International Journal of Intelligent Information Processing
Volume 1, Number 2, December 2010

Solving Job Shop Scheduling Problem Using Genetic Algorithm with Penalty Function

Liang Sun [1, 3], Xiaochun Cheng [2], Yanchun Liang [1*]
[1] *College of Computer Science and Technology, Jilin University, Changchun, 130012, China*
[2] *School of Computing Science, Middlesex University, London, UK*
[3] *School of Information, Kochi University of Technology, Kochi, 782-8502, Japan*
[*] *Corresponding author email: ycliang@jlu.edu.cn*
doi:10.4156/ijiip.vol1. issue2.7

Abstract

This paper presents a genetic algorithm with a penalty function for the job shop scheduling problem. In the context of proposed algorithm, a clonal selection based hyper mutation and a life span extended strategy is designed. During the search process, an adaptive penalty function is designed so that the algorithm can search in both feasible and infeasible regions of the solution space. Simulated experiments were conducted on 23 benchmark instances taken from the OR-library. The results show the effectiveness of the proposed algorithm.

Keywords: *Job Shop Scheduling Problem, Genetic Algorithm, Penalty Function*

1. Introduction

The job shop scheduling problem (JSSP) is one of the most well-known problems in both fields of production management and combinatorial optimization. The classical n-by-m JSSP studied in this paper can be described as follows: scheduling n jobs on m machines with the objective to minimize the completion time for processing all jobs. Each job consists of m operations with predetermined processing sequence on specified machines and each operation of the n jobs needs an uninterrupted processing time with given length. Operations of the same job cannot be processed concurrently and each job must be processed on each machine exactly once.

Efficient methods for solving JSSP are important for increasing production efficiency, reducing cost and improving product quality. Moreover, JSSP is acknowledged as one of the most challenging NP-hard problems [1] and there is no any exact algorithm can be employed to solve JSSP consistently even when the problem scale is small. So it has drawn the attention of researches because of its theoretical, computational, and empirical significance since it was introduced. Due to the complexity of JSSP, exact techniques, such as branch and bound [2, 3], and dynamic programming [4, 5] are only applicable to modest scale problems. Most of them fail to obtain good solutions solving large scale problems because of the huge memory and lengthy computational time required. On the other hand, heuristic methods, include dispatching priority rules [6-8], shifting bottleneck approach [9-11] and Lagrangian relaxation [12-14], are attractive alternatives for large scale problems. With the emergence of new techniques from the field of artificial intelligence, much attention has been devoted to meta-heuristics. One main class of meta-heuristics is the construction and improvement heuristic, such as tabu search [15-17] and simulated annealing [18, 19]. Another main class of meta-heuristic is the population based heuristic. Successful examples of population based algorithms include genetic algorithm (GA) [20-22], particle swarm optimization (PSO) [23, 24], artificial immune system and their hybrids [25-27], and so on.

Among the above methods, GA, proposed by John Holland [28, 29], is regarded as problem independent approach and is well suited to dealing with hard combinational problems. GAs use the basic Darwinian mechanism of “survival of the fittest” and repeatedly utilize the information contained in the solution population to generate new solutions with better performance. Classical GAs use binary strings to represent potential solutions. One main problem in classical GA is that binary strings are not naturally suited for JSSP. Another problem in classical GAs is premature convergence. Although GAs have better performance than most of conventional methods, they could not guarantee to resist premature convergence when individuals in the population are not well initialized. From the view point of the JSSP itself, it is a hard combinatorial optimization problem with constraints. The goal of the scheduling methods is to find a solution that satisfies the constraints. However, some of the infeasible solutions are of similarity with the feasible optimal solutions, and may provide useful information for generating optimal solution. If we search only within the feasible regions, the generation of the optimal schedule not only requires long time but also decreases the possibility to obtain the optimal solution. Motivated by these perspectives, we proposed a novel genetic algorithm which adopts an operation based representation and a novel search scheme. The proposed genetic algorithm searches in both feasible and infeasible regions. In the proposed algorithm, the potential solutions are generated without considering the constraints. To handle the infeasible solutions, we proposed an adaptive penalty function to impose penalties on the evaluation functions.

The remainder of this paper is organized as follows. Section 2 briefly introduces the genetic algorithms, and then proposes the modified genetic algorithm. Section 3 first introduces the basic penalty function, and then proposes the dynamic and adaptive penalty function. In Section 4, the implementation of proposed algorithm is presented. In Section 5, the corresponding computational and comparative results are given. Finally, conclusion remarks are given in Section 6.

2. The modified genetic algorithm

2.1 Classical genetic algorithm

Genetic algorithm is a stochastic searching method with the features of generating and testing. It works on a randomly generated candidate solution pool, which is usually called "population". Each encoded candidate solution is called "chromosome". During the searching process, the selection, crossover and mutation operators are executed repeatedly until the stop criteria is satisfied. The flow chart of the classical genetic algorithm is shown in Fig. 1.

