Modified Adaptive Evolutionary Algorithm for Solving JSSP Problems

VID OGRIS and TOMAŽ KRISTAN
Algit, d.o.o.
SLOVENIA
vid.ogris@algit.si,tomaz.kristan@algit.si,http://www.algit.si

DAVORIN KOFJAČ
University of Maribor
Faculty of Organizational Sciences
SLOVENIA
davorin.kofjac@fov.uni-mb.si, http://kibernetika.fov.uni-mb.si

Abstract: A job-shop scheduling problem is one of the classic scheduling problems considered to be NP-hard. In this paper, we presenta modified adaptiveevolutionary algorithm (EA) that uses speculative mutations, variable fitness functions and a pseudo-random number generator for solving job-shop scheduling problems. The algorithm was tested on well-known benchmark datainstances, such as Ft10, La01, Swv01, etc., with the goal of achieving the shortest make-span. The results show that using speculative mutations and interval placing reduces the number of steps and computational time to achieve a (near) optimal make-span. Some testing results on an early version of the proposed algorithm are also added,whichwere used to define the most effective types of mutations to generate better offspring.

Key-words: evolutionary algorithm, scheduling, job shop, variable fitness function, speculative mutations

1 Introduction

In a manufacturing process, planning and scheduling are two of the most demanding and critical tasks. The difficulty of determining the optimal schedule depends on the shop environment, the process constraints and the performance indicators. One of the most difficult problems in this area is the job-shop scheduling problem (JSSP)[1]. JSSP has been studied by many authors, and many algorithms have been proposed to solve it.

It is probably impossible to solve it in real time using the “brute force” approach, since JSSP is a NP-hard problem, which was proven by Garey et al. [42]. Problemsofdimensionsof 15×15 are stillconsidered to bebeyondthereachoftoday’s exact methods[7];therefore, other meta-heuristic approaches have beenintroduced:

- Evolutionary algorithms (EA) [8],[9];
- Genetic algorithm [7];
- Taboo search [10],[11],[12];
- Simulated annealing [14];
- Different combinations of methods [15],[16];
- EA-related techniques, such asparticle swarm optimization (PSO), etc.[34], [41].

All these methods are capable of finding a (near) optimal solution in real time.Evolutionary algorithms often perform well in approximating solutions to all types of problems. The most popular type of EA is the GA [7]. Vidal et al.[36] claim that EA solves combinatorial problems effectively, because it adapts to search solutions in a large search space of possible solutions.

This paper describes the use of a modified EA to effectively solve JSSP in a dynamic environment in real time by utilizing speculative mutations, variable fitness functions and a pseudo-random generator.

The rest of the paper is organized as follows. In Section 2, we define the problem; in Section 3, we present our modified EA approach. In Section 4, testing and results performed on benchmark data are presented and discussed. Finally, theconclusion with future work guidelines isgivenin Section 5.

2 Problem Formulation

There are many types of production processes, e.g. Product Layout (Flow Shop) and Process Layout, (Job Shop). The Job Shop problem can be defined as a set of jobs $J = \{ j_1, j_2, \dots, j_n \}$on a set of machines $M = \{ m_1, m_2, \dots, m_m \}$. Job j_i contains a set of *k*operations $O_i = \{o_{i1}, o_{i2}, \ldots, o_{ik}\}$, which are performed on a subset of machines $H \subseteq M$. Operation O_{ik} is defined with continuous time t_{ik}, which is needed for the operation to be completed and machine h_{ik}, with whichthe operation must be executed. The beginning time and ending time of an operation Ojk are denoted by $t^b_{jk}$and t^e_{jk}, respectively. A sequence of operations at each job must be strictly taken into consideration [16].The time required to complete all the jobs is called the “make-span”C_{max}. The objective when solving or optimizing this general problem is to determine the schedule that minimizes C_{max}. Trying to minimize the make-span often brings algorithm to some local minimums. The algorithm is generally not able to determine whether its solution is the best possible or merely another local optimum. Every EA can reachthe global optimum eventually; however, preventing the algorithm from becoming stuck in a local optimum to search for other solutions inthe defined search spaceseems to be the most difficult task.

3 Proposed Evolutionary Algorithm

Evolutionary algorithms are inspired by natural mechanisms of natural selection and population genetics [19]. Some of the current evolutionary approaches include evolutionary programming, evolutionary strategies, genetic algorithms, and genetic programming [2],[3],[4].

Our EA technique essentially consists of the following steps:

1. Initialize the population,
2. Calculate the fitness of the initial population,
3. Perform mutation(s) on population (using speculative mutations and statistics),
4. Calculate fitness of new population,
   a) If a new population is as good as the parent or better, adopt it and delete statistics,
   b) If new population is worse, perform de-mutation to return to the previous state andupdate statistics,
5. Go to Step 3 until some condition is met.

