Volume: 1 Issues: 1 [September, 2016] pp.21-42]
Journal of Information System and Technology Management
eISSN: 0128-1666
Journal homepage: www.jistm.com

Applying the Clonal Selection Principle to Solving Flexible Job-Shop Scheduling Problem

Ahmad Shahrizal Muhamad[1]
Zalmiyah Zakaria[2]
Safaai Deris[3]

[1]Universiti Teknologi Malaysia, Skudai, Johor, xenixware@gmail.com
[2]Universiti Teknologi Malaysia, Skudai, Johor, zalmiyah@utm.my
[3]Universiti Malaysia Kelantan, safaai@umk.edu.my

Abstract: *This work deals with the problems of flexible job-shop scheduling and proposes ways to find the most optimal and robust solutions. Finding such solutions is of the utmost important for real- world applications, as scheduling operates in a dynamic environment. Several methods have been used to solve job-shop scheduling problems and the method proposed here is artificial intelligence by using the clonal selection principle algorithm. The advantage of this algorithm is that it is structured in such a way as to imitate the natural immune system. The results produced by this method compare well with the results of previous research.*

Keywords: scheduling, artificial intelligence, flexible job-shop scheduling, robustness, artificial immune system, evolutionary computation.

Introduction

Flexible job-shop scheduling problem is an extension of the classical job-shop scheduling problem which allows an operation to be processed by any machine from given set of available machines. Like the job-shop, flexible job-shop still consists of a set of n jobs $\{j_1, j_2, \ldots, j_n\}$ with a number of m machines $\{m_1, m_2, \ldots, m_m\}$. In each job J_i there are a series of operations $\{o_{i,1}, o_{i,2}, \ldots, o_{i,n_i}\}$ with each operation having a processing time $\{\tau_{i1}, \tau_{i2}, \ldots, \tau_{im}\}$. For the job-shop each operation only can be process on one machine. Otherwise, for the flexible job-shop, each operation $o_{i,j}$, i.e. the operation j of job i, can be processed on any among a subset $M_{i,j} \subseteq M$ of compatible machines. In other words, it does not belong to the subset of compatible machines for that operation or no operation j of job i. Bruker and Schlie [1] were among the first to address flexible job-shop scheduling problem. In flexible job-shop scheduling (FJSS), the definition of a job is a piece of work that goes through a series of operations. The shop is the place or machinery for manufacturing or repairing goods, and

scheduling is the process that aims to deduce the order of processing. The rule to produce a solution FJSSP as follows:

i. Each operation in the job series visiting the machine one time only.
ii. At a particular time, one machine can only address at most one operation.
iii. The preceding operation must be scheduled to be completed before the machine can proceed to its next operation in the job series.
iv. All operations are required to be completed continuously without any interruption for a specific machine.
v. Number of machine for each type of machine is one.
vi. After operation completed at current machine, it will be transferred to another machine as soon as possible with transfer time disregard.
vii. Each operation having specific activity and with specific processing time.
viii. Each operation can be processed on any among a subset $M_{i,j} \subseteq M$ of compatible machines.

The ultimate aim of flexible job-shop scheduling is to produce scheduling that minimizes the total time taken to complete all the activities. Figure 1 illustrates the flexible job-shop scheduling problem. The symbol ∞ in the Figure 1 means that a machine cannot execute corresponding operation.

	Operation	Machine		
		M_1	M_2	M_3
Job$_1$	O_1	6	6	∞
	O_2	∞	5	∞
	O_3	4	5	4
Job$_2$	O_1	∞	6	∞
	O_2	∞	5	7
	O_3	7	9	∞
	O_4	6	3	∞
Job$_3$	O_1	5	3	3
	O_2	4	∞	∞

Figure 1: Example of flexible job-shop scheduling problem

The solution to any optimization problem is evaluated by an objective function. Objective functions are associated with minimized cost, resources and time. There are several objective functions within the job-shop scheduling problem, the common objective function as follows:

i. Minimize the total completion time (makespan)
ii. Minimize the completion time for each job (flow-time)
iii. Minimize the maximum lateness for problem with due date (lateness)
iv. Minimize the maximum waiting time on machines (tardiness)
v. Minimize the maximum starting time (earliness)
vi. Minimize the number of tardy jobs

In this paper we discuss how to minimize the *makespan* and determine the completion time for the last job to be completed. The *makespan* is important when having a finite number of jobs and is closely related to the throughput objective. When the maximum completion time are minimized, the machine resources can be used to process other jobs as soon as possible, and other resources can be indirectly saved, such as electricity and man power. In addition, many products can be produced in the shortest time, efficiently fulfilling the demand of the product. As a result, French [2] states that when considering minimum *makespan* at least one of the optimal solutions to a job-shop problem is semi-active.

Problem Solving Methods for Job-Shop Scheduling

Recently, artificial intelligence has become the popular technique for solving problems in scheduling and specifically in job-shop scheduling. Currently, the best known production scheduler is the intelligent scheduling and information system. The production scheduler will be able to visually optimize real-time work-loads in various stages of production. Then the production scheduler becomes the intelligent scheduling and information system and it is able to manage the schedule dynamically. The advantage of artificial intelligence techniques is that their computation times are much shorter than traditional techniques. They provide high quality solutions at low computational times for even very complex problems. However, their performance cannot be guaranteed for any particular problem.

There are several techniques for artificial intelligence [2, 3], which include genetic algorithm, artificial immune system, neural networks and others. In this paper we discuss how to using artificial immune system (AIS) approach to solving job-shop scheduling problem. Immunity basically means either natural or acquired resistance to disease. Cells and molecules create immunity from the immune system or medication.

The natural immune system has become an important subject of research recently due to its ability to process a huge amount of information. While the study of the natural immune system was becoming popular, an “imitation” immune system was introduced. The imitation immune system the artificial immune system is a set of techniques with algorithm that imitate the natural immune system, so that its behaviour functions like the natural immune system (see Hart and Timmis [5]). These techniques are commonly used in pattern recognition, detection of defects, diagnosis, and other functions, including optimization [6].

