Testing the permutation space based Geometrical Differential Evolution on the Job-Shop Scheduling Problem

Antonin Ponsich and Carlos A. Coello Coello*

CINVESTAV-IPN, Departamento de Computación
Av. IPN No. 2508, Col. San Pedro Zacatenco. México, D.F. 07300
antonin@computacion.cs.cinvestav.mx, ccoello@cs.cinvestav.mx

Abstract. From within the variety of research that has been devoted to the adaptation of Differential Evolution to the solution of problems dealing with permutation variables, the Geometric Differential Evolution algorithm appears to be a very promising strategy. This approach is based on a geometric interpretation of the evolutionary operators and has been specifically proposed for combinatorial optimization. Such an approach is adopted in this paper, in order to evaluate its efficiency on a challenging class of combinatorial optimization problems: the Job-Shop Scheduling Problem. This algorithm is implemented and tested on a selection of instances normally adopted in the specialized literature. The results obtained by this approach are compared with respect to those generated by a classical DE implementation (using Random Keys encoding for the decision variables). Our computational experiments reveal that, although Geometric Differential Evolution performs (globally) as well as classical DE, it is not really able to significantly improve its performance.

1 Introduction

The Differential Evolution (DE) technique is an Evolutionary Algorithm proposed by Storn and Price in the mid 1990s [22]. Characterized by a novel mutation operator, this stochastic search technique has been found to be a powerful optimization tool for solving continuous optimization problems [12]. Unlike other methods such as Genetic Algorithms, the canonical DE scheme is based on a floating-point representation of the variables. Thus, the treatment of discrete optimization problems requires an adaptation of its original operators. This latter observation is even more relevant when dealing with permutation-based problems since such problems involve, in addition to the discrete values restriction, inherent constraints of interdependence among variables.

A significant amount of research has been recently devoted to this issue and many techniques have been proposed, particularly focusing on methods for converting real variables into permutations. However, none of all these techniques

* The second author acknowledges support from CONACyT project no. 103570.

has succeeded in avoiding the redundancy in the mapping from real to permutation spaces. As a consequence, the results obtained by DE are not able to satisfactorily compete with those obtained by other metaheuristics when tackling complex permutation-based problems. The other path for adapting DE to problems dealing with permutation variables is the modification of the DE's operators. In 2009, A. Moraglio et al. adapted a geometric framework (previously introduced for Particle Swarm Optimization [13]) for its use with the DE metaheuristic. The differentiation was transformed into a geometric operation, applicable in any space when adopting an appropriate metric. In [14], the authors mainly focused on the consideration of binary spaces and reported interesting results for NK landscapes and Spears-DeJong functions. In a recently released technical report [15], permutation and program spaces are further tackled. The aim of the present paper is thus to evaluate the behavior of such a technique, called *Geometric Differential Evolution* (GDE) for permutation spaces, on a challenging problems class.

As a case study, the Job-Shop Scheduling Problem is chosen, for two main reasons. First, there is no need to emphasize its inherent complexity, already evidenced in many studies and simply justified by the inability for state-of-the-art algorithms to identify optimal solutions of complex instances of this problem. The second reason is a previous work on the JSSP, which analyzed the performance of techniques based on the transformation of real numbers to permutations [20]. This precedent will allow us to compare the two strategies, i.e., transforming either the variable representation mode or the operators. The remainder of this paper is organized as follows. Section 2 presents a short overview on JSSP, while Section 3 is dedicated to the definition of the permutation-based GDE algorithm. The experimental methodology and computational results are presented in Section 4. Finally, some conclusions are drawn in Section 5.

2 Overview of the JSSP

2.1 A review on solution techniques

Scheduling problems, because of their many applications not only in the industrial but also in the service fields [19], have attracted an increasing interest within the Operations Research community. In the manufacturing area, the Job-Shop Scheduling Problem is one of the most complex examples. In this problem, a set of jobs, which all consist of several operations, is processed in a certain order on a set of machines. The processing sequence of each job operations on the machines and the associated processing times are the problem data. The objective is, then, to minimize the completion date of the last scheduled operation.