A more detailed explanation of the evolutionary process is given in Fig. 1. The algorithm holds a list of the top 999 schedules. One of them is selected,for which the higher ranked schedules have a better chance of being selected. Some mutation(s) are performed on this schedule and recorded. Next,an evaluation is performed (with the probability of 0.25, we also change the fitness function), at which statistics of the success rate of mutations are recorded. If the new schedule is better or equally good as it was before the mutations, the statistics are deleted, the cycle counter is set to zero, and the process is repeated. If the new schedule is worse, the mutation statistics areupdated, de-mutations on the schedule are performed to set the schedule to the previous version, and the counter is increased by 1. De-mutations reduce the amount of data that has to be held in computer memory;therefore, more operations can be done in less time, since there is less data traffic. If the counter is lower than 100,000 we repeat the cycle;otherwise, the schedule is placed back to the list of the top 999 (according to the initial fitness function),and another one is selected. The time needed to put the schedule to the list and select a new one is much higher than performing operations on one schedule; as a result, the algorithm is working with one schedule for at least 100,000 times. The number of times mutations on a single schedule are performed is empirically defined through the historic data of our EA being used. For example, if the maximum counter number was set to 500,000 or 1 million, we have discovered that the possibility of obtaining a new better schedule was too low against the computational time needed to achieve such a solution.

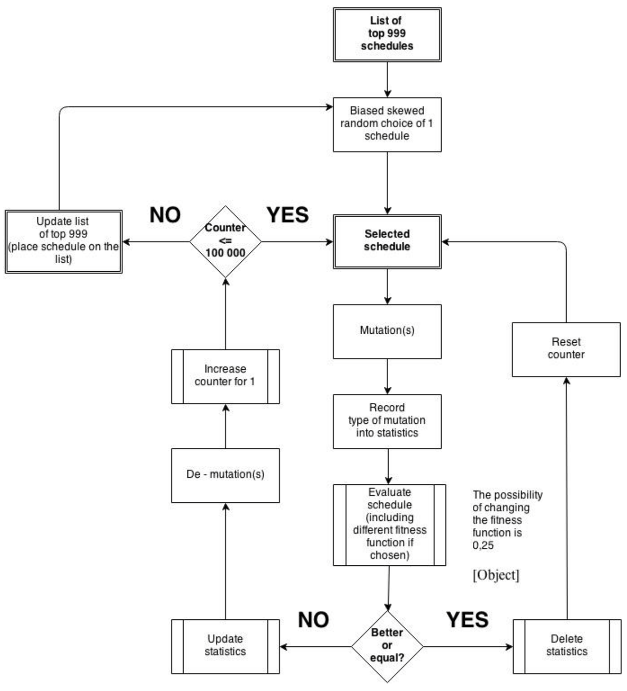
List of top 999 schedules
Biased skewed random choice of 1 schedule
Update list of top 999 (place schedule on the list)
NO
Counter <= 100 000
YES
Selected schedule
Mutation(s)
Reset counter
Increase counter for 1
Record type of mutation into statistics
De - mutation(s)
Evaluate schedule (including different fitness function if chosen)
The possibility of changing the fitness function is 0,25
[Object]
NO
Better or equal?
YES
Update statistics
Delete statistics

Fig.1: Evolutionary process flowchart

It is assumed that a system evolves in a mannerto provide an individual that is in a way better in the next generation, i.e. a mutant with a positive mutation. One of the main advantages of EA is that it has no heuristic or similar rules; it works only on its internal rules.Internal rules when working with JSSP for EA are the same as for JSSP itself. These are:

- An operation cannot be performed on more than one machine at the time;
- The machine cannot perform more than one operation at the time;
- Operations must follow a given order;
- The smaller the time span (or any other criterion), the better the solution.

The algorithm follows only these fundamental rules.

3.1 Fitness function

To evaluate the quality of the population, a fitnessfunction is needed (cost function, criteria function). The fitness function evaluates a solution and is determined by the weighted sum of the violations of the constraints [33].

The fitness function is a measure of how well an algorithm has learned to predict the outputs from the inputs. Anh,Caldeira, Rosa, Beligiannis et al., Manar, Shameem, Cardeira-Pena etal., Birbas et al., Gaemperle et al. [22,23,24,25,26,27,40]have all described the meaning and the use of the fitness function. Birbaset al. [26] are using negative weights to minimize bad properties and to define a better offspring. Dahaletal.[27] described the meaning of a variable fitness function. They are claiming that its use is reasonable in the real world, where optimization and multiple-goal definition are needed. With problems that have a large search space of solutions,difficulty occurs when the search can become stuck at a local optimum. Therefore, the adaptive fitness function is recommended to change the search space, enabling the solution to “escape” from the local minimum[29],[30],[31],[32].When the solution is stuck in a local optimum,to achieve further improvement of the offspring, we can use the landscape change. This is a form of tabooing or advanced tabooing, in which the fitness function changes multiple times during the evolution process (of a time table or schedule).