AIS can be defined as a computational system based on metaphors borrowed from the biological immune system. To better comprehend the AIS model, a basic understanding of the functioning of the human immune system is essential. The human immune system is characterized by its adaptive and robust nature, and its duty is to protect the human body from infection. For primary immune responses, it launches a response to invading pathogens, and for secondary immune responses, it remembers past encounters to faster responses. This can be explained by considering the simple example of an infection attacking the body. The antigen attacking the body is countered by a defence mechanism called the antibody. The antibody consists of varied combinations of T-cells and B-cells that can adapt themselves to counter the antigen (see Figure 2). By binding to any antigen they find, the antibodies can neutralize it or precipitate its destruction through complement enzymes or scavenging cells. Therefore, a well performing immune system gives the individual a higher chance of survival. AIS is a set of techniques that try to algorithmically mimic a natural immune system’s behaviour [5]. To develop AIS for engineering, we need to consider the following important aspects:

i. Hybrid structures and algorithms that are translated into immune system components;

ii. Algorithm calculations based on the immunology principle, distribution processing, clone selection algorithms, and network theory immunity;

iii. Immunity based on optimization, self-learning, self-organization, artificial life, cognitive models, multi-agent systems, design and scheduling, pattern recognition and anomaly detection;

iv. The immune tools for engineering.

It might be argued that the immune system does not optimize at all, at least not in the manner of the term when solving traditional combinatory or numerical ‘‘optimization’’ problems. Most computational optimization problems have a single goal to obtain; the natural immune system on the other hand can be regarded as having multiple and possibly contradictory goals, and although it does improve its own response towards particular goal(s) as the result of feedback [7] it has no reason to evolve an optimal response. In fact, its network structure does not lead to the development of the best response, but results in the best possible response under existing conditions [8].

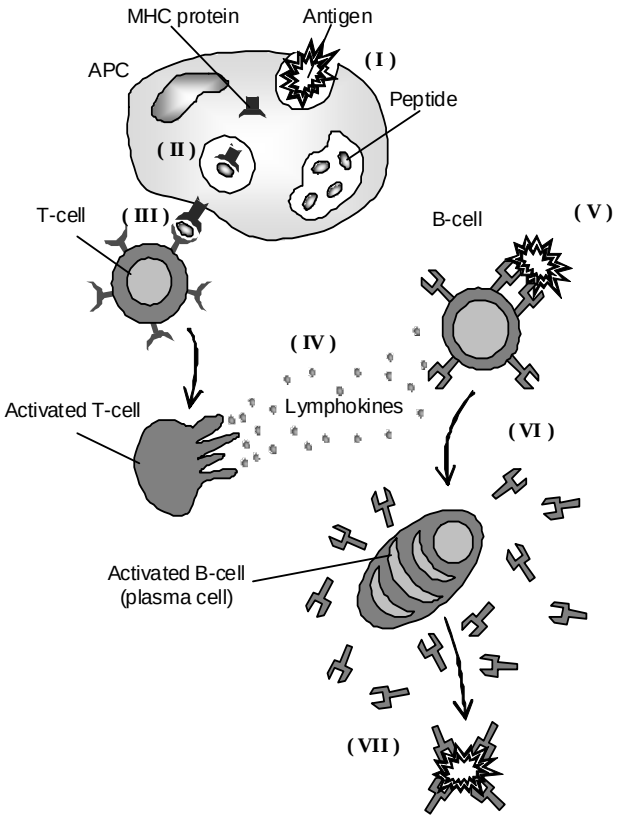

Figure 2: The process of the human immune system

Artificial Immune Systems for Job-Shop Scheduling

Applying the AIS analogy to a shop-floor environment, it would extremely useful to maintain a scheduling system able to produce schedules that can cope with the wide range of potential situations, both predictable and unpredictable. Although in a purely deterministic job-shop, all job-arrival dates and machine processing times are known, it is easy to envisage many practical situations occurring which would require a change in the original schedule for example, a machine breaking down, the due dates of jobs changing due to a customer's new priorities, or jobs arriving later than planned. Much previous work in the job-shop scheduling domain, for example [9], [10], has concentrated on producing optimal schedules that minimize some criterion, for instance turnaround time or job tardiness. However, an optimal schedule may often be extremely fragile: a slight alteration to one or more of the jobs or machine attributes may drastically affect the schedule.

Past research by Hart, Ross and Nelson [11] and Hart and Ross [12], has shown that the AIS model can be used to solve the scheduling problem in the industry environment for real-world situations which require a scheduler to make a new schedule when there are changes, such as the changed environment and that change was unexpected. In another study by Hart and Ross [13], they define antigen as "a sequence of jobs on a particular machine given a particular scenario" and antibody as "a short sequence of jobs that is common to more than one schedule".

In the AIS approach, there are five models that can be used; they include the Bone Marrow Model, Negative Selection Algorithm (NSA), Clonal Selection Algorithm (CSA), Somatic Hyper Mutation and Immune Network Model. For scheduling problem purposes, the suitable models are the Negative Selection Algorithm and Clonal Selection Algorithm. In this paper, we use the CSA approach to solve the JSSP and achieve optimal solutions. Figure 3 illustrates how the Clonal Selection Principle (CSP) works.

The CSP algorithm is based on observations made of B-cells in the natural immune system. The B-cells, along with the T-cells, help combat the various infections and viruses that attack

the human body. The cells merge into different permutations and combinations so as to overcome infections. However, B-cells have been observed to possess a unique tendency that causes the cells to multiply those particular combinations of cells that are capable of destroying the infection attacking the body.

The CSP helps the algorithm presented in this study to be very flexible and rapid in its approach to attaining the best solution. Cloning the solution prior to mutating allows the algorithm to search through a wider range of potential results because more solutions can be mutated simultaneously. The cloning principle also incorporates the virtue of speed into the algorithm.

Based on the CSP function, we derive a model to solve the JSSP; and from this model a modification was made to minimize the *makespan*. In this model, the job-shop problem is translated into a string, call an antibody, and this antibody is generated randomly to get an adequate population of antibodies. All the antibodies in the population will go through the clone and mutation process to get the antigens. Typically, antigen is the solution and the best antigen is chosen as the final solution before we encode it into the schedule. To find the best antigen, affinity measure was used. For job-shop scheduling problem in this paper, the total completion time as affinity measure. Algorithm 1 illustrates the basic model proposed to solve the FJSSP.

