Computers & Operations Research 67 (2016) 174–183

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/caor

Hybrid method with CS and BRKGA applied to the minimization of tool switches problem

A.A. Chaves [a,*], L.A.N. Lorena [b], E.L.F. Senne [c], M.G.C. Resende [d]

[a] Federal University of São Paulo, São José dos Campos 12231-280, Brazil
[b] National Institute for Space Research, São José dos Campos 12201-970, Brazil
[c] São Paulo State University, Guaratinguetá 12516-410, Brazil
[d] Amazon.com, Mathematical Optimization and Planning (MOP), Seattle 98109, USA

ARTICLE INFO

Available online 24 October 2015

Keywords:
Hybrid heuristics
Clustering search
Genetic algorithm
Scheduling
Tool switches

ABSTRACT

The minimization of tool switches problem (MTSP) seeks a sequence to process a set of jobs so that the number of tool switches required is minimized. The MTSP is well known to be NP-hard. This paper presents a new hybrid heuristic based on the Biased Random Key Genetic Algorithm (BRKGA) and the Clustering Search (CS). The main idea of CS is to identify promising regions of the search space by generating solutions with a metaheuristic, such as BRKGA, and clustering them to be further explored with local search heuristics. The distinctive feature of the proposed method is to simplify this clustering process. Computational results for the MTSP considering instances available in the literature are presented to demonstrate the efficacy of the CS with BRKGA.

1. Introduction

The minimization of tool switches problem (MTSP) considers a set of jobs $T = \{1, \ldots, N\}$ to be processed on a single machine. Let $F = \{1, \ldots, M\}$ be the set of tools available for this machine and T_f the set of jobs that require the tool $f \in F$. Each job $t \in T$ requires a subset of tools $F_t \in F$ and can only process this job when all of this subset of tools is on the machine. Consider that the machine is capable of holding a maximum of C tools at a time, and $C \geq \max_t\{|F_t|\}$. It is assumed that C is less than the total number of tools required to process all jobs. Thus, tool switches are necessary, i.e., remove a tool from the machine and add another in its place. In the MTSP we are seeking a sequence to process a set of jobs so that the number of tool switches required is minimized.

MTSP is NP-hard [1,2] and it has been studied by several authors, mostly through heuristics. Tang and Denardo [1] show that the MTSP for a given sequence of jobs can be solved in polynomial time by a policy known as KTNS (Keep Tool Needed Soonest). This policy states that when tool switches are required, the first tools required for the next job should be the first to be held in the machine.

Crama et al. [2] propose heuristics, based on heuristics for the traveling salesman problem, to solve the MTSP. Hertz et al. [3] also present and compare heuristics based on the traveling salesman problem, but the authors use a more appropriate definition of the “distance” between tools to be minimized. This improvement provides superior performance over the heuristics of Crama et al. [2].

Matzliach and Tzur [4] present three heuristics for the MTSP with non-uniform tool sizes in a dynamic (online) environment. The proposed heuristics are based on the static problem and consider various assumptions with respect to the randomness of the process.

Shirazi and Frizelle [5] assess the efficiency of the methods currently employed by seven manufacturing companies for solving the tool switches problem and compare these methods with some available tool switching heuristics.

Fathi and Barnette [6] propose three heuristic procedures for solving the problem of scheduling a set of parts with given processing times and tool requirements on m identical parallel machines. The paper shows that these heuristics are effective to find a tool-switching plan for each machine in order to minimize the makespan.

Song and Hwang [7] propose an optimal tooling policy utilizing the concept of early insertions of tools in order to minimize the frequency of movements of the tool transporter for a flexible machine. This machine must process a set of parts with its production sequence already prescribed.

Ghrayeb et al. [8] consider the problem of scheduling printed circuit packs on sequencers. The authors present a mathematical model and a fast heuristic to solve this problem. The proposed

* Corresponding author.
E-mail addresses: antonio.chaves@unifesp.br (A.A. Chaves), lorena@lac.inpe.br (L.A.N. Lorena), elfsenne@feg.unesp.br (E.L.F. Senne), resendem@amazon.com (M.G.C. Resende).

http://dx.doi.org/10.1016/j.cor.2015.10.009
0305-0548/© 2015 Elsevier Ltd. All rights reserved.

heuristic is effective in reducing the number of changeovers of input tapes and can be used for the MTSP.

Al-Fawzan and Al-Sultan [9] present a tabu search algorithm to solve the MTSP. Senne and Yanasse [10] present three variants of the Beam Search algorithm for the MTSP. The authors adopt a depth first strategy of the enumeration tree and a scheme that considers partially ordered job sequences. All approaches found good results for the 1350 instances tested.

Konak and Kulturel-Konak [11] propose an Ant Colony Approach to minimize the number of tool switching instants, when the tool switch time is independent of the number of tool switches. The algorithm was applied to solve large sized instances of practical importance. Konak et al. [12] apply two Tabu Search approaches to solve this problem and show that they find solutions close to optimal in reasonable times.

Amaya et al. [13] present a memetic algorithm that combines Genetic Algorithm and local search heuristics. A hill climbing heuristic is used just after the mutation operator on every new individual generated. Computational tests show that hybrid evolutionary approaches are effective to solve the MSTP. Later, Amaya et al. [14] study three cooperative methods with local search mechanisms and one cooperative method with a model based on heterogeneous techniques. The last approach provides better solutions than those found by the Beam Search and Tabu Search. Also, Amaya et al. [15] combine a Genetic Algorithm with three different local search heuristics: hill climbing, Tabu Search and Simulated Annealing. The memetic algorithm with hill climbing found the best results.

Chaves et al. [16] present a new heuristic for the MTSP. This heuristic has a constructive phase, based on a graph where the nodes correspond to tools and an arc links two nodes if and only if these tools are required to execute some job. An additional improvement phase is based on Iterated Local Search (ILS).