Globally, we distinguish two principal forms of setting parameter values in a fitness function: parameter tuning and parameter control. Parameter tuning sets the parameters before the algorithm is run, while the parameter control forms an alternative, as the algorithm starts with initial parameter values that are changed during the run [6]. The historic description was done by Eibenetal.[5];they summarize parameter control in the following way:

- Static parameters are not only hard but can be impossible to tune: no good static value exists for the step-size in Gaussian mutation;
- Adaptive methods use some information about the current state of the search, and are as good as the information they get: the success rate is very raw information, and leads to the “easy-to-defeat” one-fifth rule, while Parameter Control in Evolutionary Algorithms 25 CMA-ES uses high-level information to cleverly update all the parameters of the most general Gaussian mutation;
- Self-adaptive methods are efficient methods when applicable, i.e. when the only available selection (based on the fitness) can prevent bad parameters from proceeding to future generations. They outperform basic static and adaptive methods but are outperformed by clever adaptive methods.

Parameter control,which is not commonly used, may provide a useful mechanism for increasing the performance of the algorithm [6].

The fitness function F used in our research is given in Eq. 1. We are trying to minimize the make-span of the schedule and the penalties if the certain criteria are violated, such as wrong order of operations, length operation overlaps per job and length of operations overlaps per machine.

$$\min F = 10^x \sum_{j=1}^{n}\sum_{k=1}^{m} Y_{jk} + 10^y \sum_{j=1}^{n}\sum_{k=1}^{m} T_{jk} + 10^z \sum_{k=1}^{m}\sum_{j=1}^{n} T^J{}_{jk} + 10^u C_{\max} \quad (1)$$

Where

- $\Delta \in [0,1]$ – weight, which is randomly changed in every step during the evolution.
- Variables x, y, z and u change in every step during the evolution according to:$x = 6+\Delta x$, $y = 3+\Delta y$, $z = 3+\Delta z$, $u = 0+\Delta u$.
- $Y_{jk} = 1$ if operation $O_{jk}$is scheduled before operation(s) that should be performed prior to O_{jk} in job j; 0 otherwise.
- $T_{jk}$is the time overlap for operation $O_{jk}$with the operation $^*O_{jk}$if $t^e_{jk} > {}^*t^b_{jk}$, on machine k, where $T_{jk} = t^e_{jk} - {}^*t^b_{jk}$.
- $T^J{}_{jk}$ is the time overlap for operation $O_{jk}$with the operation $^+O_{jk}$if $t^e_{jk} > {}^+t^b_{jk}$, within a job j, where $T^J{}_{jk} = t^e_{jk} - {}^+t^b_{jk}$.

- C_{max} – length of the whole process.

Buecheet al.[28]optimize the fitness function or define an alternative fitness function, but they never bring it back to the initial state. If, after a certain number of generations, there is no improvement, the algorithm changes the fitness function, creates a new branch, works with it; then, after a certain number of generations without improvement, the new offspring are evaluated and compared with generations based on the original values of fitness function. If anyschedule amongthe new ones is placed on the list of top schedules, we continue with the primary branch of schedules (including new ones); otherwise, the process is repeated. Furthermore, in most cases in the literature, the schedule is taken into the next generation only if it is strictly better than the best schedule of the previous generation. However, we take the generation's best schedule in the next generation if the schedule is at least as good as the best schedule in the previous generation. Preliminary tests have shown that random selection between equally good solutions increases the possibility of obtaining a new, better solutions.We believe that the new better offspring can be found from the generation that is as good as others but different.

Since validating the schedule is time consuming, we provide some shortcuts. First, if the difference between absolute minimum and maximum fitness value has increased, the new schedule is considered to be bad, and no further validation is performed. Then, machines and operations where no mutations occurred are not checked. After that, all other validation tasks are performed.

3.2 Mutations

After the initial population is randomly set, our algorithm works exclusively with mutations. No cross-over is involved, since our tests have proven to be computationally too expensive, because of computationally expensive schedule validation.

The algorithm uses seven types of mutations. With a random choice of RAND(RAND(7)), the algorithm chooses how many mutations it will perform inone generation. This number was chosen after some initial testing, which will be explained later in the paper. The mutations considered in our algorithm are the following:

- Shift operation left or right on one machine;
- Shift operation left or right on an interval of more machines;
- Swaptwooperations on one machine;
- Rotatethree operations on one machine(on M_n operations O_1, O_2, O_3 are rotated, so the new order is O_3, O_1, O_2);
- Shift left or right one operation on all machines;
- Adjusted rotation (two operations on one machine are rotated, and the start time of the second one is then corrected, so it starts right after first one. The total time of both operations summed together remains the same);
- Random set of one operation(place operation $O_n$randomly on the machine).