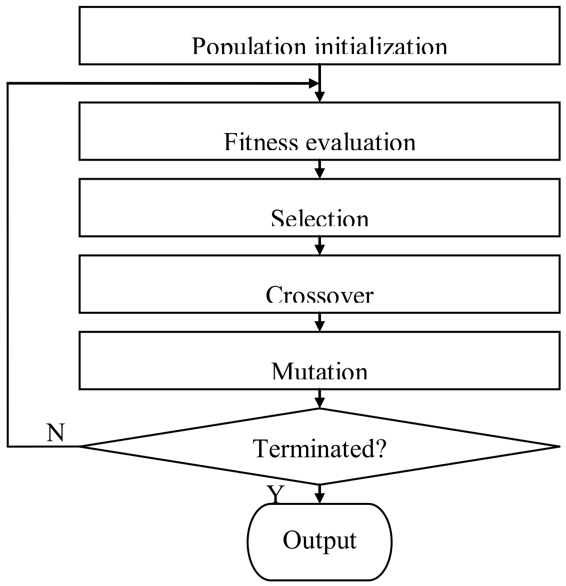

Figure 1. Flow chart of the classical genetic algorithm.

The quality of the chromosome is evaluated by the fitness value. Generally speaking, the chromosomes with higher fitness value have higher probability to generate child chromosomes. Thus, the selection rate is in accordance with the fitness value of each chromosome. However, if there is no improvement of the highest fitness degree for a certain number of generations, it means that the selected parent chromosomes cannot generate better individuals. But if we still select parent chromosomes according to the fitness value for crossover and mutation operations, the chromosomes with similar scheme will be added to the population again and again, and the pseudo-optimal chromosomes will occupy the whole population at last.

Let's examine the crossover and mutation operator. Crossover exchange parts of genes of two parent chromosomes to generate new chromosomes. By combining two parent chromosomes' features, the child chromosomes may get higher fitness value. The mutation operator changes genes in a variable region of the chromosome. The mutation operator keeps the diversity of the population and introduces features that are not present in the current population, therefore prevent premature convergence. But the crossover and mutation operators cannot work well when the pseudo-optimal individual dominates the whole population. Crossover operator uses parent chromosomes to produce child chromosomes, thus the child chromosomes have the same schemes with their parents. Mutation operator is a random process, if we perform high mutation rate, most of the chromosomes may become non-function or develop into harmful specificities; on the other hand, if we perform low mutation rate, the new introduced chromosome will soon be eliminated by the selection strategy because of the domination of the pseudo-optimal chromosomes.

2.2. Modified genetic algorithm

Based on the above discussions, the classical genetic algorithm cannot escape premature convergence consistently if the pseudo-optimal individual dominates the whole population. In this paper, we address two operations, one is the clonal selection based hypermutation, and the other is the lifespan extended strategy. The idea of these two operations is based on two observations. Firstly, to prevent premature convergence when the pseudo-optimal chromosomes dominate the whole population, chromosomes with high diversity are needed. Secondly, the new introduced chromosomes should have high diversity, but if they cannot compete with the pseudo-optimal chromosomes, they will be eliminated soon, thus, the diversified chromosomes should also have high fitness values.

2.2.1. Clonal selection based hyper mutation

According to the theory of clonal selection [30, 31], when animals are exposed to antigens, some of its bone marrow derived a kind of cells, named B lymphocytes. The B lymphocytes can recognize the antigen with certain affinity. These B lymphocytes will be stimulated to proliferate and mature into terminal antibody secreting cells. Proliferation of B lymphocytes is an asexual and amitotic process, which creates a set of clones identical to the parent B lymphocytes. And the proliferation rate is in direct proportion to affinity, i.e., the higher affinity, the higher proliferation rate. During proliferation, the clones undergo a hyper mutation, which diversifies the repertoire of the antigen-activated B cells. The receptor edit guides the process of proliferation hyper mutation which results in the B cells with high affinity survived.

Motivated by the clonal selection theory, the proposed clonal selection based hyper mutation operation can be described as follows. First, setup a clonal library, and then the elite chromosomes in the population are copied to the clonal library. After this, the elite chromosomes repeatedly reproduce themselves until the clonal library is full. Then the chromosomes in the clonal library undergo a high rate mutation. At last, replace the worst chromosomes in the population by the best chromosomes in the clonal library. By the clonal selection based hyper mutation, the new generated chromosomes either have higher fitness value in competence with the pseudo-optimal and higher diversity. Thus the algorithm has a higher probability to escape from the pseudo-optimal trap.

