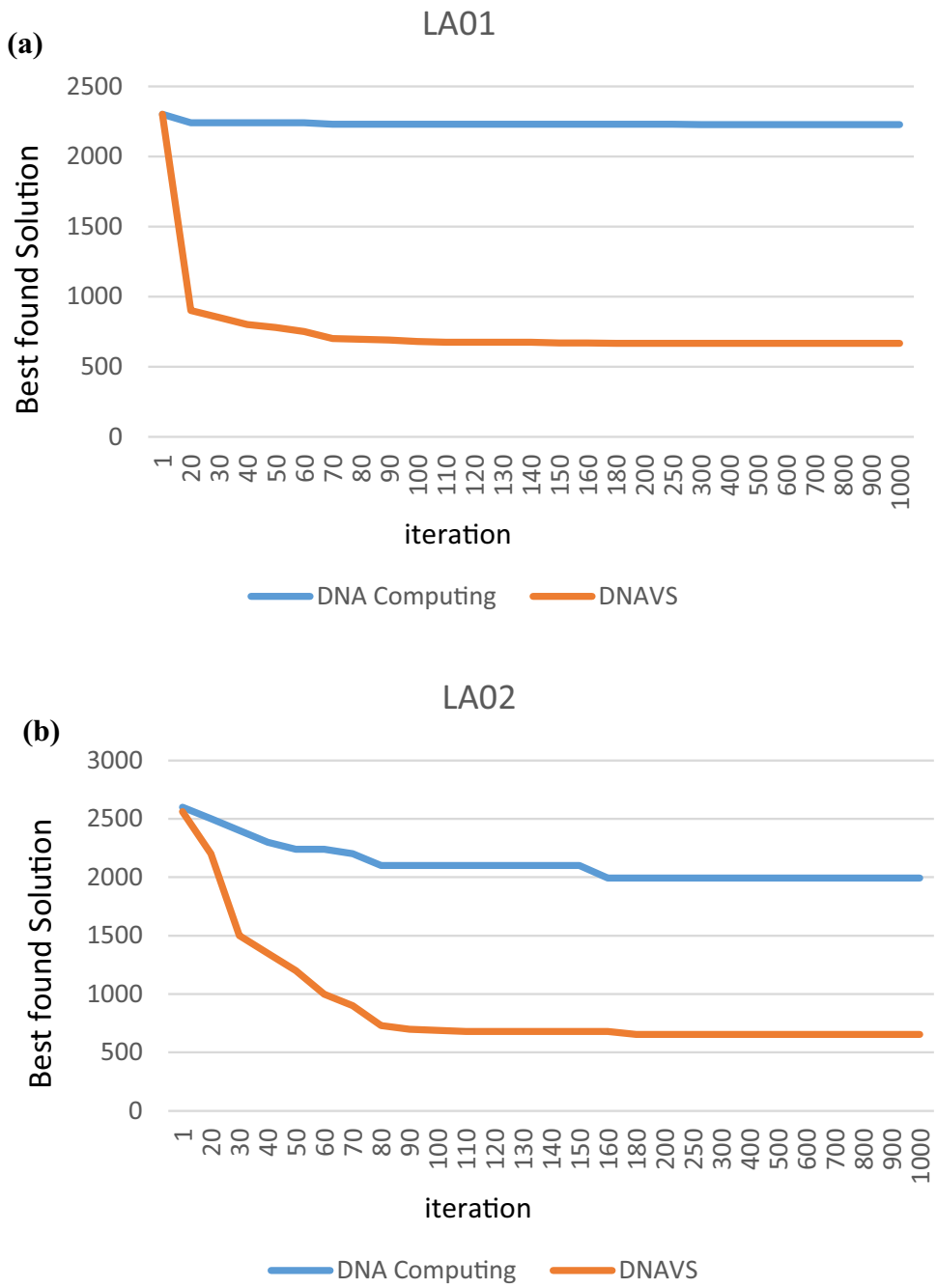


Fig. 7 The convergence diagram of DNAVS and DNA computing in finding the solution of **a** LA01 and **b** LA02

population. It has been shown that for random functions and problems, the Vortex search had a promising result, and it is very probable that it finds the correct solution in a few iterations. The computational analysis also has shown that the time complexity of the DNAVS algorithm is a linear function of the length of strands (here is $n+m$), which is better than the simple version of DNA computing algorithm $O(n^2)$. This helps DNA computers to solve the np-complete or np-hard problems more effectively and with a better approximation of the actual solution.