3.3 Co-evolution

Mohammadi et al. [38] suggest using co-evolution to solve the problem of scheduling (school time-tabling problem), which provided good results in real time. They believe that the usual EA cannotyieldacceptable results in real time;therefore, its use is not preferable. We believe that every evolution is a co-evolution. If something evolves in an unchanged environment,it is evolution without co-evolution, but since the schedule of one machine is an environment for another machine, we already have co-evolution; furthermore, the results are obtained in real time.

3.4 Speculative mutations

If mutationsprovide a schedule that is as good or better than the previous one, we consider the mutation to be successful. Inside each cycle and with a certain probability, the knowledge about successful mutations is used to choose more successful mutations, since we can expect that the success rate of a mutation will drop during execution time. The success rate of a mutation (statistics) is held in the program's memory cache, between two successive improvements (i.e. a generation is better or as good as previous one); after which it is deleted. This is used in a so-called speculative mutation process and drastically improves the speed of algorithm;therefore, itcan significantly decrease the calculation time.

The algorithm decides with aprobability of 0.5 whether or not it will rely on thestatistics. If it decides not to, then each mutation hasa probability of p = 1/7of being chosen;however, after it is chosen, some additional pre-calculations are made, e.g. if the number of mutations in one generation is one and the operation, which is the subject of mutation, is neither the first or last, it is pointless to move (or set) it out of the current intervalat whichthe interval is a space between the starting time of the first operation on M_n and the start time of the last operation on machine M_n. When we have a sequence of mutations in one generation, at least one needs to have a potential to decrease the time span of out process.In other words, if sixmutations are selected and none of them improves or at least does not worsen the make-span, that sequence is obsolete.

In addition, we prevent some successive mutations on one object from happening, e.g. in a sequence of mutations, where we move O_1 on $M_1$to the left byfive units and then another mutation wants to move O_1 on M_1 to the right by two units. The second case is disabled in advance, thus saving computational time.

Another way to save computational time is to prevent other unfeasible solutions by narrowing intervals, where the set mutation can place an operation on a certain machine. An operation cannot be set to a location before its ideal place, e.g. operation O_3 cannot be placed before the end of total time of $O_1 + O_2$, even in an ideal case.Further, O_3 cannot be placed after total time span minus the sum of all operations following O_3,including the duration of O_3.

$$\mathrm{T}(\mathrm{O_m}) \geq \sum_{i=1}^{m-1} \mathrm{T}(O_i) \qquad (2)$$

$$\mathrm{T}(\mathrm{O_m}) \leq \mathrm{C} - \sum_{i=m}^{n} \mathrm{T}(O_i) \qquad (3)$$

where:

C - (current) total make-span,

T - starting point of operation.

This rule is applied to mutations of type set, shift left and shift right. Furthermore, if we swap two operations between two machines (O_1 on M_1 starts when O_2 on M_2 started, and O_2 on M_2starts when O_1 on M_1has started), this rule is verified for both operations on their new place. If the condition is not met for only one of them, the mutation is discarded.

For mutations that are surelynot valid, we apply another constraint so that they never occur. For example, an interval for an operation where it can begin (and end)is known; therefore, they are never set outside of this interval and, consequently, we drastically reduce the number of trials to obtaina valid position for an operation.

3.5Random number generator

The role of the random generation is to insert some chaos into the system and consequently reduce the time needed to obtain a solution. This is crucial to exit local optimum(s).

One of the key features to improve the speed of the algorithm is our random number generator: a modified MersenneTwister[39] algorithm;our version of the algorithm is optimized for speed. Our primary goal is to obtain the random numbers as quickly as possible, while the quality (the numbers being regularly distributed) of the random generator is of secondary meaning, since obtaining those numbers is very time consuming with regards to theCPU. With one query from theCPU,we obtain 32 random numbers that our EA can then use for its manipulation. We believe that the solution is obtained much more quickly this way.

Although the random number generator is optimized for speed, it still represents a large bottleneck in the entire process, since speculative mutations reject the majority of the random numbers received from the CPU, and obtaining them is very time consuming, e.g. if the numbers received from the CPU are 1, 1, 4, 1, 3...,and 1 turns out to be unsuccessful, it skips to number 4;if that turns out to be successful, it uses number 1 again since it follows in the sequence; otherwise, it skips 1 again and uses the next number, i.e. 3.

4 Results

First, we wanted to find the success rate of mutations. We tested the following instances and performed some analytics on the results:

- From la01 to la05 – size 10 × 5,proposed by Lawrence[35],
- From la06 to la10 – size 15 × 5, also proposed by Lawrence[35],
- From swv06 to swv10 – size 20 × 15, proposed by Storer, Wu and Vaccari[37],
- Results of all instances together.