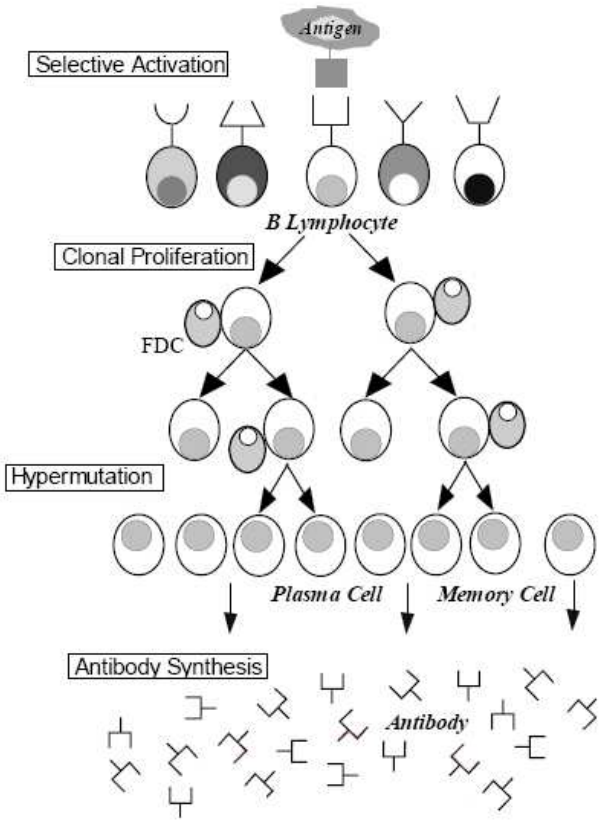

Figure 3: Clonal Selection Principle

procedure *AIS(I, A, C, G, H, m, n)*
    antibody population *p*
    antibody *a*
    antigen *b, c, s*
    integer *d, e, f*

```
f ← 0
if G = 1 then p ← RULE1_ANTIBODY(A, m, n) endif
if G = 2 then p ← RULE2_ANTIBODY(A, m, n) endif
for i ← 1 to I do
    for j ← 1 to A do
        if i = 1and j = 1 then s ← p(j) endif
        for k ← 1 to C do
            a ← p(j)
            b ← MUTATE(a, m, n)
            if k = 1 then d ← AFFINITY(b, m, n) endif
            e ← AFFINITY(b, m, n)
            if k = 1 or e < d then c ← b endif
            if e < d then d ← e endif
        repeat
        p(j) ← c
        if AFFINITY(c) < AFFINITY(s) then s ← c endif
        if AFFINITY(c) = AFFINITY(s) then f ← f + 1
                                     else f ← 0 endif
    repeat
    if f = H then exit endif
repeat
DECODE(s, m, n)
end AIS
```

Algorithm 1: Main function to solve FJSSP using AIS

Step 1 (Generate antibody population)

In the proposed model, the integer string encoding antibody population is generated in two ways, which include (*Rule1*) random processing where jobs and machine are randomly permuted and (*Rule2*) jobs are randomly permuted and machine are permuted from the operation with the global minimum of processing time in the processing time table. The length of each antibody is equal to total number of operations in all jobs, where each job *j* will appear *o* times in an antibody. Based on the problem illustrated in Figure 1, the integer string is generated using *Rule1*, encoding the antibody as Figure 2, while Figure 3 illustrates the antibody generated by using *Rule2*. The gene represent in antibody is *(job, machine)*.

The antibody in the population generated by using *Rule1* or *Rule2* represents the initial solutions for the FJSSP. In the same manner as in biological immune systems, each antibody is separated in two type of genes, a heavy-chain gene (H) and light-chain gene (L). The number of light-chain gene is determined by using the formula in Figure 4.4 and is assigned

from later part of the jobs appearing in the same machine. The others genes, except light-chain genes, are defined as a heavy-chain. For the antibody in Figure 2 and Figure 3, the ratio *R* is 0.3 to calculate the number of light-chain genes in same machine.

Antibody$_i$	(1,1)	(3,3)	(2,2)	(3,1)	(2,3)	(1,2)	(1,2)	(2,1)	(2,2)
Time List	6	3	6	4	7	5	5	7	3
Chain Type	L	L	L	H	L	L	H	H	H

Figure 2: Generate Antibody using *Rule1*

Antibody$_i$	(3,2)	(1,2)	(2,2)	(2,2)	(1,2)	(3,1)	(2,1)	(1,3)	(2,2)
Time List	3	6	6	5	5	4	7	4	3
Chain Type	L	L	L	L	H	L	H	H	H

Figure 3: Generate Antibody using *Rule2*

```
procedure RULE1_ANTIBODY(A, m, n)
   ..
end RAND_ANTIBODY
```

Algorithm 2: Algorithm to generate antibody using *Rule1*

```
procedure RULE2_LIBRARY(A, m, n)
   ..
end GENERATE_LIBRARY
```

Algorithm 3: Algorithm to generate antibody using *Rule2*

```
procedure LIGHT_CHAIN(R, m)
   integer v
   v ← round[(R * m) + 0.5]
   return v
end LIGHT_CHAIN
```

Algorithm 4: Formula to calculate number of light-chain genes

Step 2 (Clone the antibody and mutate the clone)

For the second stage, all the antibodies in the population will be cloned for some pre-determined number. This clone will be mutated to get the antigen using some type of mutation. For the first iteration, the antigen will be assigned to the best solution. For the next iteration, the clone (after mutation) will be compared with the current solution to determine which one is better. If the clone is better than the current solution, that clone will be assigned as the current solution. Before proceeding to the next iteration, the antibody population will be updated. The role of the mutation type is very important in influencing the final solution. In the proposed model the mutation type includes:

i) Random Somatic Point Mutation and Heavy-Light Somatic Point Mutation

For random somatic point mutation, two genes will be randomly chosen from the antibody and those genes will be swapped. For heavy-light somatic point mutations, they are implemented based on the chain type value, where two genes are chosen from the antibody with the heavy value and light value, and those genes will be swapped [14]. Figures 11 and 12 illustrate how the random somatic point mutation and heavy-light somatic mutation function.

