Solving permutation problems with Differential Evolution: An Application to the Jobshop Scheduling Problem

Antonin Ponsich[1], Ma. Guadalupe Castillo Tapia[2], Carlos A. Coello Coello[1]

[1]CINVESTAV-IPN (Evolutionary Computation Group)
Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D.F. 07300
antonin@computacion.cs.cinvestav.mx
ccoello@cs.cinvestav.mx

[2]Universidad Autónoma Metropolitana, Unidad Azcapotzalco
Av. San Pablo No. 180, Col. Reynosa Tamaulipas. México, D.F. 02200
m3g2c5t@prodigy.net.mx

Abstract

This study addresses the solution of Jobshop Scheduling Problems using Differential Evolution (DE). The issue of representing permutations through real numbers constitutes the key issue for developing an efficient implementation. Several techniques are empirically validated on problem instances traditionally adopted in the specialized literature. We also present a simple hybridization of DE with tabu search, which produces significant performance gains.

1. Introduction

Among the wide diversity of techniques developed in the last decades for the solution of hard optimization problems, Differential Evolution (DE) has attracted a lot of attention from the continuous optimization community. The specific mutation operator introduced by Storn and Price [16] in the mid-1990s indeed constituted a breakthrough in the evolutionary computation area, combining great efficiency and simplicity. The efficacy and efficiency of this optimization technique have been shown in a wide variety of continuous optimization problems. However, the lack of studies on the use of DE in permutation-based problems (i.e., for which the decision variables must represent a permutation of integers) is remarkable. In fact, the representation of permutations using real numbers is still an ongoing task, with a lot of room for improvement.

The classical Jobshop Scheduling Problem (JSP) provides, in this perspective, a very good framework for the above-mentioned task. Due to its inherent complexity, JSP has been widely used in order to test the development of new optimization techniques. Exact methods such as branch-and-bound [10] or shifting-bottleneck based heuristics [1, 3] have been applied to it, but, more recently, a lot of effort has been devoted to the use of metaheuristics in this problem: simulated annealing [17], genetic algorithms [5] or GRASP [2] only represent a few examples among the pletora of available solution strategies. Currently, the most powerful approach available for this problem is still Nowicki and Smutnicki's TSAB algorithm [14], subsequently improved with the *i*-TSAB algorithm [15].

However, almost no mention of applications of simple DE to JSP appears in the literature. Thus, the aim of this paper is to provide some insights about the use of real numbers encodings for permutation problems with DE, adopting the JSP as a framework. The remainder of this paper is organized as follows. The problem framework, including a short presentation of the DE method and of JSP, is defined in Section 2. Section 3 proposes various strategies for the permutations representation inside DE, as well as two versions of the schedule builder, needed for the objective evaluation. Computational experiments are discussed in Section 4, while Section 5 shows how the former results can be significantly improved through hybridization with a local search method. Finally, some conclusions and perspectives are drawn in Section 6.

2. Study framework

2.1. Outline of the DE method

Differential Evolution is a very recent Evolutionary Algorithm (EA), developed by Storn and Price [16]. Its basic principle relies on the design of a simple mutation operator based on the linear combination of three different individuals x_{1j}, x_{2j}, x_{3j}, to obtain a mutant u_{ij} as shown in Equation (1):

$$u_{ij} = x_{1j} + F(x_{2j} - x_{3j}), \forall j = 1, ..., N \qquad (1)$$

Then, a crossover step mixes the initial and mutated solutions, according to probability *CR*. The typical termination criterion is to reach a given number of generations *NG*. *F* (amplification factor) and *CR* (crossover rate) are control parameters that, with *NP* (population size) and *NG*, must be appropriately tuned. Among the existing DE versions, the DE/rand/1/bin one will be used next (see [16] for details).