Because solving exactly the JSSP constitutes a challenging issue, much of the research efforts have focused on the development of efficient methods. Researchers have first concentrated on exact optimization techniques based on the disjunctive graph representation [21]. However, the JSSP is hard for exact algorithms and optimal solutions can be provided for instance sizes up to about 10 machines and 10 jobs. This feature has led to the development of heuristics

methods: the Shifting Bottleneck heuristic (particularly the SB-I and SB-II versions presented in [1]) is an example of a heuristic which achieves very good results even for mid-size and large instances.

With the development and enhancement of several metaheuristics, a variety of local search and population-based heuristics have been applied to the JSSP: Simulated Annealing [24], Tabu Search [23], Genetic Algorithms [6], GRASP [2], etc. Aiming at simultaneously exploiting the benefits of several methods, hybrid techniques have also been frequently used: Genetic Local Search [9], Ant Colony Optimization coupled with a Tabu Search approach [10], a Tabu Search/Simulated Annealing hybrid [26]. The two Tabu Search based algorithms proposed by Nowicki and Smutnicki (TSAB [17] and i-TSAB [18]) have provided the best results reported until now for this problem.

2.2 Problem formulation

The classical JSSP aims at assigning a finite set O of operations to a finite set M of machines ($|M| = m$). Each operation $o_{ij} \in O$ belongs to the sequence of a job j and must be processed on a specific machine i. Conversely, each job $j \in J$ is characterized by a subset of operations $O_j \subset O$ that must be processed according to a defined order. Unlike the Flow-Shop case, the processing sequence differs from one job to another. In most cases, each job j is processed exactly once on each machine i, so that the total number of operations is $|O| = nm$ and the commonly used nomenclature refers to $n \times m$-instances.

The typical objective of the classical JSSP is then to minimize the completion time of the last operation scheduled, namely the makespan, while respecting the following major constraints: (i) one machine cannot simultaneously process more than one job at a time, (ii) preemption is not allowed, (iii) the processing sequence of the operations belonging to a job must be respected (the starting time of any operation is higher than the completion time of its predecessor).

The solution is an assignment of operations to machines on a precise time period and is called a *schedule*. When an appropriate schedule builder is used (see next section), this solution can be formulated as a multi-permutation, i.e., a set of job permutations associated to each machine. The cardinality of all possible solutions to the JSSP is therefore $(n!)^m$.

2.3 Schedule classes

Attention must be paid to the fact that, for a given multi-permutation, an infinite number of schedules might be built. An important issue therefore concerns the construction of the schedule. Among the feasible schedules, 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 [16]. Another relevant schedule class is the *non-delayed* one, for which no idle time is allowed on a machine if this latter is free and an operation is available for processing. This latter class also constitutes a subset of the active schedules class, but may not contain the optimal solution.

In the following, a schedule builder based on the parameterized Giffler & Thompson's algorithm is used [8]. This technique produces schedules that lie on an region intermediate between the active and non-delayed classes, through the introduction of a parameter δ that controls the number of possible schedules, i.e., the number of solutions in the search space of the considered problem. When δ=0, the built schedule is non-delayed while δ=1 allows the generation of active schedules. This parameter should be tuned for each treated instance.

3 Geometric DE for multi-permutation spaces

The aim of this section is not presenting the whole theoretical framework and detailed insights of the geometric adaptation of Differential Evolution to permutation spaces, since such information can be found in [15]. We only provide here the features necessary for a global understanding and for the implementation of permutation-based GDE.

3.1 A geometric framework for DE

The basic idea in GDE is a re-interpretation of the evolutionary operators according to a geometric point of view. Considering two vector solutions as points in the space, the crossover of these two parents can be seen, in this sense, as a geometric operation returning a point within the segment defined by the two original solutions. The distance between the offspring vector and both parents is then implicitly determined by the crossover rate. Clearly, in this context, the definition of an appropriate metric associated to the considered seach space is required; but this allows the extension of such concepts to any space endowed with an adequate distance.