The success in solving the MTSP exactly is limited to small instances. Laporte et al. [17] report that just a few instances among a set of 25 jobs instances were solved to optimality using the branch-and-bound scheme they proposed. Yanasse and Rodrigues [18], Yanasse et al. [19] present an enumeration algorithm based on partial orders that obtained good results on instances in which the algorithm of Laporte et al. [17] failed. The computational results of Chaves et al. [16] show that the hybrid heuristic contributes to a significant reduction in the number of nodes in the tree of the enumeration algorithm.

This paper presents a new application of the hybrid method Clustering Search (CS) [20] to solve the MTSP. The CS detects promising areas of the search space using a metaheuristic that generates solutions to be clustered. These promising areas should be explored with local search heuristics as soon as they are discovered. A Biased Random Key Genetic Algorithm (BRKGA) [21] was chosen to generate solutions for the clustering process. A BRKGA encodes a solution as a vector of random keys and produces a feasible solution through the decoder. The use of the BRKGA made it possible to simplify some components of the CS. The users have at their disposal a robust method in which they need to implement only the decoder and local search heuristics. The computational results are compared with other methods found in the literature.

The remainder of the paper is organized as follows. Section 2 describes the basic ideas of CS and BRKGA. In Section 3 we introduce the new approach, describing in detail the BRKGA and CS applied to the MTSP. Section 4 reports computational experiments and Section 5 makes concluding remarks.

2. Methods

In this section we present the basic ideas of the CS and the BRKGA, including descriptions of the solution encoding and decoding, clustering process and local search.

2.1. Clustering search

The Clustering Search (CS) [20] is a hybrid method which combines metaheuristic-based heuristics and local search heuristics. The search is intensified only in areas of the search space that deserve special attention (promising regions). The CS introduces intelligence and priority to the choice of solutions on which to apply local search, instead of randomly choosing or applying local search to all solutions. Therefore, an improvement is expected in the convergence process associated with a decrease in computational effort through a more rational employment of the heuristics.

The CS divides the search space in regions called *clusters*. A solution center, c, represents the location of a cluster. This center is, generally, initialized at random and tends to progressively traverse promising points in the search space. The number of clusters $\mathcal{NC}$ is defined a priori.

The value of $\mathcal{NC}$ has no influence on the amount of local search performed by the CS. Thus, there is little impact on computational time. However, with a larger number of clusters, CS can efficiently discover more promising regions and intensify the search in only those regions.

Chaves [22] analyzes the behavior of CS with different numbers of clusters. The author shows that a number of clusters in the interval [10, 25] provides good results in solution quality and increases the efficiency of local search.

The CS consists of four components: the search metaheuristic (SM), the iterative clustering (IC), the analyzer module (AM), and the local searcher (LS). Fig. 1 shows the conceptual design of these components.

The SM component can be implemented by any optimization algorithm that generates diversified solutions in the search space. It must work as a full-time solution generator, exploring the search space by manipulating a set of solutions, according to its specific search strategy.

Solutions s_k generated by the SM are sent to the IC. The IC gathers similar solutions into groups, maintaining a representative cluster center for each group. A distance metric, Δ, must be defined to provide a similarity measure for the clustering process. For example, in combinatorial optimization, the similarity can be defined as the number of movements needed to change a solution into the cluster center [20].

An assimilation process is applied over the closest center c_i to each newly generated solution s_k. The assimilation can assume two different forms: crossover and path [20]. In crossover assimilation, we alter one or more positions in the center c_i with information derived from solution s_k, resulting in a new center. Path assimilation can generate several solutions, keeping the best one to be the new center. These exploratory moves are commonly used in path relinking [23].

The AM component examines each cluster when it is active, indicating a probable promising cluster. A cluster density, δ_j, is a measure that indicates the activity level inside the cluster j. For simplicity, δ_j can count the number of solutions generated by SM and grouped into cluster j. Whenever δ_j reaches a certain threshold λ, i.e. some information template becomes predominantly generated by SM, such cluster must be further investigated to accelerate the convergence process in it.

If the value of λ is large, the LS component will be applied a few times, while small values of λ results in many local searches. This

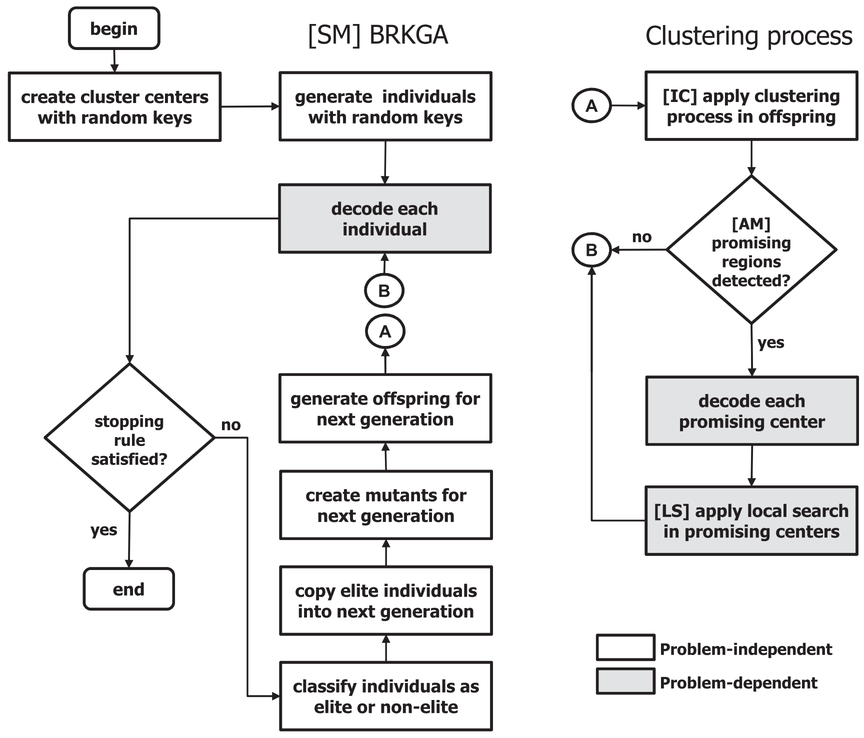
begin
[SM] BRKGA
Clustering process
create cluster centers with random keys
generate individuals with random keys
A
[IC] apply clustering process in offspring
decode each individual
B
no
[AM] promising regions detected?
B
A
yes
generate offspring for next generation
stopping rule satisfied?
no
decode each promising center
create mutants for next generation
yes
[LS] apply local search in promising centers
end
copy elite individuals into next generation
Problem-independent
classify individuals as elite or non-elite
Problem-dependent