We recorded the time to find the solution that matches the best known solution.We also analyzed some of the mutations:

- Number of mutations per generation and success;
- Number of mutations per generation and instance and, consequently, the success;
- Shift operation left or right and, consequently, the success;
- Mutation set and, consequently, the success;
- Mutations swap two or three operations and, consequently, the success.

Analytics wereperformed with SPECTORsoftware (www.algit.eu). Thisis a data-mining program that searches for samples in sets of data. It transforms data sets to logical implications, finds all regularities in data and quantitatively evaluates them. We let the EA run for 30seconds,regardless of the final result, since the files with the results would become too large to be processed for analytics. Despite the time thatwas spent to record all of the mutations, the best known results for some instances were achieved. The algorithm could choose between seven mutations, and the maximum number of mutations per generation was between 1 and 16. The results were as follows:

- Instances from La01 to La05
  - If the number of mutations per generation (NOMG) equals 1, the success rate is above average;
  - If NOMG was greater than 10, the success rate was under average;
  - If the NOMG equals 2, the success rate was above average for instances la01 and la03;
  - If the mutation set is used above average, the success rate is below average.
- Instances from La06 to La10:
  - If NOMG is 1,8,9,10, the success rate is above average;
  - If mutation set is used above average, the success rate is below average.
  - If mutations swap 2 or swap 3 are used above average, the success rate is below average.
- Instances from Swv06 to Swv10:
  - If NOMG is used above average, the success rate is below average;
  - The mutation set turns out to be successful under average;
  - Shifting operations to the right increases the success rate.
  - Analytics on all instances yield the following results:
  - Success rate on instances Swv08 and Swv09 is above average;
  - If operations are movedto the right, success is above average;
  - Using the mutation swap of two elements is successful if it is used below average.

This analysis helped us in the further development of our algorithm, because we can guide the program to use mutations that are more successful than others in producing a better offspring.

The aforementioned analytics yielded sevenof the most successful mutations described earlier. These mutations were utilized in our tests on the following instances: Ft06 (6 × 6), Ft10 (10 × 10), La01 (10 × 5), La02 (10 × 5), La11 (20 × 5), La12 (20 × 5),La30 (20 × 10), La31 (30 × 10), Swv01(20 × 10), Swv02 (20 × 10). We performed 100 runs per each instance, with the following options on the program:

- All optimizations are on (OPT);
- The program can choose from all mutations, the interval setting is on, using statistics is off (OPT_noSTAT);
- The program can choose from all mutations, the interval setting is off, using statistics is off (NoSTAT_NoINT);
- The program can choose from all mutations, the interval setting is off, using statistics is on (OPT_noINT);
- The program cannot use mutations shift left and shift right, the interval setting is on, using statistics is off (NoSTAT_NoMUT);
- The program cannot use mutations shift left and shift right, the interval setting is on, using statistics is on (NoMUT);
- The program cannot use mutations shift left and shift right, the interval setting is off, using statistics is off (NoOPT);
- The program cannot use mutations shift left and shift right, the interval setting is off, and using statistics is on (NoINT_NoMUT).

Computational time was limitedto eitherthebestknownsolutionbeingfoundorto a maximum of 100,000 generations without improvement.

The results are shown in the following tables. Table 1 shows the average number of steps for the algorithm to reach its best solution, Table2 showsaveragetimesfor the analgorithmto reachitsbestresult,andTable3shows thebestresultthealgorithmhasachieved.

In Table 1, we can see how many steps (improvements) a different version of algorithm needed (in average) to achieve its best result for a particular benchmark problem. We can see that both versionsof the program that did not use the shift left or shift right mutations (NoSTAT_NoMUT and NoMUT) performed best. NoMUT achieved the lowest number of steps 7 times, while the NoSTAT_NoMUT achieved it 2 times. It seems like the mutation shift left and right is usually unsuccessful; therefore, these two algorithms benefit from that.

Table 2 represents the average computational time to achieve the best result. Similar to results in

Table 1, NoSTAT_NoMUT and NoMUT perform best, yielding lowest average times, again benefiting from not using the shift left and shift right mutations.

Table 1: Average number of steps to find a solution

Instance	OPT	OPT_ noINT	NoMUT	NoINT_ NoMUT	OPT_ noSTAT	NoSTAT_ NoINT	NoSTAT_ NoMUT	NoOPT
ft06	59.88	68.61	58.85	68.83	66.61	73.62	***56.09***	70.09
ft10	378.97	503.12	***298.37***	352.25	397.31	535.63	362.61	482.75
la01	243.73	386.99	183.94	234.07	257.51	368.99	***176.60***	259.02
la02	251.37	323.14	***164.93***	195.29	234.36	347.55	190.47	229.11
la11	468.54	753.09	***312.00***	362.73	395.24	709.46	323.07	405.98
la12	415.32	606.18	***314.88***	344.26	403.68	664.84	343.58	409.97
la30	801.64	1012.77	***674.88***	906.96	818.74	1165.46	713.72	810.46
la31	1177.67	1409.36	930.34	***849.98***	1083.61	1379.83	905.97	972.37
swv01	718.83	1078.21	***595.39***	642.75	778.91	1084.95	631.07	644.06
swv02	744.70	1030.10	***590.59***	598.73	776.56	1050.12	693.30	630.80