2.2. Statement of the JSP

The classical Jobshop Scheduling Problem consists in assigning *m* (scarce) resources, i.e. machines, to *O* operations belonging to *n* jobs. The operations of each job *j* must follow a specific sequence and must be processed exactly once on each machine *i*. One machine cannot process simultaneously more than one operation. The objective is to minimize the makespan.

Many formal representations of the JSP exist but the most traditional model is a disjunctive graph [10], whose nodes stand for the operations. These nodes are connected by conjunctive arcs that denote the operation sequence of a job, while disjunctive arcs represent the permutation of the different jobs' operations on a given machine. Therefore, the resulting complexity of the problem can be formulated as *(n!)m*.

2.3. Aims of this study

The key issue, when trying to solve JSP with DE, is to determine a representation scheme that allows the use of real numbers to encode (feasible) permutations. Several techniques, implemented in the framework of this study, focus on viable transformation methods from the real numbers space to the space of permutations.

A further issue concerns the construction of a feasible schedule, since the only determination of a job permutation on each machine does not allow computing the makespan objective. Actually, for a defined multi-permutation (i.e., a set of permutations associated to each machine), an infinity of schedules might be built. Among them, the active schedules are those for which no operations can be brought forward without delaying another operation. It is well known that optimal schedules belong to the active class [12]. Another relevant schedule class is the non-delayed one, for which an operation is sequenced as soon as a machine becomes free. This latter class constitutes a sub-group of the former that may not contain the optimal solution.

Extensive computations will be carried out on a selection of widely used instances, in order to determine which procedures might be implemented within DE in order to treat both issues. Some conclusions about the quality of the results achieved will finally lead us to propose a hybridization of DE with a local optimizer.

3. Adapting DE to JSP

This section presents the modifications that must be integrated into DE in order to solve the JSP. All the other features of the DE algorithm remain unchanged.

3.1. Permutation representation modes

Among the existing versions of permutation representations available in the literature, only few of them can be adapted to the real-numbers encoding DE:

- *Random keys*.

This technique, initially proposed in [4], uses for each operation a real number, bounded between 0 and 1. The operations on a machine are sequenced according to the increasing order of their associated variable. For instance, consider five jobs with the following variable vector on machine *i*: [0.41 0.68 0.02 0.85 0.37]. The lowest value is 0.02 in position 2 so job 2 will be sequenced first; the second lowest value is 0.37 in position 4 so job 4 will be sequenced in second place; etc. The final sequencing order is: [2 4 0 1 3]. For a $n \times m$-JSP, *nm* variables are needed to represent a complete multi-permutation.

- *Dispatching rules*.

Dispatching rules are widely used in scheduling problems and have been found to achieve good results for simple machine configurations. So, [6] introduced the idea of evolving a set of dispatching rules, which will be subsequently used during the schedule building process in order to solve conflict cases (i.e. cases when several operations are valid candidates for the processing on a machine). We refer the reader to [12] for a complete description of the most commonly used dispatching rules. In this study, the following ones are accounted for:
- SPT: shortest processing time first;
- LST: least work remaining first;
- FCFS: first come first served;
- FOFO: first off first on;

- MS: minimum slack time first;
- COVERT: parameterized combination of SPT/MS;
- MS/OPN: min. slack time per remaining operation. A variable must, in this sense, identify a rule that would settle a conflict in iteration *j* of the schedule building process. Since seven rules are taken into account, the continuous variables are bracketed between 0 and 7 and the real value is truncated in order to obtain a rule identifier. There may be at most *n*-1 conflicts on one machine, and thus this encoding technique needs (*n*-1)*m* variables for a *n*×*m*-JSP.

- *Binary priorities*.