Fig. 1. Components of CS with BRKGA.

parameter has to be tuned before the execution of the CS. The parameter λ may have a strong influence on the efficiency and effectiveness of the search as well as on computational time.

Finally, the LS component is an internal searcher module that provides the exploitation of a potentially promising search area, represented by a cluster.

2.2. *Biased Random Key Genetic Algorithm*

The Biased Random Key Genetic Algorithm (BRKGA) is a recent metaheuristic proposed by Gonçalves and Resende [21]. It has been used to solve sequencing and optimization problems. In the BRKGA, solutions are encoded as vectors of randomly generated real numbers in the interval [0, 1]. These numbers are called *random keys*.

The *decoder*, a deterministic algorithm, takes as input a solution vector and associates with it a feasible solution of the problem for which an objective value or fitness can be computed [24]. Thus, points in the random keys space are mapped to points in the problem space for evaluation [25].

A BRKGA evolves a population of random-key vectors, or individuals, over a number of generations. The initial population is made up of p real n-vectors of random keys. Each component of an initial solution vector is generated randomly and independently in the real interval [0, 1]. The fitness of each individual is computed by the decoder. Then, the population is partitioned into two groups of individuals: a small group of p_e elite individuals, i.e., those with the best fitness values, and the remaining set of $p-p_e$ non-elite individuals. To evolve the population, a new generation of individuals must be produced. All elite individuals in the population of generation k are copied without modification to the population of generation $k+1$ [21].

The BRKGA implements mutation by introducing mutants into the population. A mutant is simply a vector of random keys generated in the same way as an individual in the initial population. At each generation, a small number (p_m) of mutants are introduced into the population. With the p_e elite individuals and the p_m mutants accounted for in population $k+1$, $p-p_e-p_m$ additional individuals are needed to produce the p individuals that make up the new population. This is done by producing $p-p_e-p_m$ offspring through the process of crossover, by combining pairs of individuals of the current population.

The *parameterized uniform crossover* [26] is used in BRKGA. Let ρ_e be the probability that an offspring inherits the vector component of its elite parent. Let n denote the number of components in the solution vector of an individual. For $i=1,\ldots,n$, the ith component of the offspring vector takes on the value of the ith component $e(i)$ of the elite parent e with probability ρ_e and the value of the ith component $n_e(i)$ of the non-elite parent n_e with probability $1-\rho_e$ [27]. Parameter ρ_e is always greater than 0.5.

The important feature of random keys is that all offspring formed by crossover are feasible solutions. Toso and Resende [28] concluded that the BRKGA has two distinct parts: one consisting of the genetic algorithm with its chromosome methods and generations, called *problem-independent* and the other consisting of the decoder, called *problem-dependent*.

2.3. *CS with BRKGA*

In this paper we propose to use the BRKGA as the SM component of CS. Thus, we seek to make the implementation of CS more independent in relation to the optimization problem. We use a random-key vector to represent a solution. Therefore, the distance metric is the Euclidean distance and the assimilation process is performed over the random-key vector. The local search (LS) is the only component that works with decoded solutions. Fig. 1 shows the conceptual design of CS with BRKGA.

Initially, we define a number of clusters $\mathcal{NC}$. Their centers are generated with random keys by the method used in the BRKGA to generate the initial population. This set of cluster centers is not part of the population of BRKGA, and it evolves separately.

The BRKGA generates solutions for the clustering process. After each generation, the offspring are analyzed and clustered. The IC component determines a *distance metric* to compute the similarity between a given solution and a cluster center. With centers represented by random keys, the similarity is based on the Euclidean distance. Thus, we avoid the cost of computing the number of movements needed to change a solution into the cluster center.

The assimilation process is applied over the closest center, by adding the new solution to this cluster, thus causing an update of the center. In this paper, we use a path assimilation. Then, we generate several solutions keeping the best evaluated solution to be the new center. These exploratory moves are commonly referred to in path relinking [23]. In the case of random key vector, a move consists of replacing one of its component. The process terminates when a percentage of solutions in the path have been analyzed.

After performing the assimilation process, we conduct an analysis of the density δ_j (AM component), verifying if this cluster can be considered promising. A cluster becomes promising when its density reaches the threshold λ ($\delta_j \geq \lambda$).

The local search of CS (i.e. the LS component) intensifies the search in the neighborhood of a promising cluster center. The center is first decoded into a solution of the problem and then specific local search heuristics are applied to find the best possible solutions in the corresponding region.

Changes in the solution made by the local search need to be taken into account in the new cluster center. Then, the cluster center is adjusted to reflect these changes carrying out the inverse steps taken by the algorithm used to decode a random-key solution.