Table 2a: Average computational time

Instance	OPT	OPT_ noINT	NoMUT	NoINT_ NoMUT	OPT_ noSTAT	NoSTAT_ NoINT	NoSTAT_ NoMUT	NoOPT
ft06	0.54	0.46	0.33	0.43	0.94	0.61	***0.26***	0.51
ft10	4.03	3.67	***3.13***	4.03	5.84	6.35	5.27	5.77
la01	0.7	0.71	0.67	0.74	0.99	1.21	***0.65***	0.7
la02	1.14	1.3	***0.72***	1.21	1.8	1.97	1.67	1.57
la11	1.12	1.11	***0.91***	1	1.42	1.37	1.4	1.38
la12	1.26	1.28	***1.2***	1.24	1.74	1.67	1.79	1.75
la30	24.92	22.71	***20.18***	26.18	25.9	29.86	23.41	30.96
la31	29.78	27.93	19.3	***19.1***	33.3	33.81	30.61	31.15
swv01	19.15	24.12	***16.6***	20.25	26.24	29.33	31.42	30.24
swv02	22.97	19.9	***18.94***	21.45	29.48	28.72	25.72	25.7

Table 3a: The best make-span result

Instance	OPT	OPT_ noINT	NoMUT	NoINT_ NoMUT	OPT_ noSTAT	NoSTAT_ NoINT	NoSTAT_ NoMUT	NoOPT
ft06	55	55	55	55	55	55	55	55
ft10	978	987	971	1017	971	992	992	978
la01	666	666	666	666	666	666	666	666
la02	655	655	655	655	655	655	655	655
la11	1222	1222	1222	1222	1222	1222	1222	1222
la12	1039	1039	1039	1039	1039	1039	1039	1039
la30	1355	1355	1355	1355	1355	1355	1355	1355
la31	1784	1784	1784	1784	1784	1784	1784	1784
swv01	1570	1542	1556	1540	1556	1537	1558	1569
swv02	1585	1574	1580	1614	1577	1620	1624	1573

In Table3 the best make-span results are presented. It is worth noticing that for problem instances ft06, la01, la02, la11, la12, la30 and la31 all algorithms yielded the same results. The difference is in the instances ft10, sww01 and sww02, where NoMUT, NoSTAT_NoINT and

NoOPT yielded the best results, respectively.Again, in general, NoMUT algorithm achieved good results. It seems that using statistics is not a major benefit for these benchmarks.

We can see that, without using statistics, the computational times are longer, and the number of steps needs to obtain the best results ishigher. In addition, using interval placementgenerallyresultsin achieving a smaller number of steps to obtaina better result, while that is not true for shortercomputational time, since it will not yield better results. If we want a good comparison of all methods,the following considerations must be taken into account. Not only the best solution matters, but also the time needed to achieve this solution. If we look at Table 1, where the number of steps is shown, we have to compare them with the best results these methods have achieved. For some problem instances, some methods seem to be better, but when examining the computational time and best result, we can see that the NoMUTalgorithm yieldsthe best results in 7 out of 10 cases. The best result is usually achieved with two of the most optimized methods. If also considering NoSTAT_NoMUTalgorithm, it is obvious that shift left and/or shift right mutations do not contribute in achieving the best results. Moreover, to obtainonly a slightly better result, the number of times required is considerably higher, meaning that an algorithm performing computations several times faster does not necessarily provide us with results that are several times better.

5Conclusion

Examining the results achieved with the proposedevolutionaryalgorithm, we can see that using speculative mutations (statistics) and interval placing aid in achieving better results in a shorter amount of time.Using SPECTOR, we have defined how many mutations per generations and which ones to use, running it on early versions of the algorithm.It was shown that using shift left and/or shift right mutations did not contribute in achieving better results. Also, a very important feature of our EA is the ability to exit local optimums in case it is stuck in one of them. This is achieved with a variable fitness function and pseudo-random number generator that enables all available numbers to be selected (from 1 to 7, since there are seventypes of mutations).

This algorithm, with some modifications, is also used in the program for calculating school time tables (iTimeTable) and worker schedules (WoShi), where the number of combinations between teachers, students, lessons and classrooms exceeds 10^{54} and brute force cannot be used;our algorithm achieves quality results in a reasonable amount of time.