In [14], the reformulation of the DE operators is proposed according to this geometric paradigm. Consider the classical DE mutation operator:

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

where u_{ij}^G is the mutated offspring generated for parent x_{ij}^G, while x_{1j}^G, x_{2j}^G, x_{3j}^G are randomly selected individuals in the current population ($x_{1j}^G \neq x_{2j}^G \neq x_{3j}^G \neq x_{ij}^G$) and F ($F \in$ [0,1]) is the scaling factor.

Setting $W = \frac{1}{1+F}$, then equation (1) can be written as:

$$W \cdot u_{ij}^G + (1 - W) \cdot x_{2j}^G = W \cdot x_{3j}^G + (1 - W) \cdot x_{1j}^G, \forall j \in \{1, ..., N\} \tag{2}$$

Consider that *U*, *X1*, *X2* and *X3* are the point-wise representations associated to vectors u_{ij}^G, x_{1j}^G, x_{2j}^G and x_{3j}^G, respectively. The differentiation operator can be represented as illustrated in Fig. 1 (taken from [14]): point *E* is the result of the convex combination of points *X3* and *X1* with weights W and $1 - W$ respectively, and can be constructed since *X1* and *X3* are known. Subsequently, point *U* can be deduced as the point on the extension ray (*ER*) going out of *X2*

and passing through E (E is the result of the convex combination of points U and $X2$ with weights W and $1 - W$, respectively). Thus, creating algorithmic procedures that translate the convex combination and extension ray operations would allow to design a geometric interpretation of the differentiation operator based on the definition of a metric adapted to the considered space.

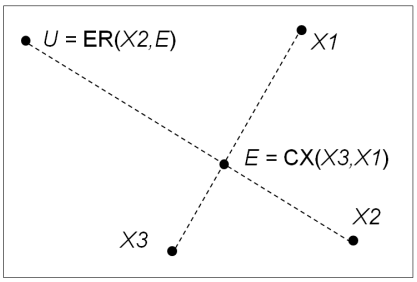

Fig. 1. Construction of U using convex combination and extension ray procedures

Similarly, regarding the crossover procedure, Moraglio et al. prove that the discrete recombination applied to u_{ij}^G and x_{ij}^G (or, according to the geometric paradigm, points U and X_i) produces an offspring that can be seen as the convex combination of points U and X_i with weights C_r and $1-C_r$ respectively, where C_r is the crossover rate used within DE.

3.2 GDE for the JSSP

In order to adapt the GDE framework to the job-shop scheduling problem, a metric must be defined for permutation spaces. The distance between two configurations will subsequently be computed as the sum of the distances between the permutations, in each schedule, associated to every machine. This is proposed in [15]. The swap distance appears to be particularly appropriate for the JSSP: swapping two jobs in a sequence is a neighborhood definition commonly used within many local search methods. The swap distance between two sequences is then equal to the number of swapping mutation steps necessary to transform one permutation into another.

In other words, let us consider that the permutation vectors constitute the nodes of an undirected graph whose edges represent a swapping move between two neighbor permutations (i.e. one permutation is attainable from the other by only one swapping mutation step). The swap distance is thus the shortest path, on this graph, to get from one vector to another. Note that this concept involves, for a permutation having n elements, a total "diameter" of the search space (maximum distance between two permutations) equal to $n - 1$.

Accordingly, for the convex combination procedure, the swap distance between both initial vectors is computed and the new vector is derived in such a

way that it lies between the initial strings, at a distance obtained from multiplying each weight by the total swap distance. These distances are interpreted as probabilities in order to construct a mask that determines, for each position, to which parent element the offspring must be equal to. A similar procedure is developed for the extension ray, in such a way that the convex combination of the new vector and one of the initial ones results in the other parent. When the distance between the initial strings is equal to the diameter of the search space, it is impossible to generate an offspring farther away from one of them (outside the segment defined by the initial vectors). Both of these processes are repeated for each permutation in the schedule configuration (i.e., for every machine).