2.2.2. Lifespan extended strategy

The process of lifespan extending strategy can be achieved by a pop_size dimension vector. The i-th element of the vector records the survival time of the i-th chromosomes in the population. The update strategy only replaces the individuals that survived longer than a fix time period. By the lifespan extending strategy, the new introduced chromosomes can not be eliminated soon after introduced. During the extended lifespan, they mature and can compete with other pseudo-optimal chromosomes.

2.2.3. Outline of the modified genetic algorithm for JSSP

As described above, the general outline of the modified genetic algorithm for JSSP can be summarized as follows.

Step 1: Generate *pop_size* chromosomes as an initial population, where *pop_size* denotes the population size. Set *iter=0*, set up an empty clonal library *G*, and a *pop_size* dimension vector *life*.
Step 2: Set *iter*=*iter*+1. Calculate fitness value *f* of each individual chromosome.
Step 3: Select *c* pairs of chromosomes by direct selection strategy, then generate *c* pairs of child chromosomes using crossover operator, add these *c* pairs of child chromosomes to the clonal library.
Step 4: Select *s* chromosomes by direct selection strategy. Copy these *s* chromosomes to the clonal library.
Step 5: Reproduce chromosomes in clonal library until the clonal library size reaches *clonal_size*.
Step 6: Select *m* chromosomes in the clonal library, then generate *m* child chromosomes using mutation operator, add these *m* child chromosomes to the clonal library.
Step 7: Rearrange the chromosomes whose lives are longer than a fix time period in the population. Replace these worst *k* chromosomes by the best *k* chromosomes in the clonal library.
Step 8: Stop, if the termination criterion is satisfied, else, clear the clonal library *G,* update vector *life*, go to step 2.

In the above algorithm, the individuals in the clonal library undergo a high rate hyper mutation, so the parameter m is set much higher than the classical genetic algorithm.

3. Constraints in JSSP and penalty methods

3.1. Constraints in JSSP

As described in Section 1, the JSSP is a problem with tight constraints. One constraint is that an operation must be processed after all of its precedent operations finished, namely technique constraint. The other constraint is that one machine can only handle exactly one job at a time, namely resource constraint.

Generally, most genetic methods for JSSP only keep feasible solutions in their populations. There are three possible ways to preserve the feasibility [32]. One is to reject the infeasible solutions without consideration. The second is to repair the infeasible solutions to make them feasible. The third approach is to improve the operators so that they are guaranteed to produce feasible solution. However, these modified methods are disadvantageous somewhat. The modified method is often problem dependence that is computationally intensive. The operator improving methods consider no point outside the feasible region. They can be serious limitation if the search space contains large number of infeasible solutions and they can also be time consuming. Besides these disadvantageous, both of them need to be tailored according to the problem in hand. In addition, according to the scheme theorem and the building block hypothesis of Holland [28, 29], the schemes which have low schema order, short defining length and high fitness will grow in exponential progression, and will eventually mature into high schema order, long defining length and high fitness scheme, thus generate global optimal solution. If we search only within feasible regions, many of the low schema order, short defining length and high fitness schemes contained in infeasible solutions will be lost. Therefore, the optimum solution may take a longer time to be found or cannot be found.

3.2. The penalty method for constraints in JSSP

An optimization problem with constraints can be described as follows [33].

$$\begin{array}{ll} \text{Find} & \vec{x}, \vec{x} = (x_1, x_2, \cdots, x_n) \in F \subseteq S \\ \text{Such that} & f(\vec{x}) \to \min \\ \text{Subject to} & g_i(\vec{x}) \le 0, \quad i = 1, \cdots, n_C \end{array} \tag{1}$$

where $\vec{x}$ is a design vector, $f(\vec{x})$ is the objective function on the search space S, F is the feasible region defined by the set of constraints $g_i(\vec{x}) \le 0$, and n_C is the number of constraints.

To solve such problem, rather than the methods search only within the feasible solution space, the penalty method expands the search space to the infeasible solution regions by a penalty function. If a solution violates the constraints, a penalty is applied to worsen its objective function. A typical exterior penalty function has the following form [33].

$$\widetilde{F}(\vec{x}) = F(\vec{x}) + P(\vec{x}) \tag{2}$$

$$\begin{gathered} P(\vec{x}) = \sum_{i=1}^{n_C} r_i < g_i(\vec{x}) >^2 \\ < g_i(\vec{x}) > = \begin{cases} 0 & \text{if } g_i(\vec{x}) < 0 \\ g_i(\vec{x}) & \text{if } g_i(\vec{x}) \ge 0 \end{cases} \end{gathered} \tag{3}$$

where $\widetilde{F}(\vec{x})$ is the penalized fitness function, $F(\vec{x})$ is the fitness function and $P(\vec{x})$ is the penalty function, and r_i is the penalty factor for the i-th constraint. To guide the search towards promising area of solution space, the penalty function $P(\vec{x})$ should be deliberately designed.