Future research will be focused on the influence of de-mutation, co-evolution and pseudo random generator on algorithm performance. We assume that using a pseudo-random number generator reduces the time to obtain the random numbers that are essential for our evolutionary algorithm. Further, our research will pay attention to implement this algorithm inreal-world production scheduling. We also see its usage in worker scheduling on all areas from production industry to hospitals, etc., where even more constraints are included.There is also some room for algorithm improvement and with the improvement of CPU speed and the number of cores;the results will be achieved more rapidly, since our algorithm is capable of using multiple cores for calculation; it is also capable of running on multiple computers; each computer sends its results to the server and with a certain probability decides whether or not it will use the best known solution on the server and continue the evolution from that point onward.

References:

[1] R. Qing-doa-er-ji and Y. Wang. A new hybrid genetic algorithm for job shop scheduling problem, *Computers& Operations Research*, 39(10), 2012, pp. 2291-2299.

[2] W.Banzhaf, P. Nordin, R.E. Kellerand F.D.Francone,*Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Application*. San Francisco: Morgan Kaufmann, 1998.

[3] D. Dasgupta and Z. Michalewicz,*Evolutionary Algorithms in Engineering Applications*, Berlin: Springer-Verlag, 1997.

[4] C.M. Fonseca and P.J.Fleming, Multiobjective optimization and multiple constraint handling with evolutionary algorithms, *Part I:Unified formulation. — IEEE Trans/SMC*, Part A: Syst. Hum., Vol. 28, No. 1, 1998, pp. 26–37.

[5] A.E. Eiben, Z.Michalewicz, M. Schoenauer and J.E. Smith, Parameter Control in Evolutionary Algorithms, *Studies in Computational Intelligence*,Vol. 54, 2007, pp. 19–46.

[6] A.E. Eiben, R. Hinterding, and Z. Michalewicz, Parameter control in evolutionary algorithms, *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 2, 1999, pp. 124–141.

[7] J. Magalhães-Mendes,A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop scheduling Problem, *WSEAS transactions on computers*, Vol. 12, No. 4,2013, pp. 164-173.

[8] Q.X.Yun, W.W. Guo, Y. Che, C.W. Lu, and M.I. Lian, Evolutionary algorithms for the optimization of production planning in underground mines, *Application of Computers and Operation research in the Minerals Industries*, South Africa Institute of Mining and Metallurgy, 2003.

[9] F. Koblasa, F. Manlig andJ. Vavruška, Evolution Algorithm for Job Shop Scheduling Problem Constrained by the Optimization Timespan, *Applied Mechanics and Materials*, Vol. 30, 2003, pp. 350-357.

[10] A. Abraham, R. Buyya andB. Nath,Nature's heuristics for scheduling jobs in computational grids, *Proceedings of the 8th IEEE International Conference on Advanced Computing and Communication*, 2000, pp. 45–52.

[11] E. Nowicki and C. Smutnicki, A fast taboo search algorithm for the job shop problem, *Management Science*, Vol. 42, No. 6, 1996, pp. 797–813.

[12] M. Dell'Amico andM. Trubian, Applying taboo search to the job-shop scheduling problem, *Annals of Operations Research*, Vol. 41, 1993, pp. 231–252.

[13] C. Zhang, X. Shao, Y. Rao and H. Qiu, Some New Results on Tabu Search Algorithm Applied to the Job-Shop Scheduling Problem, Tabu Search, *WassimJaziri* (Ed.), ISBN: 978-3-902613-34-9, InTech, 2008.

[14] M. Brusco andL. Jacobs, A simulated annealing approach to the cyclic staff-scheduling problem, *Naval Research Logistics*, Vol. 40, 1993, pp. 69–84.

[15] R. Tavakkoli-Moghaddam, F. Jolai, F. Vaziri, P.K. Ahmed and A.Azaron, Solving stochastic job shop scheduling problems by a hybrid method,*Applied Mathematics and Computation*, Vol. 170, No. 1,2005, pp. 185–206.

[16] A. Tamilarasi andT. Anantha, An enhanced genetic algorithm with simulated annealing for job-shop scheduling, *International Journal of Engineering, Science and Technology*,Vol. 2, No. 1, 2010, pp. 144-151.

[17] D. Kofjač andM. Kljajić, Application of genetic algorithms and visual simulation in a real-case production optimization. *WSEAS transactions on systems and control*, Vol. 3, No. 12, 2008, pp. 992-1001.

[18] R. Thamilselvan and P. Balasubramanie,Integrating Genetic Algorithm, Tabu Search Approach for Job Shop Scheduling, (IJCSIS) *International Journal of Computer Science and Information Security*,Vol. 2, No. 1, 2009, p. 6.

[19] K. Mesghouni, S.Hammadi and P. Borne, Evolutionary algorithms for job-shop scheduling, *International Journal of Applied Mathematics and Computer Science*, Vol. 14, No. 1, 2004, pp. 91–103.