3. CS with BRKGA applied to MTSP

In this paper, we encode a solution to the problem as a vector of random keys that is later used by a decoding procedure to obtain a solution. A solution to the MTSP is represented indirectly by the following solution structure:

$$solution = (k_1, \ldots, k_n)$$

where n is the number of jobs. The decoding of the n random keys $k_1, \ldots, k_n$ of each solution into a sequence of jobs is accomplished by sorting the jobs in ascending order of their corresponding random keys values. Fig. 2 shows an example of the decoding process for the MTSP. In this example there are five jobs. The sorted random keys correspond to the jobs sequence (3, 5, 1, 2, 4).

The fitness of a solution with the job sequence is obtained with the KTNS algorithm [1]. It returns the total number of tool switches needed to process the jobs in this sequence. The pseudocode of the KTNS is presented in Appendix A.

The initial clusters of CS and the initial population of BRKGA are made up of vectors with n random keys. Each component of the solution vector, i.e. each random key, is generated independently at random in the real interval [0, 1].

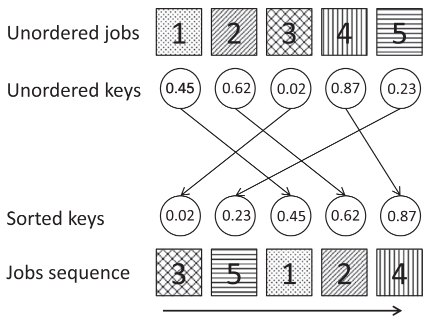

Fig. 2. Decoding of the job sequence.

This paper uses a greedy method based on maximum diversity to create the initial centers. It generates a large set with q ($q \gg \mathcal{NC}$) random key vectors and a subset of cardinality $\mathcal{NC}$ with the largest diversity selected based on Euclidean distance. A possible example of initial clusters (with $\mathcal{NC} = 3$ and $n = 5$) is

$$c_1 = (0.45, 0.62, 0.02, 0.87, 0.23) \equiv (3, 5, 1, 2, 4) \quad \delta_1 = 0,$$

$$c_2 = (0.71, 0.12, 0.89, 0.38, 0.57) \equiv (2, 4, 5, 1, 2) \quad \delta_2 = 0,$$

$$c_3 = (0.21, 0.47, 0.84, 0.41, 0.38) \equiv (1, 5, 4, 2, 3) \quad \delta_3 = 0.$$

To implement a BRKGA, we simply need to specify how solutions are encoded and decoded, including how their corresponding fitness values are computed. When the next population is complete, the vector of random keys are decoded and their corresponding fitness values are computed for all of the newly created random-key vectors. The population is partitioned into elite and non-elite individuals to start a new generation.

The offspring of a new population are used in the clustering process of CS. At each iteration, one individual s_k is grouped into the closest cluster j; i.e. the cluster that minimizes the Euclidean distance between s_k and the cluster center. The density δ_j is increased by one unit and the center c_j is updated with the new attributes of s_k (assimilation process). For example, if $s_k = (0.41, 0.92, 0.02, 0.27, 0.68) \equiv (3, 4, 1, 5, 2)$, the Euclidean distances to the centers (c_1, c_2, c_3) are respectively

$$\Delta_{s_k,c_1} = \sum_{i=1}^{5} \sqrt{(s_k(k_i) - c_1(k_i))^2} = 1.39,$$

$$\Delta_{s_k,c_2} = \sum_{i=1}^{5} \sqrt{(s_k(k_i) - c_2(k_i))^2} = 2.19,$$

$$\Delta_{s_k,c_3} = \sum_{i=1}^{5} \sqrt{(s_k(k_i) - c_3(k_i))^2} = 1.91.$$

Then, the cluster c_1 is the closest cluster for this individual s_k. The density δ_1 is increased by one unit.

The assimilation process uses the path-relinking method [29]. The procedure starts by computing the symmetric difference between the center c_j and the solution s_k, i.e., the set of moves needed to reach s_k from c_j. A path of solutions is generated, linking c_j and s_k. At each step, the procedure examines all moves from the current solution s and selects the one that results in the best-cost solution, applying the best move to solution s. The set of available moves is updated.

The procedure terminates when a percentage ρ of the solutions in the path have been analyzed. Thus, the center is not moved to a new location too far away from the current one. The new center c_j is the best solution along this path. In this paper, one move is to replace a single random key of c_j by a random key of s_k. An example of path-relinking moves between c_1 and s_k are

$$c_1 = (0.45, 0.62, 0.02, 0.87, 0.23)$$

$$(\mathbf{0.41}, -, 0.02, -, -)\ (-, \mathbf{0.92}, 0.02, -, -)\ \underline{(-, -, 0.02, \mathbf{0.27}, -)}\ (-, -, 0.02, -, \mathbf{0.68})$$

$$\underline{(\mathbf{0.41}, -, 0.02, 0.27, -)}\ (-, \mathbf{0.92}, 0.02, 0.27, -)\ (-, -, 0.02, 0.27, \mathbf{0.68})$$

$$(0.41, \mathbf{0.92}, 0.02, 0.27, -)\ \underline{(0.41, -, 0.02, 0.27, \mathbf{0.68})}$$

$$s_k = (0.41, 0.92, 0.02, 0.27, 0.68)$$

where solutions in the path are represented by the random-key replaced (in bold) and the random-keys that are already present in s_k. Random-keys that do not appear in the solution ($-$) are the same as in c_1. The underlined solution is the best solution at each iteration, and the new center c_1 is the best solution of the path analyzed.

After performing the path-relinking, we conduct an analysis of the density δ_j, verifying if this cluster can be considered promising. A cluster becomes promising when its density reaches the threshold λ ($\delta_j \geq \lambda$).