The detailed algorithms for all the above-mentioned procedures (computing swap distances, convex combination and extension ray) are not presented here but are explained in detail in [15].

4 Computational experiments

4.1 Methodology

The permutation-based GDE algorithm is evaluated on a selection of instances chosen among several sets of JSSP instances commonly used in the specialized literature:

- 3 instances due to [7]: FT06, FT10, FT20.
- 2 instances due to [1]: ABZ5, ABZ6.
- 6 instances due to [3]: ORB01-ORB06.
- 7 instances due to [11]: LA22, LA24, LA25, LA27, LA37, LA38, LA40.
- 4 instances due to [25]: YN1-YN4.

These examples, drawn from the OR-library [5], can be basically divided into three groups, according to their complexity: easy (FT and ABZ), medium (ORB and LA) and difficult instances (YN).

With respect to the GDE parameters tuning, population size NG and generation number NG are set in such a way that the number of objective evaluations $NP \times NG$ is equal to the values commonly reported in the literature (note, however, that this criterion may not always be fair since, in many cases, a simplified objective computation mode is devised in order to shorten computational times). Concerning the crossover rate C_r and the amplification factor F, preliminary computations indicated that the best results are obtained when the former adopts rather high values (between 0.8 and 1) and when the latter adopts values randomly generated between 0.3 and 0.9.

4.2 Standard random-keys DE

As mentioned before, the aim of this study is to compare both strategies when adapting DE to problems dealing with permutation variables, i.e., adapting the variable representation mode or the internal DE's operators. So, according to

this, the results obtained by the described GDE algorithm are compared here against those of a standard DE.

The DE version used here, fully described in [20], is DE/rand/1/bin, which means that the vectors used within the differentiation operator are randomly selected, only one difference is used and a binary crossover is performed for each variable independently. Besides, the main feature of the algorithm is the encoding procedure. In order to handle permutations, the Random Keys method, initially proposed in [4], was implemented. A real number, bounded between 0 and 1, is used for each operation and operations corresponding to the same machine are sequenced according to the increasing order of their associated variable. For instance, considering 5 jobs with the following variable vector on machine i: [0.41 0.68 0.02 0.85 0.37], the resulting sequencing order is: [2 4 0 1 3]. The Random Keys technique is chosen here because it proved to be more efficient and effective than two others (i.e., evolving dispatching rules and binary matrix priority based on the disjunctive graph) as indicated in the comparative study reported in [20].

Besides, since the differentiation mutation is likely to produce variables lying outside their bounds, a mixed constraint-handling technique is applied according to a given probability P_B (tuned for each instance): (i) setting the variable value to the violated bound; (ii) using the violated bound as a symmetry center to send the considered variable to the feasible side of the boundary.

4.3 Results

For each instance, we performed 20 runs of each method (classical DE and GDE). The comparison is obviously drawn for equal numbers of objective evaluations. The values reported in Table 1 indicate, for each technique, Makespan Relative Errors (with respect to a reference makespan) and standard deviation of the results over the 20 runs. *b-MRE* (respectively, to *m-MRE*) reports the error of the best objective value found over 20 runs f_{Best} (respectively, the mean value of the objective over the 20 runs f_{Mean}). The reference makespan is, typically, a lower bound LB (optimal for all instances, except for YN1-YN4). Note that the standard deviations provided in Table 1 are computed according to the makespan values and not to the relative errors. The relative error is computed according to the following formula:

$$b - MRE = 100 \times \frac{f_{Best} - LB}{LB} \tag{3}$$

$$m - MRE = 100 \times \frac{f_{Mean} - LB}{LB} \tag{4}$$

A global observation of Table 1 shows a similarity of both techniques' performance. Regarding the *b-MRE* indicator, both methods provide similar results for 36% of the treated instances. With respect to *m-MRE*, GDE is better in 45% of the considered cases while the Random Keys DE is better for 50%.