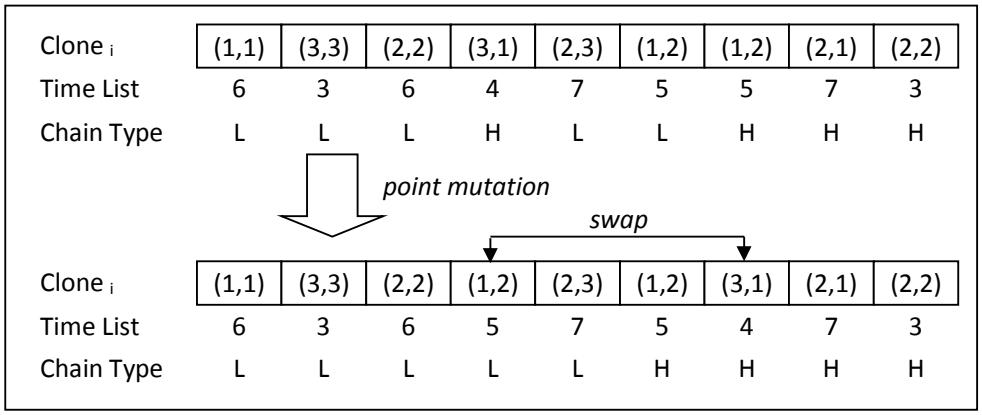

Figure 6: Random somatic point mutation

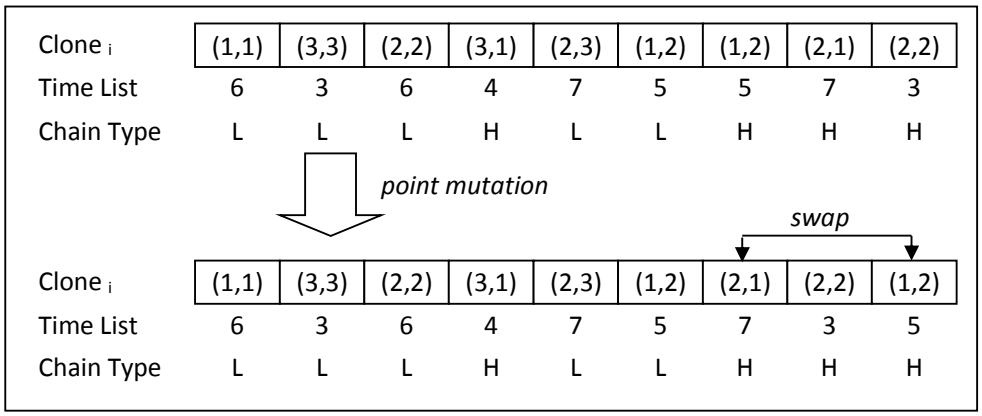

Figure 7: Heavy-light somatic point mutation

ii) Random Somatic Point Recombination

Random somatic point recombination is randomly chose two gene fragments of the same length from the antibody. Following this, a partial exchange is performed between the two chosen fragments [14]. Figure 8 illustrate how the random somatic point recombination function.

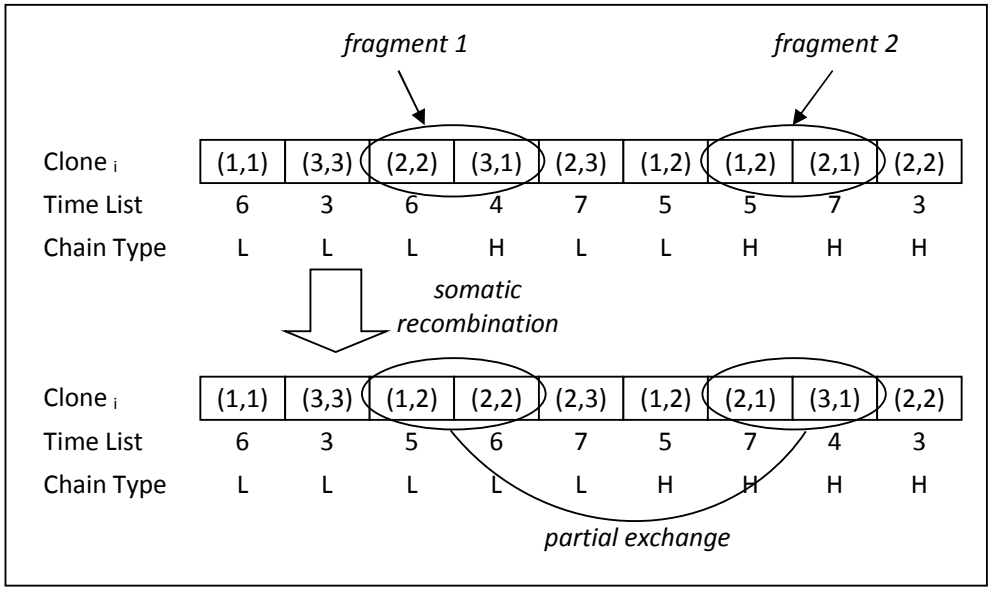

Figure 8: Random somatic point recombination

iii) Gene Conversion

Gene conversion is performed by choosing genes randomly and generating them at random. The lengths of the gene fragments are predetermined and those genes between the starting and ending site are swapped with the other genes again, randomly. Figure 9 illustrates how the gene conversion functions.

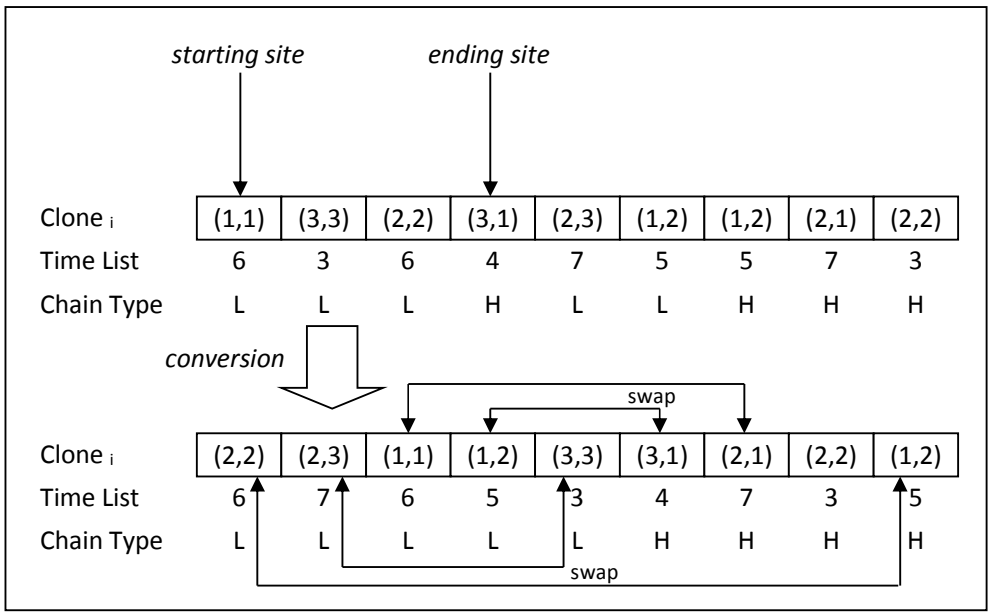

Figure 9: Gene conversion

iv) Gene Inversion

Gene inversion also functions randomly by randomly generating the selected gene at the starting site. For this mutation, the gene fragment is inversed from front to rear and from rear to front. Figure 10 illustrates how the gene inversion functions.