If the density δ_j reaches λ, local search heuristics are applied to the center c_j. In this paper, the Variable Neighborhood Descent (VND) [30] is implemented as local search component of CS, intensifying the search in the neighborhood of a promising cluster. The promising centers are decoded into a feasible solution of MTSP. For example, let us assume that the density δ_j of the center $c_j = (0.32, 0.78, 0.02, 0.41, 0.93)$ reaches the threshold λ. Thus, we apply the VND on the decoded solution $c_j \equiv (3, 1, 4, 2, 5)$.

Our VND utilizes four descent heuristics: Shift(1), Swap(1,1), Shift(2), and Swap(2,2) (see examples of these neighborhoods in Fig. 3):

(a) Shift(1) – $N^{(1)}$: A job k is transferred from its current position to position i;
(b) Swap(1,1) – $N^{(2)}$: Permutation between a job k and a job l;
(c) Shift(2) – $N^{(3)}$: Two adjacent jobs k and l are transferred from their current positions to positions i and $i+1$;
(d) Swap(2,2) – $N^{(4)}$: Permutation between two adjacent jobs k and l, and two other adjacent jobs k' and l'. Two different ways for exchanging jobs (k,l) and (k',l') are considered.

The descent heuristics are applied in the order $\{N^1, N^2, N^3, N^4\}$. Whenever a given heuristic fails to improve the incumbent solution, the VND chooses the next heuristic to continue the search. It returns to the first heuristic each time a better solution is found.

The solution spaces of the neighborhoods can be explored exhaustively, that is, all possible combinations are examined, and the best improving move is considered. When no improvement can be obtained, we stop.

The CS attempts to apply the VND only in promising regions. Thus, one strives to obtain the best possible solution within the neighborhood of a promising center.

At the end of VND, we adjust the cluster center to reflect the new order of the jobs. For example, if $c'_j = (3\ 5\ 4\ 2\ 1)$ in Fig. 3(b) is a local optimum obtained from c_j, we sort the keys of c_j to obtain $(0.02, 0.32, 0.41, 0.78, 0.93)$ and associate each element of c'_j with the corresponding key. That is, 3 is associated with 0.02, 5 is associated with 0.32, etc. Then, the encoding of the new center is $(0.93, 0.78, 0.02, 0.41, 0.32)$ by considering the jobs in the order 1, 2, 3, 4, 5.

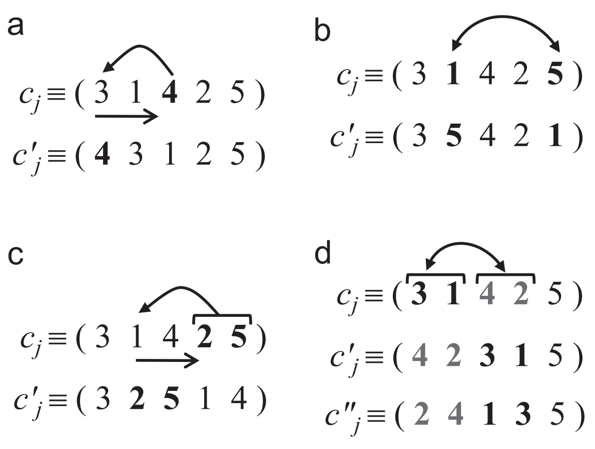
a
$c_j \equiv (3\ 1\ 4\ 2\ 5)$
$c'_j \equiv (4\ 3\ 1\ 2\ 5)$
b
$c_j \equiv (3\ 1\ 4\ 2\ 5)$
$c'_j \equiv (3\ 5\ 4\ 2\ 1)$
c
$c_j \equiv (3\ 1\ 4\ 2\ 5)$
$c'_j \equiv (3\ 2\ 5\ 1\ 4)$
d
$c_j \equiv (3\ 1\ 4\ 2\ 5)$
$c'_j \equiv (4\ 2\ 3\ 1\ 5)$
$c''_j \equiv (2\ 4\ 1\ 3\ 5)$

Fig. 3. Examples of the four neighborhood structures for MTSP.

This process of CS+BRKGA is applied repeatedly. The stopping criterion used in this paper is a fixed number of generations. There are other possible stopping criteria, including stopping after a fixed number of generations since the generation of the last solution improvement, after a time limit is reached, or after a solution at least as good as a given threshold is found.

4. Computational results

The CS and BRKGA were coded in C++ and the computational tests carried out on an Intel Core i7 3.4 GHz processor with 16GB of RAM. Nine problem sets are used in these tests: five sets introduced by Yanasse and Rodrigues [18] named A, B, C, D and E, and four sets introduced by Crama et al. [2], named C_1, C_2, C_3 and C_4.

Table 1 shows the values of the parameters of the five sets of Yanasse and Rodrigues [18] and the four sets of Crama et al. [2]. These instances as well as our best solutions can be found at http://www.sjc.unifesp.br/docente/chaves.

We chose a subset of instances to tune the parameters of the proposed method. We tune one parameter at a time. Its best value (in terms of quality of solutions and computational time) is determined empirically. Although setting these parameters is kind of an art form, our experience and suggestions from [21] have led us to the parameters values shown in Table 2. We can observe that the population size of CS+BRKGA was smaller and the number of mutants was larger than for the BRKGA. Though the CS+BRKGA used a small population, it was robust and enjoyed sufficient diversity.

Tables 3–8 present the results for the ILS [16], BRKGA, and CS+BRKGA. The entries in tables are the number of jobs (N), the number of tools (M), the machine capacity (C), the best solution ($S*$), the average solution (S) over 20 runs, the average running time to find the best solution ($T*$), and the average running time (T) in seconds. The values in boldface show the best objective function value for each instance.