This encoding technique was initially proposed in [13] to solve the JSP with genetic algorithms (GAs). It is based on the disjunctive graph representation mentioned in Section 2.2. A binary variable x_{ijk} is associated to each disjunctive arc and enforces the priority of two operations *j* and *k* on machine *i*: x_{ijk}=0 if *k* is scheduled before *j* and x_{ijk}=1, otherwise. The drawback is that a feasible schedule is not implicitly produced. Indeed, two kinds of infeasibilities can appear:

(i) Local infeasibilty occurs when an operation cycle is obtained on a single machine (*j*→*k*→*l*→*j*), leading to an invalid permutation. Nakano and Yamada [13] designed a local harmonization procedure that repairs the inconsistent permutation by computing a priority matrix M_i for machine *i*. Each term (*j*,*k*) of this matrix represents the variable x_{ijk}. The sum S_j of each matrix row *j* is an integer that actually constitutes the global job sequencing priority on the machine. Thus, the vector [S_j, *j*=1,...,*n*] should constitute a valid permutation. If not, one randomly chosen matrix term is replaced by its complementary value to get a valid permutation The process starts from the row with the highest S_j value and is repeated for each row, in the decreasing order of S_j, to obtain a valid permutation.

(ii) Global infeasibility occurs when a discrepancy appears between local machine permutations and job operating sequences. This issue is fixed during the schedule building process (although the final schedule does not always respect the encoded permutation).

In this study, a similar method is applied by rounding off the real variables, bounded by 0 and 1, to the closest integer value, i.e. x_{ijk}=0 if x_{ijk}<0.5 and x_{ijk}=1, otherwise. In case of local infeasibility, a modified local harmonization procedure is carried out, taking advantage of the continuous nature of the DE variables. When various terms, on different rows or not, are candidates for the complementary value swapping, we choose the one whose value is closest to 0.5 instead of randomly. For instance, let us consider a 5×*m*-JSP. The priority matrix for machine *i* and its binary interpretation are shown in Figure 1. The associated vector S_j is [4 2 2 2 0], which is clearly not a valid permutation. A term among rows 1, 2 or 3 must be replaced by its complementary value and x_{i12} is chosen since it has the value closest to 0.5. Rows 1 and 2 of the interpreted matrix are now respectively (0 - 1 1 1) and (0 0 - 0 1), leading to the consistent permutation S_j = [4 3 1 2 0]. To conclude, notice that the number of variables is greater than those of the former cases: for a *n*×*m*-JSP, *n*(*n*-1)*m*/2 variables are needed.

$$M_j = \begin{bmatrix} - & 0.56 & 0.79 & 0.61 & 0.90 \\ 0.44 & - & 0.48 & 0.56 & 0.72 \\ 0.21 & 0.52 & - & 0.42 & 0.89 \\ 0.39 & 0.44 & 0.58 & - & 0.64 \\ 0.10 & 0.28 & 0.11 & 0.36 & - \end{bmatrix}$$

$$\equiv \begin{bmatrix} - & 1 & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 1 \\ 0 & 1 & - & 0 & 1 \\ 0 & 0 & 1 & - & 1 \\ 0 & 0 & 0 & 0 & - \end{bmatrix}$$

Figure 1. Priority matrix example

3.2. Schedule builder

- *Giffler and Thompson's parameterized algorithm*.

This option is one of the most classical ones in order to build active schedules [8]. Rather than the initial version, its modified variant was adapted here to produce schedules in a region intermediate to the non-delay and the active classes, by introducing a tunable parameter δ. When δ=0, the algorithm designs a schedule restricted only to the non-delay region; conversely, when δ=1, the whole active schedule class can be described; 0<δ<1 refers to an intermediate case. The δ parameter will be tuned for each treated instance.

Note that Giffler and Thompson's (GT) algorithm needs a variable that settles a conflict at each iteration. For the dispatching rules representation mode, this information is obtained by applying the selected rule. For the two other cases, the computed permutations determine the priority operation.

- *Manual schedule builder*.