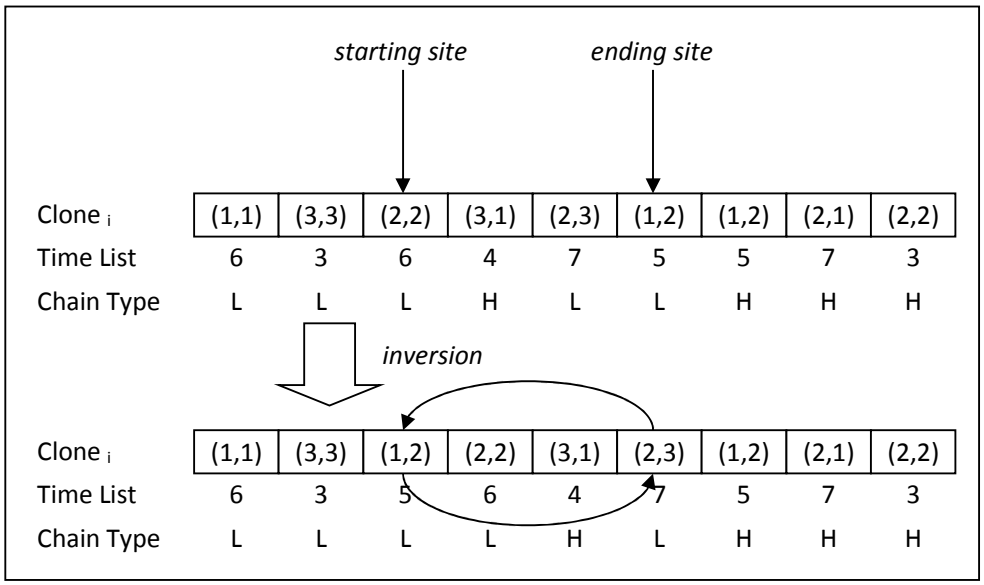

Figure 10: Gene inversion

v) Gene Right-Shift and Gene Left-Shift

Gene right-shift and gene left-shift randomly choose the starting and ending points and the number of shifted genes are predefined. The selected gene fragments exchange their locations to the right and the left.

vi) Nucleotide Addition

Nucleotide addition will randomly generate the gene fragment of a predetermined length and randomly select the location where this fragment is to be inserted into the antibody. Displaced genes are then shifted to the right with excessive genes removed and the antibody boundary repaired.

In the proposed model, only one of eight of the mutation types is chosen for the mutation process of each clone. In addition, of course, the mutation type will be chosen at random.