Chaves et al. [16] propose a ILS based in [31]. It starts from an initial solution generated by a constructive heuristic based on a graph where each vertex corresponds to a tool and there is an arc $k = (i, j)$ between vertices i and j when the tools i and j are required for the execution of task k. The perturbation uses swap(1,1) moves to escape from local optima. The local search phase is also based on this move, with all solutions in the neighborhood being evaluated before the best one is returned. The stopping criterion of the ILS is a maximum number of iterations, defined as 3000 iterations.

The tables of results show that CS+BRKGA performed better than ILS [16]. For groups A, B (Tables 3 and 4), C and E (Tables 5 and 7), the CS+BRKGA found optimal solutions for all instances (proven by enumeration using the ILS upper bound

Table 1
Characteristics of MTSP instances.

Group	Number of jobs		Number of tools		Capacity		Number of instances
	Min	Max	Min	Max	Min	Max	
A	8	8	15	25	5	20	340
B	9	9	15	25	5	20	330
C	15	15	15	25	5	20	340
D	20	25	15	25	5	20	260
E	10	15	10	20	4	12	80
C_1	10	10	10	10	4	7	40
C_2	15	15	20	20	6	12	40
C_3	30	30	40	40	15	25	40
C_4	40	40	60	60	20	30	40

[19,16]). The ILS [16] does not find the optimal solution for two instances of C and one instance of E.

For group D (Table 6), the CS+BRKGA found optimal solutions for 189 of 260 instances (enumeration proved optimality of only those 189 instances) while ILS [16] did not find optimal solutions for 13 of these 189 instances. The ILS [16] found a better solution than CS+BRKGA for only one instance. Moreover, the CS+BRKGA obtained new upper bounds for three instances.

The instances of Crama et al. [2] were solved by enumeration [18,19], the ILS [16], the BRKGA, and CS+BRKGA. For each problem size (N, M, C), 10 instances were generated. Table 8 also shows the number of tool switches required by the best sequence found by the heuristics of Crama et al. [2]. The enumeration algorithm [18,19], limited to 3600 s of computational time, was able to get the optimal solution only for instances with 10 and 15 jobs. For these instances, the ILS and CS+BRKGA found all optimal solutions. Considering instances with 30 jobs, CS+BRKGA found better solutions than ILS in 32 of 40 instances and solutions with the same number of tool switches for 8 instances (5 in set C_4, that have larger capacities). For the instances with 40 jobs, CS+BRKGA found better solutions than ILS on all tested instances.

To show the advantage of the CS+BRKGA with respect to ILS when applied to MTSP, we calculated minimum (*Min*), maximum (*Max*), mean (*Mean*), median (*Median*), and standard deviation (*StdDev*) of the solution values obtained over 20 independent runs of these methods for each instance with 30 and 40 jobs. We also analyzed two other criteria for which lower values represent better results: the number of runs where the best known solutions (*BK*) was not obtained (*Nopt*) and the average relative distance value to best known solution for those runs that did not reach *BK* (*Avd*), expressed in percentages. Table 9 shows these results.

We observe that the CS+BRKGA found the best-known solutions for all instances. Moreover, the CS+BRKGA found the *BK* solution in 22% of tests (average $Nopt = 0.78$) and the deviation from the *BK* solution, when the *BK* solution was not found, was 1.79% on average (column *Avd*). The ILS found the *BK* solutions on a very small number of tests (only for sets $(30, 40, 20)$ and $(30, 40, 25)$) and its deviation from the *BK* solution was greater than 7%. The average improvement of the CS+BRKGA with respect to the ILS was 5% and 6% for instances with 30 and 40 jobs, respectively. The CS+BRKGA algorithm was robust, producing low percentage deviation from the best solution found (the average deviation found by CS+BRKGA was 0.8% for the instances of Yanasse and Rodrigues [18] and 1.5% for the instances of Crama et al. [2]).

Finally, we performed a statistical analysis to compare our CS+BRKGA and ILS proposed in [16] considering the 80 instances of 30 and 40 jobs with 20 independents runs for each one (two sets of 1600 solutions). First, we apply the Shapiro–Wilk normality test, which reject the null hypothesis that the data are normally distributed ($W = 0.685$, p-value $< 2.2e-16$).

We also apply a Wilcoxon signed-rank test (WSR) a non-parametric statistical hypothesis test [32]. This test is used to compare the two sets of solutions to investigate if a significant difference exists between the solutions of CS+BRKGA and ILS. The WSR indicated that the CS ranks were statistically less significant than the ILS ranks ($Z = 580$, $p < 2.2e-16$). This result suggests that the use of CS+BRKGA on this set of instances improves the results found in comparison with ILS.

The local search method (LS) is responsible, on average, for 60% of the computational time and the assimilation process (path-relinking method) is responsible for 30% of the computational time. Therefore, the total computational time of the ILS were better than the CS+BRKGA. However, the computational time of CS+BRKGA to find the best solution ($T*$) is, on average, less than for the ILS. The CS+BRKGA converges to best solutions in approximately 2% of the total time for smaller instances and 10% for instances with 30 and 40 jobs. Whereas the ILS converges within 7% and 47% of the total time, respectively. Generally the computational times of CS+BRKGA were competitive, finding very good solutions within few seconds for the instances up to 25 jobs and in a reasonable time for instances with 30 and 40 jobs.

For the 80 instances with 30 and 40 jobs, we made a boxplot analysis based on the computational time needed to find the best solution in each algorithm execution. In Fig. 4 we observe that CS+BRKGA had a better overall behavior, being more efficient with regard to computational time.

We can also observe that BRKGA without search intensification did not find good results for the tested instances, although the computational time is very low. This same fact was reported by Amorim [33], Roque et al. [34], who studied the application of BRKGA to solve the p-Median and the unit commitment problem,

Table 2
Values for the BRKGA and CS parameters.