References

1. Abdolrazzagh-Nezhad M, Abdullah S (2017) Job shop scheduling: classification, constraints and objective functions. Int J Comput Inf Eng 11(4):429–434
2. Karaboga D (2005) An idea based on honeybee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department
3. Gromicho JAS, van Hoorn JJ, Saldanha-da-Gama F, Timmer GT (2012) Solving the job-shop scheduling problem optimally by dynamic programming. Comput Oper Res 39(12):2968–2977

4. Brucker P (1998) Scheduling algorithms, 2nd edn. Springer, Secaucus
5. Jackson JR (1995) Scheduling a production line to minimize maximum tardiness, Issue 43 of Research report. Los Angeles University of California Management Sciences Research Project
6. Hefetz N, Adiri I (1982) An efficient optimal algorithm for the two-machines unit-time job shop schedule-length problem. Math Oper Res 7:354–360
7. Daley MJ, Kari L (2002) DNA computing: models and implementations. Comments Theor. Biol. 7:177–198
8. Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor
9. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11:341–359
10. Kirkpatrick S, Gelatt CD Jr, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
11. Černý V (1985) Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. J Optim Theory Appl 45:41–51
12. Dorigo M (1992) Optimization, learning and natural algorithms. Ph.D. Thesis, Politecnico di Milano, Italy
13. Kennedy J, Eberhart RC (1995) Particle swarm optimization. In: 1995 IEEE international conference on neural networks, vol 4, pp 1942–1948
14. Omaraa FA, Arafa MM (2010) Genetic algorithms for task scheduling problem. J Parallel Distrib Comput 70:13–22
15. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471
16. Hekmatnia E, Sajedi H, Habib Agahi A (2020) A parallel classification framework for protein fold recognition. Evol Intel. https://doi.org/10.1007/s12065-020-00350-7
17. Qyyum MA, Yasin M, Nawaz A, He T, Ali W, Haider J, Qadeer K, Nizami AS, Moustakas K, Lee M (2020) Single-solution-based vortex search strategy for optimal design of offshore and onshore natural gas liquefaction processes. Energies 13(7):1732
18. Adams J, Balas E, Zawack D (1988) The shifting bottleneck procedure for job shop scheduling. Manag Sci 34:391–401
19. Van Laarhoven PJM, Aarts EHL, Lenstra JK (1992) Job shop scheduling by simulated annealing. Oper Res 40:113–125
20. Steinhofel K, Albrecht A, Wong CK (1999) Two simulated annealing-based heuristics for the job shop scheduling problem. Eur J Oper Res 118:524–548
21. Croce FD, Tadei R, Volta G (1995) A genetic algorithm for the job shop problem. Comput Oper Res 22:15–24
22. Dorndorf U, Pesch E (1995) Evolution based learning in a job shop scheduling environment. Comput Oper Res 22:25–40
23. Wang C, Shi H, Zuo X (2020) A multi-objective genetic algorithm based approach for dynamical bus vehicles scheduling under traffic congestion. Swarm Evolut Comput 54:100667
24. Zhang G, Yifan H, Sun J, Zhang W (2020) An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. Swarm Evolut Comput 54:100664
25. Taillard ED (1994) Parallel taboo search techniques for the job-shop scheduling problem. ORSA J Comput 6:108–117
26. Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop scheduling problem. Manag Sci 42:797–813
27. Zhang CY, Li P-G, Rao Y-Q, Guan Z-L (2008) A very fast TS/SA algorithm for the job shop scheduling problem. Comput Oper Res 35:282–294
28. Rego C, Duarte R (2009) A filter-and-fan approach to the job shop scheduling problem. Eur J Oper Res 194:650–662
29. Janes G, Perinic M, Jurkovic Z (2017) Applying improved genetic algorithm for solving job shop scheduling problems. Teh vjesn 24(4):1243–1247
30. Bhatt N, Chauhan NR (2015) Genetic algorithm applications on job shop scheduling problem: a review. In: International conference on soft computing techniques and implementations
31. Werner F. Genetic algorithms for shop scheduling problems: a survey. Otto-von-Guericke-Universität, Fakultät für Mathematik, 39106 Magdeburg, German
32. Park BJ et al (2003) A hybrid genetic algorithm for the job shop scheduling problems. Comput Ind Eng 45(4):597–613
33. Gonzalez MA, Vela CR, Varela R (2008) A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In: Proceedings of the eighteenth international conference on automated planning and scheduling
34. https://www.britannica.com/science/molecule. Accessed 12 Mar 2019
35. Yang X (2011) Metaheuristic optimization: algorithm analysis and open problems, vol. 6630, pp 21–32
36. Negoita MG, Neagu D, Palade V (2005) Computational intelligence: engineering of hybrid systems. Springer, Berlin
37. Qiu X, Lau HYK (2014) An AIS-based hybrid algorithm for static job shop scheduling problem. J Intell Manuf 25:489–503
38. Lewin DI (2002) DNA computing. Comput Sci Eng 4(3):5–8. https://doi.org/10.1109/5992.998634
39. Wang Z, Ji Z, Wang X, Wu Tu, Huang W (2017) A new parallel DNA algorithm to solve the task scheduling problem based on inspired computational model. Biosystems 162:59–65
40. Doğan B (2016) A modified voertex search algorithm for numerical function optimization. Int J Artif Intell Appl 7(3):37–54
41. Sajedi H, Razavi SF (2016) MVSA: multiple vortex search algorithm. In: IEEE 17th international symposium on computational intelligence and informatics (CINTI). https://doi.org/10.1109/cinti.2016.7846398
42. Atay Y, Kodaz H (2014) Optimization of job shop scheduling problems using modified clonal selection algorithm. Turk. J Electr Eng Comput Sci 22(6):1528–1539
43. Asadzadeh L, Zamanifar K (2010) An agent-based parallel approach for the job shop scheduling problem with genetic algorithms. Math Comput Model 52(11–12):1957–1965