This option was designed in the framework of this study. The working mode simply consists in scheduling the operations of each machine according to the order of their defined job permutation. If several operations can be scheduled on different machines, then the one with the lowest completion time is selected. When no operation can be scheduled (because the next operations to be scheduled on each machine do not respect the job sequences), the next operations of the permutation of each machine are considered. The selected operation is scheduled as soon as possible, resulting

Table 1. Computational results for small instances (distance to the optimum, %)

Instance	Size	Optimum (or best known solution)	Giffler and Thompson's algorithm			Manual Schedule Builder	
			Rnd. keys	Disp. rules	Bin. prior.	Rnd. keys	Bin. prior.
FT10	10×10	930	1.40	2.47	1.40	3.76	4.96
FT20	20×5	1165	1.29	3.00	1.12	5.06	6.98
ABZ5	10×10	1234	0.41	0.41	0.41	1.78	2.31
ABZ6	10×10	943	0.53	0.53	0.53	3.71	4.65

in an active schedule. This "manual schedule builder" is obviously adapted for the random keys and binary priorities representations but it is impossible to apply the dispatching rules scheme in this case.

4. Computational experiments

Computational experiments are executed for selected instances drawn from the (abundant) JSP literature. Experiments are divided into two steps: the first one is carried out on simple examples to rule out one schedule's builder version. A second set evaluates the DE efficiency on intermediate/difficult instances.

The set of simple problems is composed of two instances due to [7] (FT10, FT20) and two instances due to [1] (ABZ5, ABZ6). The set of intermediate problems is composed of six instances due to [3] (ORB01-ORB06) and seven instances due to [11] (denoted LAi). Finally, four instances due to [19] (YN1-YN4) constitute the set of difficult problems.

4.1. Results for small examples

All the above permutation encodings and schedule builder versions were tested on this set of problems. For each instance, 20 trials were executed. The DE working parameters mentioned in Section 2.1 were tuned through a preliminary sensitivity analysis phase (not presented here). For all the examples, *F* and *CR* were randomly generated in the intervals [0.3,0.9] and [0.8,1.0]. *NP* and *NG* were set as 250 and 600; the resulting number of 150,000 function evaluations is considerably lower than the values usually adopted in the literature. However, it was noted that increasing *NG* did not provide significant improvements.

The obtained solutions, presented in Table 1, are not compared with respect to any other algorithm, since the aim of this first experiment is to draw some preliminary conclusions about the behavior of DE according to its internal procedures. In fact, state-of-the-art algorithms always find the optimal solutions for these problems.

Columns 4 to 8 indicate, for each DE version, the distance (in %) of the best found solution to the optimum (or best known Upper Bound) provided in column 2. With regard to the choice of a schedule builder, Table 1 clearly shows that the GT algorithm is the best option. Note that the δ parameter implemented in the algorithm was, for the four instances, bracketed betwen 0.15 and 0.3, cutting an important region of the active schedule space. On the one hand, this means that this method might prevent us from finding the optimum (that would explain the distance to the optimal value, between 0.53 and 3.00). On the other hand, additional tests (not shown here) proved that considering a larger part of the active schedules class does not improve the results: for higher δ values, DE seems to drift within this huge search space. Indeed, the manual schedule builder (which describes the entire active schedule class) obtains poor results, far from the optimal values.

In terms of repeatability, the mean value (for 20 trials) distance to the optimum lies in the interval [0.83,2.04] with random keys; [0.53,5.87] with dispatching rules; [0.75,3.16] with binary priorities. So, DE with the GT schedule builder does not work so badly, although it cannot reach the optimum.

Concerning the permutation representations, the random keys and the binary priority version are better than the remaining option, albeit without huge differences. In terms of computational cost, the binary priority representation is much slower than the former ones: the higher number of variables used with this strategy needs computational times that are about twice higher than those needed by the other versions. So, considering both solutions quality and computational cost, the random keys encoding seems to present a slight advantage over the other two. The second instance set will try to confirm this first trend.