[20] A.E. Eiben and J. Smith, *Introduction to Evolutionary Computing*, Springer, Natural Computing Series, 1st edition, 2003.

[21] D.T. Anh, V.H. Tam andN. Hung,Generating Complete University Course Timetables by Using Local Search Methods, *Research, Innovation and Vision for the Future*, International Conference, 2006.

[22] J.P.Caldeira and A.C. Rosa, School Timetabling using Genetic Search, *PATAT* 97, 1997, pp. 115-122.

[23] A. Cerdeira-Pena, L. Carpente, A.Farina andS. Diego, New approaches for the school timetabling problem, Seventh Mexican *International Conference on Artificial Intelligence*, 2008.

[24] G.N. Beligiannis, C.N. Moschopoulos, G.P. Kaperonis andS.D. Likothanassis, Applying evolutionary computation to the school timetabling problem: The Greek case, *Computers & Operations Research*, Vol. 35,2008, pp. 1265 – 1280.

[25] H.Manar andF. Shameem,A Survey of Genetic Algorithms for the University Timetabling Problem, *International Conference on Future Information Technology IPCSIT* Vol.13, 2011.

[26] T. Birbas, S. Daskalaki andE. Housos, Timetabling for Greek high schools, *Journal of the Operational Research Society*, Vol. 48, 1997, pp. 1191-1200.

[27] K. Dahal, S. Remde andP. Cowling, Improving metaheuristic performance by evolving a variable fitness function, Evolutionary computation in combinatorial optimization, proceedings,*Book Series: lecture notes in computer science*,Vol. 4972,2008, pp. 170-181.

[28] D. Bueche, N.N. Schraudolph andP. Koumoutsakos, Accelerating Evolutionary Algorithms with Gaussian Process Fitness Function Models, *IEEE transactions on systems, man, and cybernetics*, Vol. 35, 2004, pp. 183-194.

[29] M.A. Majig and M. Fukushima, Adaptive Fitness Function for Evolutionary Algorithm and Its Applications,*International Conference on Informatics Education and Research for Knowledge-Circulating Society*, Kyoto, 2008, pp. 11-124.

[30] P. Tang andG. K. Lee, An Adaptive Fitness Function for Evolutionary Algorithms Using Heuristics and Prediction, *World Automation Congress*, 2006. WAC '06, 24-26 July 2006, Budapest,pp. 1-6.

[31] R. Farmani andJ.A Wright, Self-Adaptive Fitness Formulation for Constrained Optimization, *IEEE transactions on evolutionary computation*, Vol. 7, No. 5, 2003, pp. 445-455.

[32] A.E. Eiben and Z. Ruttkay, Self-Adaptivity for Constraint Satisfaction: Learning Penalty Functions, *International Conference on Evolutionary Computation*, 1996, pp. 258-261.

[33] P. De Causmaecker, P. Demeester andG.V. Berghe, A decomposed metaheuristic approach for a real-world university timetabling problem, *European Journal of Operational Research*, Vol. 195,2009, pp.307–318.

[34] D.Y.Sha andC. Hsu, A hybrid particle swarm optimization for job shop scheduling problem,*Computers & Industrial Engineering*, Vol.51,2006, pp. 791-808.

[35] S. Lawrence, *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*, *GSIA*, Carnegie Mellon University, Pittsburgh, PA, 1984.

[36] J.C. Vidal, M. Mucientes, A. Bugar´in andM. Lama, An Adaptive Evolutionary Algorithm for Production Planning in Wood Furniture Industry,*International Symposium on Evolving Fuzzy Systems*, September, 2006.

[37] R.H. Storer, S.D. Wu and R. Vaccari,New Search Spaces for Sequencing Instances with Application to Job Shop Scheduling, *Management Science*, Vol. 38, 1992, pp. 1495-1509.

[38] M.S. Mohammadi and C. Lucas, Cooperative Co-evolution for School Timetabling Problem, *7th IEEE International Conference on Cybernetic Intelligent Systems*, CIS, 2008.

[39] M. Matsumoto and T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, Vol. 8, No. 1, 1998, pp. 3 – 30.

[40] R. Gaemperle,S.D. Muller and P.Koumoutsakos,A parameter study for differential evolution,*WSEAS International Conference on Advances in Intelligent Systems, Fuzzy Systems*, *Evolutionary Computation, 2002,* pp. 293–298.

[41] S. Rahnamayan and G.G. Wang, Solving Large Scale Optimization Problems by Opposition-Based Differential Evolution (ODE), *WSEAS Transactions on Computers*, Vol. 7, No. 10, 2008, pp. 1792-1804.

[42] M.R. Garey, D.S. Johnson, and R. Sethi, The complexityof flow-shop and job shop scheduling,*Mathematics and Operations Research*, Vol. 1, No.2, 1976, pp. 117–129.