Parameter	Meaning	BRKGA	CS+BRKGA
p	Number of individuals in population	2000	1000
Gen	Number of generations	100	100
p_e	Size of the elite set in population	0.20	0.20
p_m	Number of mutants to be introduced in population at each generation	0.15	0.20
ρ_e	Probability that an allele is inherited from the elite parent	0.70	0.70
$\mathcal{NC}$	Determines the number of clusters	–	20
ρ	Percentage of the analyzed path in the path-relinking	–	0.3
λ	Defines the maximum density for the local search	–	15

Table 3
MTSP: comparison of the results for instances of group A.

N	M	C	ILS [16]				BRKGA				CS+BRKGA			
			$S*$	S	$T*$	T	$S*$	S	$T*$	T	$S*$	S	$T*$	T
8	15	5	**12.00**	12.00	0.00	0.98	**12.00**	12.00	0.00	1.03	**12.00**	12.00	0.00	2.39
8	15	10	**6.83**	6.83	0.00	1.03	**6.83**	6.83	0.00	0.98	**6.83**	6.83	0.00	2.65
8	20	5	**16.80**	16.80	0.00	1.30	**16.80**	16.80	0.00	1.32	**16.80**	16.80	0.01	2.97
8	20	10	**13.07**	13.07	0.00	1.49	**13.07**	13.07	0.00	1.36	**13.07**	13.07	0.00	3.54
8	20	15	**7.08**	7.08	0.00	1.34	**7.08**	7.08	0.00	1.25	**7.08**	7.08	0.00	3.57
8	25	5	**20.10**	20.10	0.00	1.60	**20.10**	20.10	0.00	1.67	**20.10**	20.10	0.00	4.54
8	25	10	**18.20**	18.20	0.00	1.70	**18.20**	18.20	0.01	1.73	**18.20**	18.20	0.01	4.37
8	25	15	**12.95**	12.95	0.00	1.62	**12.95**	12.95	0.01	1.64	**12.95**	12.95	0.01	4.61
8	25	20	**6.61**	6.61	0.00	1.47	**6.61**	6.61	0.01	1.48	**6.61**	6.61	0.01	4.72
Average			12.63	12.63	0.00	1.39	12.63	12.63	0.00	1.38	12.63	12.63	0.00	3.71

Table 4
MTSP: comparison of the results for instances of group B.

N	M	C	ILS [16]				BRKGA				CS+BRKGA			
			S*	S	T*	T	S*	S	T*	T	S*	S	T*	T
9	15	5	**12.20**	12.20	0.00	1.39	**12.20**	12.20	0.00	1.10	**12.20**	12.20	0.01	2.72
9	15	10	**7.37**	7.37	0.00	1.27	**7.37**	7.38	0.01	1.07	**7.37**	7.37	0.01	3.15
9	20	5	**17.40**	17.40	0.01	1.82	**17.40**	17.40	0.03	1.38	**17.40**	17.40	0.02	3.13
9	20	10	**14.17**	14.17	0.02	1.81	**14.17**	14.19	0.01	1.43	**14.17**	14.17	0.02	3.92
9	20	15	**7.60**	7.60	0.01	1.58	**7.60**	7.62	0.01	1.32	**7.60**	7.60	0.01	3.99
9	25	5	**20.40**	20.40	0.01	2.22	**20.40**	20.40	0.01	1.73	**20.40**	20.40	0.01	4.08
9	25	10	**18.77**	18.77	0.02	2.34	**18.77**	18.80	0.05	1.81	**18.77**	18.77	0.03	5.08
9	25	15	**14.74**	14.74	0.02	2.23	**14.74**	14.77	0.04	1.77	**14.74**	14.75	0.03	5.10
9	25	20	**7.19**	7.19	0.01	2.30	**7.19**	7.20	0.01	1.62	**7.19**	7.19	0.01	5.26
Average			13.31	13.31	0.01	1.88	13.31	13.33	0.02	1.47	13.31	13.32	0.02	4.05

Table 5
MTSP: comparison of the results for instances of group C.

N	M	C	ILS [16]				BRKGA				CS+BRKGA			
			S*	S	T*	T	S*	S	T*	T	S*	S	T*	T
15	15	5	**16.60**	16.64	0.69	5.90	**16.60**	17.25	0.23	1.53	**16.60**	16.69	0.21	5.31
15	15	10	**9.80**	9.86	0.41	6.15	9.87	10.12	0.07	1.53	**9.80**	9.88	0.09	7.10
15	20	5	**20.60**	20.74	0.95	9.60	**20.60**	21.25	0.24	2.02	**20.60**	20.77	0.51	7.26
15	20	10	18.40	18.58	0.72	9.11	18.43	19.00	0.16	2.07	**18.33**	18.52	0.32	8.93
15	20	15	**10.52**	10.58	0.63	7.07	10.57	10.90	0.09	1.92	**10.52**	10.65	0.21	9.61
15	25	5	**27.50**	27.67	0.80	9.99	27.70	28.24	0.33	2.51	**27.50**	27.70	0.38	9.30
15	25	10	**25.07**	25.21	1.30	11.49	25.17	25.81	0.21	2.65	**25.07**	25.30	0.41	13.52
15	25	15	**19.07**	19.19	1.10	10.74	19.17	19.70	0.17	2.50	**19.07**	19.27	0.36	13.63
15	25	20	**9.66**	9.76	0.52	8.59	9.70	10.02	0.10	2.31	**9.66**	9.79	0.27	13.82
Average			17.47	17.58	0.79	8.74	17.53	18.03	0.18	2.12	17.53	17.62	0.31	9.83

Table 6
MTSP: comparison of the results for instances of group D.