4.2. Results for harder instances

For the second set of experiments, the three permutation encodings were evaluated only using the GT algorithm as a schedule builder. The distances to the optimum/upper bound of the best solutions obtained with 20 trials executed for each instance are shown in Table 2. These results are compared, for similar numbers of function evaluations, with respect to: GRASP [2], Hybrid Genetic Algorithm [9] and Nowicki and Smutnicki's tabu search based

Table 2. Computational results for harder instances (distance to the optimum, %)

Instance	Size	Optimum (or best known solution)	Giffler and Thompson's algorithm			Other algorithms		
			Rnd. keys	Disp. rules	Bin. prior.	GRASP	HGA	TSAB
ORB01	10×10	1059	1.04	1.04	1.04	0.00	-	-
ORB02	10×10	888	0.79	0.79	0.90	0.00	-	-
ORB03	10×10	1005	1.59	3.58	1.99	0.00	-	-
ORB04	10×10	1005	1.29	2.19	1.29	0.60	-	-
ORB05	10×10	887	1.01	2.48	1.80	0.23	-	-
ORB06	10×10	1010	0.89	1.88	1.29	0.20	-	-
LA22	15×10	927	1.40	2.59	2.16	0.00	0.86	0.00
LA24	15×10	935	2.14	3.10	2.78	2.03	1.93	0.43
LA25	15×10	977	2.56	2.76	4.09	0.72	0.92	0.00
LA27	20×10	1235	4.78	4.85	4.78	2.75	1.70	0.08
LA37	15×15	1397	2.08	4.58	4.44	0.93	0.79	0.72
LA38	15×15	1196	2.59	4.68	4.18	1.84	1.92	0.00
LA40	15×15	1222	1.96	1.96	1.96	1.80	1.55	0.57
YN1	20×20	885	6.89	7.46	7.91	-	-	Mean
YN2	20×20	909	8.80	7.15	9.13	-	-	Value
YN3	20×20	892	7.74	5.94	7.51	-	-	=
YN4	20×20	968	9.19	7.64	9.50	-	-	1.18

algorithms [14, 15]. Empty cells mean that no solution is available.

The results shown in Table 2 confirm the global superiority of the random keys encoding, although the difference with the binary priorities is not significant. However, a reverse behavior is noticed for more complex instances: the dispatching rules method provides the best solutions. Regarding the global quality, the DE's best solution lies at a distance to the optimum/UB that remains almost steady for intermediate instances, but increases drastically for harder instances. With regard to the other algorithms, DE results are not so far from GRASP and HGA, but cannot compete with those of tabu search.

4.3. Discussion

The proposed experiments highlighted the GT algorithm as the best schedule builder. Concerning encoding techniques, the random keys method has advantages over the other two, although with minor differences. With regard to the simple DE performances for the JSP, compared to other algorithms, the first point is that its working mode obviously does not produce values competitive with those achieved by state-of-the-art approaches such as the *i*-TSAB algorithm. This is not so surprising since *i*-TSAB is carefully tailored for the JSP, particularly because of its adapted neighborhood and its heuristic-based initialization technique. However, except for some harder instances, DE does not provide results so different from the other stochastic approaches (note that HGA also uses a local search process based on the *i*-TSAB neighborhood). This induces the idea of hybridizing DE with a local optimizer in order to improve its performance.

5. Hybridization with a local optimizer

Watson et al. [18] identified *i*-TSAB's most relevant innovations: the N5 neighborhood, a long-term memory and a sequence of diversification-intensification steps. We thus added tabu search (TS) with the N5 neighborhood [14] to our DE-based scheme but without *i*-TSAB long-term memory and diversification-intensification processes. Every 10 generations, a ratio of the DE population (from 2% to 10%) was chosen among the best solutions and sent to TS. In order to avoid premature convergence, the solutions modified by TS are not injected back into the DE population. The DE parameters are kept unchanged but the search is cut when a number of function evaluations similar to that of the previous section is reached. The results from Table 3 show that the proposed hybrid obtains very good results compared to GRASP and HGA.