The design of $P(\vec{x})$ depends on many factors such as the solution itself, the search history, and the other solutions, etc. To handle the constraints in JSSP, we should consider the following factors:

1. The extent to which the solution violating constraints;
2. The search history of the genetic algorithm;
3. The proportion between feasible and infeasible solutions in the population;
4. The proportion between fitness function $F(\vec{x})$ and penalty function $P(\vec{x})$.

Based on the considerations above, the penalty function proposed in this paper is described as follows

$$P(\vec{x}) = v \cdot k^{\alpha} \cdot (\frac{n_{infeasible}}{n})^{\beta} \cdot \sum_{i=1}^{n_C} w_i < g_i(\vec{x}) >^2 \tag{4}$$

where v is a constant that controls the proportion between $F(\vec{x})$ and $P(\vec{x})$, k is the iteration number, $n_{\text{infeasible}}$ is the number of infeasible solutions, n is the number of solutions, function $< g_i(\vec{x}) >$ denotes the degree of $\vec{x}$ violating the i-th constraint, w_i is the factor denoting the importance of constraint i, and α and β are constants, respectively. From the above function we can see that the penalty imposed on infeasible individuals is increased with the generation number *k*. The proportion between infeasible and feasible solutions is balanced by $\frac{n_{\text{infeasible}}}{\text{n}}$ to avoid the infeasible solutions dominating the population.

4. Implementation of modified genetic algorithm with penalty function for JSSP

4.1. Representation

Before solving the JSSP, we need to describe a proper representation for the solution of the problem, namely a scheduling, which is used in the proposed algorithms. In this paper, we adopt an operation-based representation method. For an n-job and m-machine problem, there are $n \times m$ operations in all, so a $n \times m$ list is used to represent these operations. Each element of the list contains a 3-dimension vector <*j*, *m*, *d*>, where *j* represents the job index, *m* represents the machine index and *d* represents the duration of job *j* processed on machine *m*. Thus a chromosome of a candidate solution can be represented by a permutation of the operation indexes $[1, 2, \cdots, n \times m]$. All possible permutations of the integers compose the solution space.

A complete scheduling (both feasible and infeasible) for a problem can be obtained by the process called decoding. By scanning the chromosome from left to right, we first obtain the operation index, and then obtain the exact operation information by checking operation list. Thereafter, we assign the operation to the earliest time slot of the corresponding job and machine. A complete scheduling can be obtained when all the operations in the chromosome are scanned. It is obvious that the resource constraint is easy to be satisfied by assigning operations to time slot when both the job and machine are idle. However, the technique constraints are relatively difficult to satisfy. Therefore, when we decode a chromosome we make sure the resource constraints satisfied and leave the technique constraints unconsidered.

4.2. Initial solution

Empirical study shows that the initial solution methods affect the solution, so that the better initial solution might provide better results. In this paper, a new scheduling initialization algorithm is proposed based on the well-known algorithm of Giffler and Thompson (G&T) [34]. According to G&T, when assigning an operation of a job to a machine, the operation which has short processing time and an early starting time will have high priority to be selected. In this paper, we also take into account the critical operation. If hanging its schedule by moving forward or backward influences the finishing time of all the operations, i.e., influences the makespan, then this operation is called critical operation.

Therefore, the earlier the finishing time of a critical operation appears, the higher the appearance possibility of minimum makespan is. However, it is usually impossible to identify the critical operations before we obtain the complete scheduling solution. So we need to first evaluate each operation and endow the operation identified as a possible critical operation with a higher priority to be scheduled.

4.3. Crossover

One of the important aspects of the technique involved in genetic algorithm is crossover. The process of crossover is designed as follows. First, two parental individuals parent1 and parent2 are selected. A vector of length $j \times m$ for the problem of j jobs and m machines is produced, which is randomly filled with elements of the set {0, 1}. This vector defines the order in which the operations of the newly produced offspring are drawn from parent1 and parent2, respectively. After an operation is drawn from one parent and deleted from the other one, it is appended to the newly produced offspring. This step is repeated until both parent1 and parent2 have been exhausted and the offspring contains all the genes involved. This process is repeated by exchanging parent1 and parent2, and the other offspring is produced.

4.4. Mutation

Mutation is another important operation in the proposed genetic algorithm. It is used to produce perturbations on chromosomes in order to maintain the diversity of population. Mutation process means random changes take place in a region of the chromosome. It is relatively easy to make some mutation operations for permutation representation. In this paper, three types of mutation are used.