However, some differences appear when considering separately each problem category. On the one hand, GDE provides the best results for simple and complex instances and, on the other hand, the Random Keys DE performs better

Table 1. Computational results

Instance	Size	LB/Opt.	Geometric DE			Random Keys DE		
			b-MRE	*m-MRE*	S. Dev.	*b-MRE*	*b-MRE*	S. Dev.
FT06	6×6	55	0	0.18	0.31	0	0.18	0.44
FT10	10×10	930	1.40	2.33	4.50	1.40	1.97	5.07
FT20	20×5	1165	1.12	1.12	0.45	1.29	1.45	0.34
ABZ5	10×10	1234	0.41	0.44	1.47	0.41	0.84	4.87
ABZ6	10×10	943	0.53	0.53	0	0.53	0.73	4.97
	Average		**0.69**	**0.92**	**1.35**	**0.72**	**1.04**	**3.14**
ORB01	10×10	1059	1.04	1.08	1.79	1.04	1.11	2.40
ORB02	10×10	888	0.68	0.92	1.44	0.79	0.99	1.64
ORB03	10×10	1005	1.59	3.86	9.37	1.59	2.54	6.73
ORB04	10×10	1005	1.09	1.94	3.99	1.29	1.92	4.22
ORB05	10×10	887	0.79	1.89	2.05	1.01	1.91	4.36
ORB06	10×10	1010	1.09	1.78	4.70	0.89	1.89	6.38
LA22	15×10	927	3.67	4.95	5.59	1.40	3.67	7.65
LA24	15×10	935	3.21	4.51	4.85	2.14	2.83	3.56
LA25	15×10	977	3.48	5.32	5.83	2.56	3.50	6.12
LA27	20×10	1235	5.67	7.10	7.28	4.78	6.44	8.85
LA37	15×15	1397	3.58	4.82	5.08	2.08	3.57	9.75
LA38	15×15	1254	4.85	6.59	7.96	3.01	3.71	5.30
LA40	15×15	1222	4.09	5.11	5.96	1.96	2.59	5.04
	Average		**2.68**	**3.83**	**5.07**	**1.89**	**2.82**	**5.54**
YN1	20×20	846	12.06	14.02	6.10	13.36	14.59	5.40
YN2	20×20	870	12.64	14.86	7.42	13.86	15.23	6.73
YN3	20×20	840	13.45	14.73	5.47	14.40	15.48	4.60
YN4	20×20	920	14.13	16.01	5.50	14.89	15.49	4.01
	Average		**13.07**	**14.91**	**6.12**	**14.08**	**15.20**	**5.18**

for medium instances. This behavior is further moderated for medium instances since GDE is slightly better for the ORB class while being completely outperformed by the Random Keys DE in the LA class. This comment highlights the fact that understanding why some instances are more difficult for one method than for another still remains as an open question.

It is worth recalling that state-of-the-art algorithms, such as TSAB and i-TSAB (complete results available in [17] and [18]) generally perform much better than both of the DE versions considered here (specially on hard instances but also for the simple ones, for which optimal solutions can be systematically determined). This comment must balance the previous observations concerning the quality of the obtained results.

The deceiving results obtained by GDE could be tentatively explained by the extension ray procedure used within the geometrically adapted differentiation. As underlined in [15], this operation consists in generating point U (the mutant, see Fig. 1) beyond point E, which is itself computed as the convex com-

bination of points $X1$ and $X3$. However, in many cases, the distance between $X2$ and E is already equal to the maximum distance between two permutations (called the "search space diameter" in [15]). As a consequence, point U is equal to point E, meaning that the differentiation operator degenerates into the simple convex combination of two parents $X1$ and $X3$: the explorative power of the method is thus significantly damaged. Note that this phenomenon does not occur for smaller alphabets (binary search spaces) because the probability to generate two points separated by the maximum distance is much lower than in the permutation case. Finally, this observation shows that GDE might not provide competitive results for high cardinality alphabets, especially if the variable string size is big.