However, the question might be raised of how important is the DE's influence in such results, i.e. if TS might achieve by itself solutions of equal quality. So, additional experiments were carried out: TS was randomly initialized and run for all the examples. The process is repeated a sufficient number of times to reach the number of function evaluations used by the hybridized DE-TS. The results (not presented here) underline that, especially for the harder instances, the randomly initialized TS results are much worse

Table 3. Computational results obtained by the DE-TS hybrid (distance to the optimum, %)

Instance	Dist. to Optimum	Instance	Dist. to Optimum	Instance	Dist. to Optimum
FT10	0.00	ORB04	0.00	LA37	1.29
FT20	0.00	ORB05	0.22	LA38	0.50
ABZ5	0.00	ORB06	0.30	LA40	0.33
ABZ6	0.00	LA22	0.00	YN1	1.58
ORB01	0.00	LA24	0.96	YN2	1.10
ORB02	0.00	LA25	0.10	YN3	0.90
ORB03	0.00	LA27	0.32	YN4	2.07

than the DE-TS algorithm solutions (on average, solutions are 5.32% above the optimum for the YN instances).

6. Conclusions

Three kinds of permutation encodings and two schedule builders were applied to the JSP in this paper. The computational experiments allowed us to determine the most suitable choices for both mechanisms. However, a comparative study showed that, in its current state, this technique cannot compete with other algorithms, leaving plenty of room for the design of more efficient permutation representation methods. Nevertheless, a hybridization with a local optimizer, produced a drastic improvement on the quality of the solutions obtained. This suggests an interesting path for future research, which are currently exploring.

References

[1] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[2] R. M. Aiex, S. Binato, and M. G. C. Resende. Parallel grasp with path-relinking for job shop scheduling. *Parallel Computing*, 29(4):393–430, 2003.

[3] D. Applegate and W. Cook. A computational study for the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

[4] J. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6:154–160, 1994.

[5] F. Della Croce, R. Tadei, and G. Volta. A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15–24, 1995.

[6] U. Dorndorf and P. E. Evolution based learning in a job shop scheduling environment. *Complex Systems*, 22:25–40, 1995.

[7] H. Fisher and G. L. Thompson. *Probabilistic learning combinations of local job-shop scheduling rules*. Prentice Halls, Industrial Scheduling Edition, Englewood Cliffs, New Jersey (USA), 1963.

[8] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8(4):487–503, 1960.

[9] J. F. Goncalves, J. J. M. Mendes, and R. M. G. C. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167:77–95, 2005.

[10] H. Greenberg. A branch-and-bound solution to the general scheduling problem. *Operations Research*, 16:353–361, 1968.

[11] S. Lawrence. *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement)*. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh (Pennsylvania), USA, 1984.

[12] T. E. Morton and D. W. Pentico. *Heuristic Scheduling Systems: With Applications to Production Systems and Project Management*. Ed. D.F. Kocaoglu. Wiley Series in Engineering & Technology Management, New Jersey (USA), 1993.

[13] R. Nakano and T. Yamada. *Conventional Genetic Algorithm for Job Shop Problems*. Proceedings of the International Conference on Genetic Algorithms (ICGA'91), 1991.

[14] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[15] E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.

[16] R. Storn and K. V. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[17] P. J. M. Van Laarhoeven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:112–129, 1992.

[18] J. P. Watson, A. E. Howe, and W. L. D. Deconstructing nowicki and smutnicki's i-tsab tabu search algorithm for the job-shop scheduling problem. *Computers & Operations Research*, 33(9):2623–2644, 2006.

[19] T. Yamada and R. Nakano. *A Genetic Algorithm applicable to large-scale job-shop instances*. Manner and Manderick (eds.), Parallel instance solving from nature 2, North-Holland, Amsterdam, 1992.