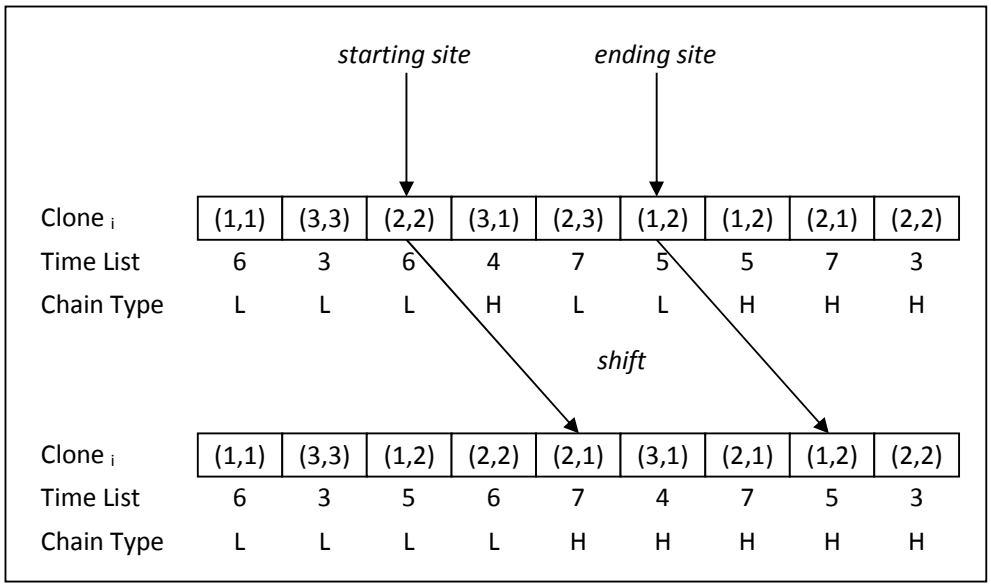

Figure 11: Gene right-shift

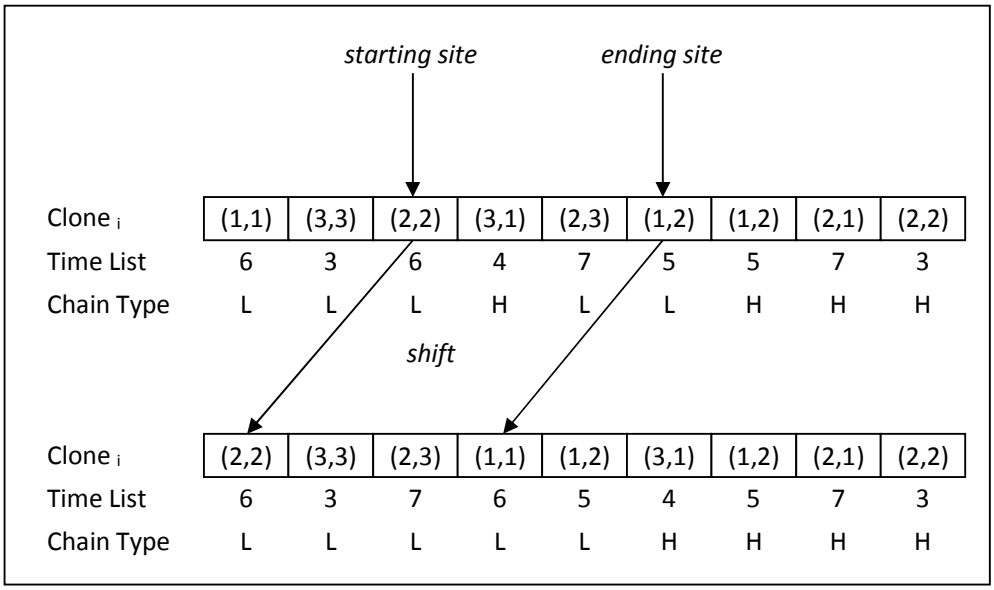

Figure 12: Gene left-shift

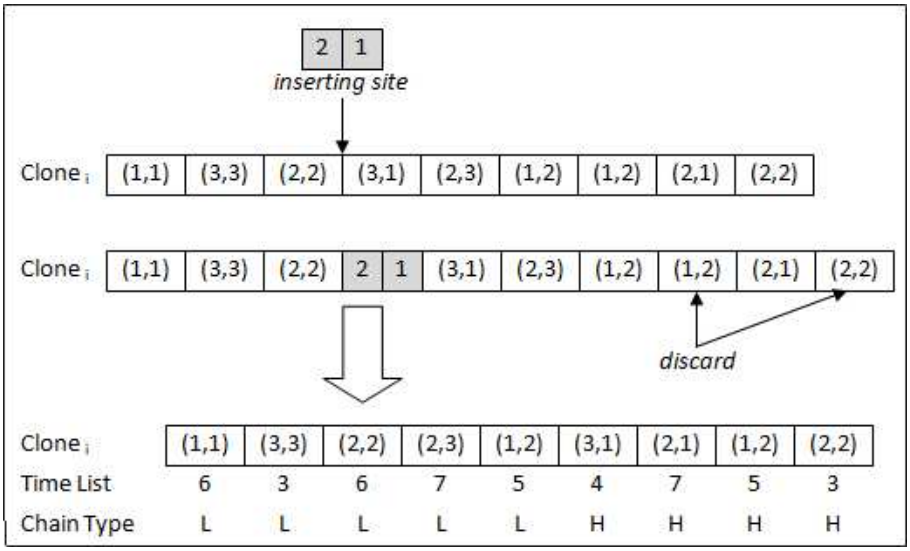

Figure 13: Nucleotide addition

Step 3 (Calculate the affinity)

After mutating the clone, the affinity (*makespan*) will be calculated. This calculation is important in determining the best clone to update the antibody population. The *makespan* is obtained by using a decoding procedure, where the first gene is scheduled for decoding first, then the second, and so on, until all the genes have been decoded.

```
procedure AFFINITY(b, m, n)
    solution s
    machine a
    integer t
    s ← DECODE(b, m, n)
    t ← 0 // assign default value for makespan
    for i ← 1 to m do
        a ← s(i)
        t ← max[t, a(n).finish]
    repeat
    return t
end AFFINITY
```

Algorithm 6: Calculate Affinity Value

```
procedure DECODE(a, m, n)
    solution s
    job j
    for i ← 1 to m do
        s(i).start ← 0 // set start time for each machine
    repeat
    for i ← 1 to n do
        j(i).start ← 0 // set start time for each job
    repeat
    for i ← 1 to m * n do
        s(a(i).machine).job ← a(i).job
        s(a(i).machine).time ← a(i).time
        s(a(i).machine).start ← max[s(a(i).machine).start, j(a(i).job).start]
        s(a(i).machine).finish ← s(a(i).machine).start + a(i).time
        s(a(i).machine).start ← s(a(i).machine).finish
        j(a(i).job).start ← s(a(i).machine).finish
    repeat
    return s
end DECODE
```

Algorithm 7: Decoding Process

Step 4 (Update the antibody population)

For each iteration, the population of antibodies will be updated by replacing the best clone with an antibody. For the first iteration, the antibody population was randomly generated or using the library, while for the second to last iteration, the population of antibodies will use the updated antibody. Let us say that an antibody, as in Figure 2, was cloned and the first clone choose the random somatic point recombination as its mutation type, second clone choose the random somatic point mutation as its mutation type, then the schedule will be as illustrated in Figure 14 and Figure 15. In this case second clone (after mutation) will be chosen to replace the antibody.