5 Conclusions

We presented in this study an evaluation of the Geometric Differential Evolution algorithm, adapted to the treatment of problems dealing with permutation variables, recently proposed in [15]. The Job-Shop Scheduling Problem has been adopted because it represents a challenging problem class for which a widely developed bank of instances is available. The results showed that the permutation-based GDE globally performs as well as a classical implementation of DE for this problem class. This may lead to the conclusion that this novel algorithm, based on a modification of the initial DE's operator, is really viable and able to compete with a DE version having a special variables encoding mechanism for the treatment of permutation-based problems. However, GDE is not able to significantly improve the performance of classical DE for non-continuous optimization problems and would be still outperformed by other metaheuristics (particularly Tabu Search in the JSSP framework). Thus, more research is required in order to produce a more competitive DE variant for problems such as JSSP.

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. J.E. Beasley. Or-library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. http://mscmga.ms.ic.ac.uk.
6. F. Della Croce, R. Tadei, and G. Volta. A Genetic Algorithm for the job shop problem. *Computers and Operations Research*, 22(1):15–24, 1995.
7. H. Fisher and G.L. Thompson. *Probabilistic learning combinations of local job-shop scheduling rules*, pages 225–251. Industrial Scheduling Edition. Prentice Halls, 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. de Magalhaes Mendes, and M.G.C. Resende. A Hybrid Genetic Algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1):77–95, 2005.
10. K.L. Huang and C.J. Liao. Ant Colony Optimization combined with Taboo Search for the job shop scheduling problem. *Computers and Operations Research*, 35(4):1030–1046, 2008.
11. S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh (Pennsylvania), USA, 1984.
12. E. Mezura-Montes, Velázquez-Reyes J., and C.A. Coello Coello. Modified Differential Evolution for constrained optimization. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC'06)*, pages 25–32, Vancouver, Canada, 2006.
13. A. Moraglio, C. Di Chio, and R. Poli. Geometric Particle Swarm Optimization. *European COnference on Genetic Programming*, 125-136, 2007.
14. A. Moraglio and J. Togelius. Geometric Differential Evolution. *Genetic and Evolutionary Computation Conference (GECCO'09)*, 1705-1712, 2009.
15. A. Moraglio, J. Togelius, and S. Silva. Geometric Differential Evolution for Combinatorial and Programs Spaces. Technical Report, School of Computing, University of Kent, Canterbury (UK), March 2010.
16. 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.
17. E. Nowicki and C. Smutnicki. A fast Taboo Search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.
18. E. Nowicki and C. Smutnicki. An advanced Tabu Search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.
19. M.L. Pinedo. *Planning and scheduling in manufacturing and services.* Springer Series in Operations Research. Springer, New-York, USA, 2005.
20. A. Ponsich, Ma.G. Castillo Tapia, and C.A. Coello Coello. Solving permutation problems with Differential Evolution: an application to the jobshop scheduling probem. In *Ninth International Conference on Intelligent System Design and Applications (ISDA'09)*, Pisa, Italy, November-December 2009.
21. B. Roy and B. Sussmann. *Les problèmes d'ordonnancement avec contraintes disjonctives.* Note DS 9 bis, SEMA, Paris, 1964.
22. 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.
23. E.D. Taillard. Parallel Taboo Search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.
24. P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by Simulated Annealing. *Operations Research*, 5(1):113–125, 1992.
25. T. Yamada and R. Nakano. A Genetic Algorithm applicable to large-scale job-shop instances. In Manner and Manderick, editors, *Parallel instance solving from nature 2*, pages 281–290. North-Holland, Amsterdam, 1992.
26. C.Y. Zhang, P. Li, Y. Rao, and Z. Guan. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers and Operations Research*, 35(1):282–294, 2008.