1. Inverse mutation. Select two inverse points in the permutation then reverse the sub permutation between the two points. For example, let the selected inverse points in permutation A be 3 and 7, then the mutated permutation A' is shown as follows

$$A = 123 \mid 4567 \mid 89$$

$$A' = 123 \mid 7654 \mid 89$$

2. Interchange mutation. Select two interchange points in the permutation then interchange their values. For example, let the selected interchange points in permutation A be 3 and 8, then the mutated permutation A' is shown as follows

$$A = 12\underline{3}4567\underline{8}9$$

$$A' = 12\underline{8}4567\underline{3}9$$

3. Insert mutation. Select two points in the permutation then insert one after the other. For example, assume that the selected two points in permutation A are 6 and 3, then the mutated permutation A' is shown as follows

$$A = 12\underline{3}45\underline{6}789$$

$$A' = 12\underline{3}\underline{6}45789$$

Of course, some mutation can be obtained is multiple application of a basic mutation. For example, the inverse mutation and interchange mutation can be obtained from multiple application of insert mutation. However, since there isn't any principle for the design of mutations, the three types of mutations in this paper are designed concerning the performance of the algorithm. They are the ones that give the best performance.

4.5. Penalized objective function

According to Eq. (2), the penalized objective function consists of original objective function $F(\vec{x})$ and the penalty function $P(\vec{x})$. In JSSP, the objective is to minimize the makespan, i.e., the final completion time of all the jobs. For an operation permutation $\vec{x}$, we can get a complete scheduling π by decoding process. Thus, the original objective $F(\vec{x})$ can be defined as

$$F(\vec{x}) = \max\{T[1], T[2], \cdots, T[m]\} \tag{5}$$

where $T[i]$ is the final completion time of machining on the i-th machine, and $F(\vec{x})$ is the final completion time of all the operations.

For a JSSP of $n_C \times m$, there are n_C technique orders for these n_C jobs, these n_C technique orders fall into n_C technique constraints. By checking π, we can easily get the operation permutation $\pi(j)$ of each job j. Then we can calculate the violation measurement function by comparing $\pi(j)$ with the technique order of the j-th job. So the constraint violation measurement function $g_j(\vec{x})$ can be described as:

$$g_j(\vec{x}) = \sum_{k=1}^{m} (\delta_{pre}(k) + \delta_{succ}(k))$$
$$\delta_{pre}(k) = \begin{cases} 1 & \text{if operation } k\text{'s precedent operation violates} \\ 0 & \text{else} \end{cases} \tag{6}$$
$$\delta_{succ}(k) = \begin{cases} 1 & \text{if operation } k\text{'s success operation violates} \\ 0 & \text{else} \end{cases}$$

so the penalized objective function can be described as

$$\widetilde{F}(\vec{x}) = F(\vec{x}) + v \cdot k^{\alpha} \cdot (\frac{n_{\text{infeasible}}}{n})^{\beta} \cdot w \cdot \sum_{j=1}^{m} g_j^2(\vec{x}) \tag{7}$$

Different from Eq. (4), the parameter w in the penalty function (7) is identical because the constraints in JSSP are of the same importance.

4.6. Termination criterion

The termination criterion is satisfied if the best solution obtained is proved to be optimal or if the algorithm has performed a given total number of generations.

5. Numerical experiments

In order to verify the validity of the proposed algorithm, the computational simulation is carried out with some well-studied benchmarks. In this paper, 23 problems that were contributed to the OR-Library [35] are selected. The instances FT06, FT10, and FT20 are designed by Fisher and Thompson (1963), and instances LA01 to LA20 are designed by Lawrence (1984). The designers used them to compare performances of some heuristics and found these problems to be particularly difficult. So these problems have been used as benchmarks for study with different methods by many researches.

5.1. Parameters selection

Table 1. Parameter Values

parameter	value
pop_size	50
clonal_size	30
c (crossover)	15
m (mutation)	30
minmum life	20
k (replacement number)	10
α	0.5
β	0.5
w	1.0

The parameters of the proposed algorithm are selected by trial and error. However, the performance of the algorithm varies when solving different instances. If we leave all of these parameters unchanged, they may not perform well. Through thorough testing, we found that a certain parameter set can get good result for instances with the same size. We also found that the proposed algorithm produces quite satisfactory solutions by tuning only the parameter v in Eq. (7) and leave all the other parameters unchanged. In this paper, all the parameters except v were tested and the selected values are those that could result in the best computational results concerning both the quality of the solution and the computational time. Thus, these select parameters are given in Table 1. For the parameter v, we can determine its value by turning from 0.50 to 0.59 with several times of running. And parameter v for different instance size is given in Table 2.

Table 2. Selection of Parameter *v*

instance size	value
6×6	0.50
10×10	0.56
20×5	0.59
10×5	0.53
15×5	0.52
20×5	0.55
15×10	0.54
20×10	0.54
30×10	0.50
15×15	0.51

5.2. Simulation results and comparisons

5.2.1. Effects of penalty function

According to the penalty function (4), the number k of the generations and the number of infeasible solutions in the population $n_{\text{infeasible}}$ are adaptive parameters, and the relationship between them is shown in Figure 2, where $\Lambda = n_{\text{infeasible}}/n$.