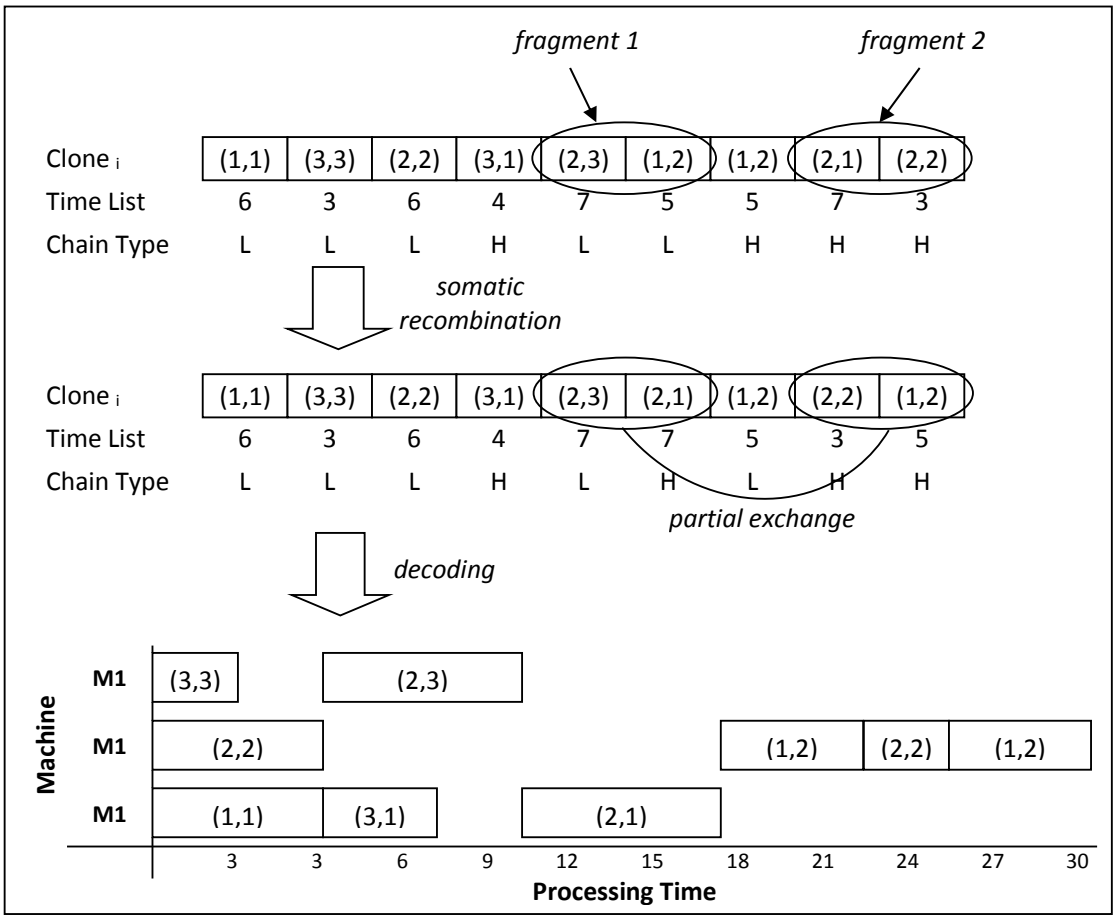

Figure 14: Mutate using Random Somatic Point Recombination

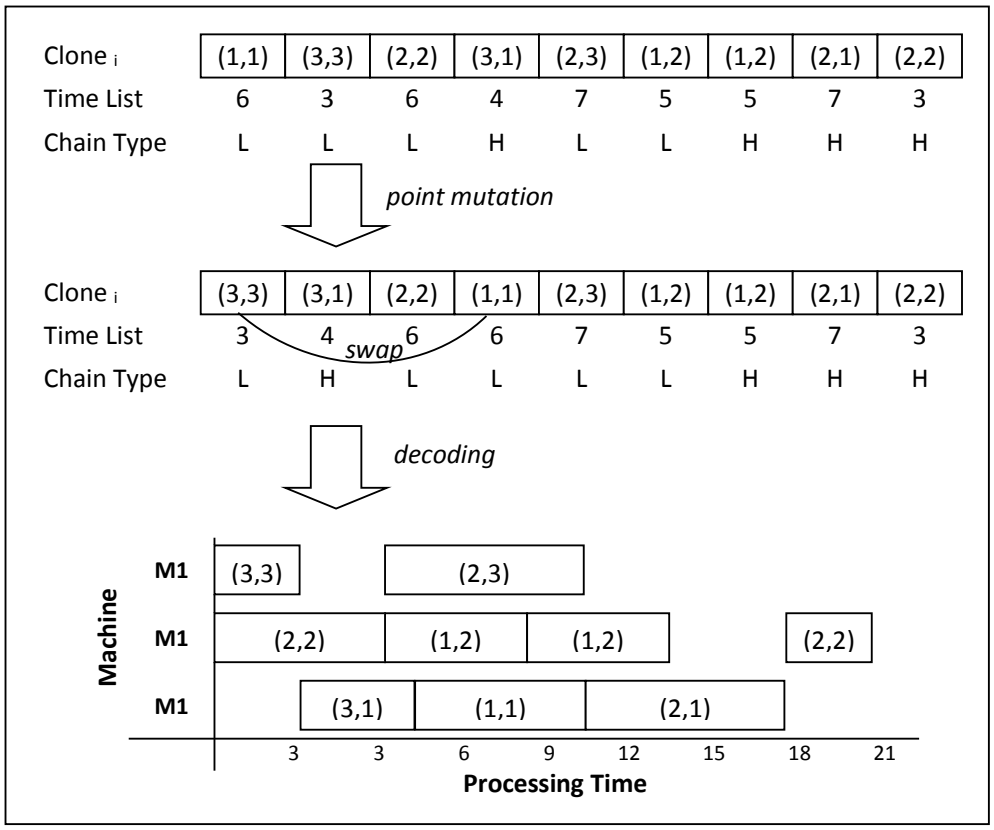

Figure 15: Mutate using Random Somatic Point Mutation

Step 5 (Stopping criterion)

The whole process stops when the iteration equals a predefined number. Otherwise the process reverts to Step 2 for another cloning. Finally, the best solution will be decoded to obtain the schedule.

Computational Result

Using the proposed model, an experiment was run using different sizes of the flexible job-shop problem. In this paper, we use 178 instances that are taken from six classes of FJSSP problems: mk01 – mk10 are taken form Brandimarte [15], eighteen problems from Dauzére-Pérés and Paulli [16], twenty one problems from Barnes and Chambers [17], and one hundred twenty nine problems Hurink [18]. To carry out the necessary computations and evaluate the performance of the proposed model, a computer program was developed using Java technology adapted to the proposed model. The parameter setting for this experiment is shown in Table 1. The results of this experiment were compared with the results from an experiment employed in previous research. Table 2 shows the comparison of the results

between the present and previous research for mk01 – mk10 problems. The contents of the table included the name of each test problem (Inst.), the scale of the problem (Size), the value of the best known solution for each problem (BKS), the value of the best solution found by using the proposed algorithm, the percentage of the deviation with respect to the best known solution (RD%), and the best result reported in other research works. Mean relative error (MRE) over best-known lower bound for all problems shows on Table 3. From this comparison it can be seen that the results of the experiment on each problem are as good as the best known results.