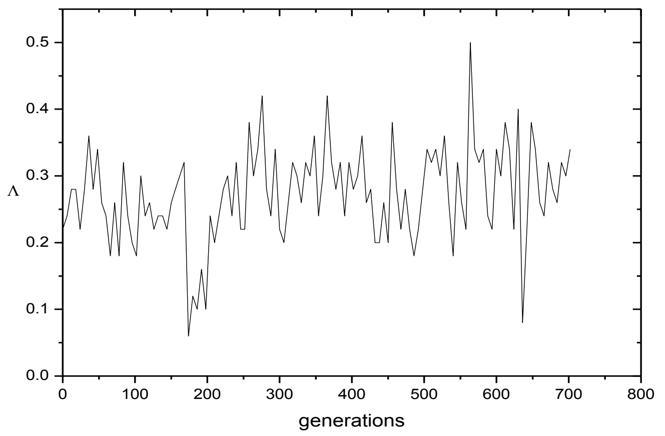

Figure 2. Evolution curve of parameter Λ (test problem: FT10).

From Figure 2, it can be seen that the curve fluctuates at around 0.25, that means the infeasible solutions usually occupy about 25% of the whole population. That is due to parameter Λ makes up a feedback loop during the search process. If the infeasible solutions are too many in the population, then Λ increases. Thus the harsh penalty is imposed on the infeasible solution and the search tends towards the feasible region. Else if the feasible solutions are too many in the population, then Λ decreases. Thus the light penalty is imposed on the infeasible solution and the search moves towards the infeasible region.

According to the objective function (10), the parameter *v* determines the proportion between $F(\vec{x})$ and $P(\vec{x})$. So it affects the explore element as well as the computational time. Table 3 presents the effect of parameter v on searching quality and computational time when solving instance FT10, where the listed items include the name of the test instance (Ins), the scale of the problem (Size), the best known solution for the problem (BKS), the value of parameter v, the best relative error to BKS (BRE), the average relative error to BKS (ARE), the worst relative error to BKS (WRE).

Table 3. Effect of parameter *v* (Test problem: FT10).

Ins	Size	BKS	*v*	BRE(%)	ARE(%)	WRE(%)
FT10	10*10	930	0.50	3.23	4.36	5.32
			0.51	2.69	3.14	6.42
			0.52	1.72	2.43	5.58
			0.53	1.29	1.96	4.89
			0.54	0.75	1.60	3.31
			0.55	0.86	2.24	2.69
			0.56	0.00	1.13	3.27
			0.57	1.18	1.45	4.23
			0.58	1.18	3.69	8.16
			0.59	1.29	2.72	5.54

From Table 3, it can be seen that the performance of the algorithm varies, and the values around 0.56 can produce better results with the turning of the parameter *v*. That is to say, parameter *v* affects the searching quality. We can select the parameter *v* by several times of turning.

5.2.2. Comparisons of the results

Table 4 summarizes the results of the experiments. The contents of the table include the name of each test problem (Ins), 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 (CSGA), the number of running generations (Itr), the percentage of the deviation with respect to the best known solution (RD%), and percentage of the deviation with respect to the best known solution reported in other literatures.

From Table 4 it can be seen that 21 best known solutions can be found among the 23 checked instances by using the proposed algorithm, which accounts for 91.3% of the total instances. And the average deviation of the obtained solutions from the best known solution is only around 0.07%. The proposed algorithm yields a significant improvement in solution quality with respect to other algorithms except the approach proposed by Nowicki. Simulated results show that the proposed approach is a feasible alternative for solving job shop scheduling problems.

6. Conclusions

In this paper we propose a promising genetic algorithm with penalty function for the job shop scheduling problems. Different from traditional genetic algorithms, a clonal selection based hyper mutation and a life span extended strategy is proposed. Moreover, different from other methods, the proposed algorithm search on both feasible and infeasible regions of solution space with the aim of searching for the global optimum solution by an adaptive penalty function. The proposed algorithm effectively exploits the capabilities of distributed and parallel computing of swarm intelligence approaches and effectively makes use of the famous scheme theorem and the building block hypothesis of Holland. The algorithm is tested on a set of 43 benchmark instances. Simulation results are compared with those obtained using other competitive approaches. The results indicate the successful incorporation of the proposed operators. In the future work, we will aim to extend the proposed algorithm to be applied to more practical and integrated problems.

Table 4. Comparisons of the results between CSGA and other approaches

Ins	Size	BKS	CSGA	Itr	RD%	RD%			
						Binato [36]	Nuijten [37]	Coello [26]	Nowicki [16]
Mt06	6×6	55	**55**	2	**0.00**	0.00	0.00	-	0.00
Mt10	10×10	930	**930**	703	**0.00**	0.86	0.00	1.18	0.00
Mt20	20×5	1165	**1165**	2093	**0.00**	0.34	0.00	-	0.00
La01	10×5	666	**666**	7	**0.00**	0.00	0.00	0.00	0.00
La02	10×5	655	**655**	572	**0.00**	0.00	0.15	0.00	0.00
La03	10×5	597	**597**	434	**0.00**	1.17	0.00	0.00	0.00
La04	10×5	590	**590**	134	**0.00**	0.00	0.00	0.00	0.00
La05	10×5	593	**593**	4	**0.00**	0.00	0.00	0.00	0.00
La06	15×5	926	**926**	3	**0.00**	0.00	0.00	0.00	0.00
La07	15×5	890	**890**	9	**0.00**	0.00	0.00	0.00	0.00
La08	15×5	863	**863**	7	**0.00**	0.00	0.00	0.00	0.00
La09	15×5	951	**951**	4	**0.00**	0.00	0.00	0.00	0.00
La10	15×5	958	**958**	6	**0.00**	0.00	0.00	0.00	0.00
La11	20×5	1222	**1222**	10	**0.00**	0.00	0.00	-	0.00
La12	20×5	1039	**1039**	11	**0.00**	0.00	0.00	-	0.00
La13	20×5	1150	**1150**	10	**0.00**	0.00	0.00	-	0.00
La14	20×5	1292	**1292**	2	**0.00**	0.00	0.00	-	0.00
La15	20×5	1207	**1207**	20	**0.00**	0.00	0.00	-	0.00
La16	10×10	945	**945**	1736	**0.00**	0.11	0.21	0.00	0.00
La17	10×10	784	**784**	769	**0.00**	0.00	0.38	0.13	0.00
La18	10×10	848	**848**	635	**0.00**	0.00	0.00	0.00	0.00
La19	10×10	842	**844**	1108	**0.24**	0.00	0.71	0.71	0.00
La20	10×10	902	**907**	2634	**1.33**	0.55	0.55	0.55	0.00

7. Acknowledgement

This work was supported in part by the National Natural Science Foundation of China (10872077), the Science-Technology Development Project from Jilin Province (20080708), and the National High-technology Development Project (2009AA02Z307) of China.

8. References

[1] E. L. Garey, D. S. Johnson, R. Sethi, "The complexity of flow shop and job shop scheduling", Mathematics of Operations Research, 1976, 1:117-129.
[2] P. Brucker, B. Jurisch, B. Sievers, "A branch and bound algorithm for job shop scheduling problem", Discrete Applied Math, vol. 49, pp. 105-127, 1994.
[3] C. Artigues, D. Feillet, "A branch and bound method for the job shop problem with sequence dependent setup times", Annals of Operations Research, vol. 159, no. 1, pp. 135-159, 2008.
[4] T. Lorigeon, "A dynamic programming algorithm for scheduling jobs in a two-machine open shop with an availability constraint", Journal of the Operational Research Society, vol. 53, no. 11, pp. 1239-1246, 2002.
[5] C. N. Potts, L. N. Van Wassonhove, "Dynamic programming and decomposition approaches for the single machine total tardiness problem", European Journal of Operational Research, vol. 32, pp, 405-414, 1987.
[6] M. X. Weng, H. Y. Ren, "An efficient priority rule for scheduling job shops to minimize mean tardiness", IIE transactions, vol. 38, no. 9, pp. 7890795, 2006.
[7] Y. B. Canbolat, E. Gundogar, "Fuzzy priority rule for job shop scheduling", Journal of Intelligent manufacturing, vol. 15, no. 4, pp. 527-533, 2004.

[8] R. Klein, "Bidirectional planning: improving priority rule-based heuristic for scheduling resource-constrained projects", European Journal of Operational Research, vol. 127, no. 3, pp. 619-638. 2000.

[9] J. Gao, M. Gen, L. Y. Sun, "A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems", Computers and Industrial Engineering, vol. 53, no. 1, pp. 149-162, 2007.

[10] L. H. Zhao, F. Q. Deng, "A hybrid shifting bottleneck algorithm for the job shop scheduling problem", Dynamics of Continuous Discrete and Impulsive Systems-Series A-Mathematical Analysis Part 3, vol. 13, pp. 1069-1073, 2006.

[11] F. Pezzella, E. Merelli, "A tabu search method guided by shifting bottleneck for the job shop scheduling problem", European Journal of Operational Research, vol. 120, no. 2, pp. 297-310, 2000.

[12] P. Baptiste, M. Flamini, F. Sourd, "Lagrangian bounds for just-in-time job shop scheduling", Computers and Operations Research, vol. 35, pp. 906-915, 2008.