Table 1: The parameter setting

Parameter	**Value**
Population	*nm*
Iteration	*nm*
Light chain ratio	0.3 to 0.4
Length of gene fragment	3 to 5
Number of gene shift	2 to 3
Number of nucleotide	3 to 4

Table 2: The computational result comparison for mk01 – mk10

Inst.	**Size (*n* x *m*)**	**BKS**	**AIS***	**RD%**	**TS [19]**	**GA [20]**	**GA [21]**
mk01	10 x 6	36	49		40	40	40
mk02	10 x 6	24	43		26	26	29
mk03	15 x 8	204	242		204	204	204
mk04	15 x 8	48			60	60	63
mk05	15 x 4	168			173	173	181
mk06	10 x 15	33			58	63	60
mk07	20 x 5	133			144	139	148
mk08	20 x 10	523			523	523	523
mk09	20 x 10	299			307	311	308
mk10	20 x 15	165			198	212	212
* Proposed algorithm							

(Continued) **Table 2:** The computational result comparison for mk01 – mk10

Inst.	**Size (*n* x *m*)**	**BKS**	**GA [22]**	**GENACE [23]**	**ClonaFLEX [24]**	**AIA [25]**
mk01	10 x 6	36	40	41	39	40
mk02	10 x 6	24	28	29	27	26
mk03	15 x 8	204	204	204	-	204
mk04	15 x 8	48	61	67	65	60
mk05	15 x 4	168	176	176	173	173
mk06	10 x 15	33	62	68	70	63
mk07	20 x 5	133	145	148	145	140
mk08	20 x 10	523	523	523	523	523
mk09	20 x 10	299	310	328	311	312
mk10	20 x 15	165	216	231	-	214

Table 3: Mean relative error (MRE) over best-known lower bound

Data set	**Num.**	**AIS***	**GA [20]**	**GA [21]**	**GA [22]**	**AIA [25]**
Brandimarte	10		17.53%	19.55%	19.11%	17.76%
Dauzére-Pérés and Paulli	18		7.63%	7.91%	10.62%	-
Barnes and Chambers	21		29.56%	38.64%	29.75%	-
Hurink EData	43		6.00%	5.59%	9.01%	6.83%
Hurink RData	43		4.42%	4.41%	8.34%	3.98%
Hurink VData	43		2.04%	2.59%	3.24%	1.29%
* Proposed algorithm						

Figure 15: Percentage of the deviation with respect to the BKS

References

Bruker, P., Schlie, R.. 1990. Job Shop Scheduling with Multi-Purpose Machine, Computing, Vol. 45, 1990, pp. 369 – 375.

French, S. 1982. Sequencing and Scheduling, Mathematics and its Applications. Ellis Horwood Limited.

Albert Jones and Luis C. Rabelo. 1998. Survey of Job Shop Scheduling Techniques. NISTIR, National Institute of Standards and Technology, 1998.

A. S. Muhamad and S. Deris. 2011. An Artificial Immune System for Solving Production Scheduling Problems: A Review. Artificial Intelligent Review, 2011, DOI: 10.1007/s10462-011-9259-1, Online First 3 June 2011.

Hart, E., and Timmis, J., 2008. Application areas of AIS: the past, the present and the future. Applied Soft Computing, 8(2008): 191-201.

Costa, A.M., Vargas, P.A., Von Zuben, F.J., Frabca, P.M., 2002. Makespan Minimization on Parallel Processors: An Immune-Based Approach. Proceedings of the 2002 Congress on Evolutionary Computation, IEEE Press, 920-925.

Segel , L.A, 2001. Diffuse feedback from diffuse informational network: in the immune system and another distributed autonomous systems, in: L.A. Segel, I. Cohen (Eds), Design Principles for the Immune Systems and other Distributed Systems, Oxford University Press, 203-226.

Orosz , G.C, 2001. An introduction to immuno-ecology and immuno-informatics, in: L.A. Segel, I. Cohen (Eds), Design Principles for the Immune Systems and other Distributed Systems, Oxford University Press, 125-150.

Ayara , M, J, de Lemos, T., R., and Forrest, S, 2005. Immunising Automated Teller Machines. Proceedings of the 4th International Conference in Artificial Immune Systems (ICARIS 2005), Banff, Canada, Lecture Notes in Computer Science 3627, Springer. Berlin, Germany.

Bersini , H. Immune Network and Adaptive Control. Proceedings of the first European Conference on Artificial Life, Paul Bourgine and Francisco Varela (Eds.), Bradford Books - MIT Press.

Hart, E., Ross, P., and Nelson, J., 1998. Producing Robust Schedules Via An Artificial Immune System. Proceeding of the ICEC '98, IEEE Press, 464-469.

Hart, E., and Ross, P., 1999. An Immune System Approach to Scheduling in Changing Environments. Proceeding of the GECCO 1999, W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela and R.E. Smith (Eds.), Morgan Kaufmann, 1559-1565.

Hart, E., and Ross, P., 1999. The Evolution and Analysis of a Potential Antibody Library for Job-Shop Scheduling. New Ideas in Optimisation. D. Corne, M. Dorigo & F. Glover (Eds.), McGraw-Hill, London, 185-202.

Chueh, C.H., 2004. An Immune Algorithm for Engineering Optimization. Tatung University, Ph.D. Thesis.

[15] Brandimarte P. 1993. Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 1993, 41, 157–83.

Dauzére-Pérés S, Paulli J. 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Annals of Operations Research 1997, 70, 281–306.

Barnes JW, Chambers JB. 1996. Flexible Job Shop Scheduling by tabu search. Graduate program in operations research and industrial engineering. Technical Report ORP 9609, University of Texas, Austin.

Hurink J, Jurish B, Thole M. 1994. Tabu search for the job shop scheduling problem with multi-purpose machines. OR-Spektrum 1994, 15, 205–15.

Mastrolilli M, Gambardella LM. 1996. Effective neighbourhood functions for the flexible job shop problem. Journal of Scheduling 1996, 3, 3–20.

F. Pezzellaa, G. Morgantia, G. Ciaschettib. 2007. A genetic algorithm for the Flexible Job-shop Scheduling Problem. Computers and Operations Research (2007), doi: 10.1016/j.cor.2007.02.014.

Chen H, IhlowJ, Lehmann C. 1999. Agenetic algorithm for flexible Job-shop scheduling. In: IEEE international conference on robotics and automation, Detroit, 1120 – 1125.

Jia HZ, Nee AYC, Fuh JYH, Zhang YF. 2003. A modified genetic algorithm for distributed scheduling problems. International Journal of Intelligent Manufacturing 2003, 14, 351 –362.

Ho NB, Tay JC. 2004. GENACE: an efficient cultural algorithm for solving the Flexible Job-Shop Problem. IEEE international conference on robotics and automation 2004, 1759 – 1766.

Z.X. Ong, J.C. Tay, C.K. Kwoh. 2005. Applying the Clonal Selection Principle to Find Flexible Job-Shop Schedules, C. Jacob et al. (Eds.): ICARIS 2005, LNCS 3627, 442 – 455.

A. Bagheri, M. Zandieh, Iraj Mahdavia, M. Yazdani. 2010. An artificial immune algorithm for the flexible job-shop scheduling problem, Future Generation Computer Systems 26 (2010), 533 – 541.