[13] H. X. Chen, P. B. Luh, "An alternative framework to Lagrangian relaxation approach for job shop scheduling", European Journal of Operational Research, vol. 149, no. 3, pp. 499-512, 2003.

[14] C. A. Kaskavelis, M. C. Caramanis, "Efficient Lagrangian relaxation algorithms for industry size job-shop scheduling problems", IIE Transactions, vol. 30, no. 11, pp. 1085-1097, 1998.

[15] C. Y. Zhang, P. G. Li, Y. Q. Rao, "A very fast TS/SA algorithm for the job shop scheduling problem", Computers & Operations Research, vol. 35, pp. 282-294, 2008.

[16] E. Nowicki, C. Smutnicki, "A fast taboo search algorithm for the job shop scheduling problem", Management Science, vol. 41, no. 6, pp. 113-125, 1996.

[17] A. M. Dell, M. Trubian, "Applying tabu-search to job shop scheduling problem," Annals of Operations Research, vol. 41, no. 3, pp. 231-252, 1993.

[18] T. Y. Wang, K. B. Wu, "A revised simulated annealing algorithm for obtaining the minimum total tardiness in job shop scheduling problems", International Journal of Systems Science, vol. 31, no. 4, pp. 537-542, 2000.

[19] M. Kolonko, "Some new results on simulated annealing applied to the job shop scheduling problem", European Journal of Operational Research, vol. 113, no. 1, pp. 123-136, 1999.

[20] I. Moon, S. Lee, H. Bae, "Genetic algorithms for job shop scheduling problems with alternative routings", International Journal of Production Research, vol. 46, no. 10, pp. 2695-2705, 2008.

[21] J. F. Goncalves, J. J. D. M. Mendes, M. G. C. Resende, "A hybrid genetic algorithm for the job shop scheduling problem", European Journal of Operational Research, vol. 167, no. 1, pp. 77-95, 2005.

[22] C. Bierwirth, D. C. Mattfeld, "Production scheduling and rescheduling with genetic algorithms", Evolutionary Computation, vol. 7, no. 1, pp. 1-17, 1999.

[23] B. Liu, L. Wang, Y. H. Jin, "An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers", Computers & Operations Research, vol. 35, no. 9, pp. 2791-2806, 2008.

[24] M. F. Tasgetiren, Y. C. Liang, M. Sevkli, "A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem", European Journal of Operational Research, vol. 177, no. 3, pp. 1930-1947, 2007.

[25] H. W. Ge, L. Sun, Y. C. Liang, F. Qian, "An effective PSO-and-AIS-based hybrid intelligent algorithm for job-shop scheduling", IEEE Transactions on System, Man and Cybernetics, Part A, Systems and Humans, vol. 38, no. 2, pp. 358-368, 2008.

[26] C. A. C. Coello, D. C. Rivera, N. C. Cortes, "Use of an artificial immune system for job shop scheduling", Lecture Notes in Computer Science, vol. 2787, pp. 1-10, 2003.

[27] J. H. Yang JH, L. Sun, H. P. Lee, Y. Qian, Y. C. Liang, "Clonal selection based memetic algorithm for job shop scheduling problems", Journal of Bionic Engineering, vol. 5, no. 2, pp. 111-119, 2008.

[28] J. H. Holland, Adaptation in Natural and Artificial Systems. Ann Arbor, MI: University of Michigan Press, 1975.

[29] G. E. Goldberg. Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[30] J. Timmis, "Artificial immune systems – today and tomorrow", Natural computing, vol. 6, no. 1, pp. 1-18, 2007.

[31] E. Hart, J. Timmis, “Application areas of AIS: The past, the present and the future”, Applied soft computing, vol. 8, no. 1, pp. 191-201, 2007.
[32] H. P. Schwefel, Evolution and optimum seeking, John Wiley and Sons, New York, 1995.
[33] T. T. Chung, C. R. Wen, “Research on adjusting penalty factors in genetic algorithms for structural optimization”, Bulletin of the college of engineering, NTU, vol. 87, pp. 47-56, 2003.
[34] J. Giffler, G. L. Thompson, “Algorithms for solving production scheduling problems”, Operations Research, vol. 8, pp:487-503, 1960.
[35] J. E. Beasley, “OR-Library: Distributing test problems by electronic mail”, Journal of the Operations Research Society. vol. 41, no. 11, pp. 1069-1072, 1990.
[36] S. Binato, W. J. Hery, D. M. Loewenstern, M. G. C. Resende, “A GRASP for Job Shop Scheduling”, Essays and Surveys in Metaheuristics, 2001, pp. 59-80.
[37] W. P. W. Nuijten, E. H. L. Aarts, “Computational study of constraint satisfaction for multiple capacitated job shop scheduling”, European Journal of Operational Research, vol. 90, no. 2, pp. 269-284, 1996.