N	M	C	ILS [16]				BRKGA				CS+BRKGA			
			S*	S	T*	T	S*	S	T*	T	S*	S	T*	T
20	15	5	21.20	21.72	3.55	15.00	21.40	22.45	0.47	3.38	**21.10**	21.58	0.80	10.78
20	15	10	**8.20**	8.41	0.97	12.62	8.40	8.78	0.26	3.46	**8.20**	8.44	0.41	12.34
20	20	5	24.40	24.86	6.17	23.21	24.80	25.93	0.64	5.49	**24.30**	24.93	1.24	14.84
20	20	10	**10.60**	10.66	0.96	21.99	**10.60**	10.76	0.16	5.78	**10.60**	10.76	0.38	16.08
20	20	15	**6.67**	6.72	0.61	17.90	**6.67**	6.85	0.12	5.09	**6.67**	6.79	0.33	24.66
20	25	5	30.40	30.87	4.41	23.74	30.40	31.68	0.76	5.89	**30.10**	30.74	1.40	19.16
20	25	10	**15.40**	15.48	1.02	22.30	**15.40**	15.56	0.26	6.11	**15.40**	15.47	0.80	21.49
20	25	15	21.43	21.79	3.58	22.65	21.55	22.45	0.47	5.92	**21.25**	21.75	0.93	28.11
20	25	20	6.18	6.24	0.58	18.34	6.18	6.35	0.18	5.21	**6.15**	6.28	0.52	35.53
25	15	10	**5.90**	5.96	0.93	21.87	6.00	6.12	0.11	4.18	**5.90**	6.00	1.02	21.14
25	20	10	**11.60**	11.93	3.55	31.27	11.90	12.40	0.38	5.45	**11.60**	12.05	1.94	27.48
25	20	15	**7.60**	7.73	2.07	27.81	7.70	8.02	0.22	5.07	**7.60**	7.82	1.57	25.66
25	25	10	**16.60**	16.86	6.51	41.19	16.70	17.45	0.51	6.63	**16.60**	17.06	2.66	36.88
25	25	15	**10.00**	10.00	0.00	38.92	**10.00**	10.00	0.04	6.61	**10.00**	10.00	0.01	54.70
25	25	20	**5.50**	5.53	0.91	35.91	5.53	5.65	0.13	6.12	**5.50**	5.59	1.14	66.10
Average			13.44	13.65	2.39	24.98	13.55	14.03	0.31	5.36	13.40	13.68	1.01	27.66

respectively. The authors obtained good results when a local search heuristic was added to the decoder of the BRKGA. Therefore, the proposed CS introduces intelligence and priority to the choice of solutions to apply, generally costly, local searches, instead of applying random or elitist choices.

We perform computational tests by applying the VND to all offspring generated by the BRKGA. However, this approach did not result in better solution quality than those obtained with CS+BRKGA. Furthermore, there was a significant increase of computational time. In some cases this increase of computational time, sometimes up to 30%. Chaves [22] has shown that the combination of metaheuristic, path-relinking and local search in CS is usually better than the combination of just two of these techniques.

Fig. 5 illustrates run-time distributions, or time-to-target (TTT) plot [35], for MTSP instances. The experiment consists in running the ILS and CS+BRKGA 100 times on the instances of type (40, 60, 20). Each run is independent of the other and stops when a

Table 7
MTSP: comparison of the results for instances of group E.

N	*M*	*C*	ILS [16]				BRKGA				CS+BRKGA			
			*S**	*S*	*T**	*T*	*S**	*S*	*T**	*T*	*S**	*S*	*T**	*T*
10	10	4	**9.50**	9.50	0.02	1.17	**9.50**	9.53	0.03	0.76	**9.50**	9.50	0.02	1.55
10	10	5	**6.20**	6.20	0.00	1.12	**6.20**	6.20	0.02	0.82	**6.20**	6.21	0.01	1.96
10	10	6	**4.30**	4.30	0.00	1.09	**4.30**	4.30	0.00	0.89	**4.30**	4.30	0.00	2.88
10	10	7	**3.00**	3.00	0.00	1.07	**3.00**	3.00	0.00	0.92	**3.00**	3.00	0.00	3.45
15	20	6	**21.40**	21.54	1.07	8.84	**21.40**	22.11	0.17	2.01	**21.40**	21.71	0.31	7.09
15	20	8	14.30	14.35	0.62	7.88	14.30	14.53	0.06	2.06	**14.20**	14.33	0.20	7.68
15	20	10	**10.30**	10.30	0.01	7.58	**10.30**	10.34	0.04	2.25	**10.30**	10.34	0.11	12.71
15	20	12	**8.20**	8.20	0.00	7.36	**8.20**	8.20	0.01	2.26	**8.20**	8.20	0.00	14.97
Average			9.65	9.67	0.21	4.51	9.65	9.78	0.04	1.50	9.64	9.70	0.08	6.54

Table 8
MTSP: comparison of the results for instances of Crama et al. [2].

N	*M*	*C*	Best(I) [2]	Enumerative [19]		ILS [16]				BRKGA				CS+BRKGA			
			*S**	*S**	*T*	*S**	*S*	*T**	*T*	*S**	*S*	*T**	*T*	*S**	*S*	*T**	*T*
10	10	4	13.20	**9.10**	0.01	**9.10**	9.10	0.01	0.48	**9.10**	9.21	0.01	0.18	**9.10**	9.11	0.01	1.57
10	10	5	11.20	**6.20**	0.01	**6.20**	6.20	0.00	0.43	**6.20**	6.20	0.00	0.19	**6.20**	6.20	0.00	2.05
10	10	6	10.30	**4.30**	0.01	**4.30**	4.30	0.00	0.46	**4.30**	4.30	0.00	0.20	**4.30**	4.30	0.00	2.64
10	10	7	10.10	**3.10**	0.00	**3.10**	3.10	0.00	0.39	**3.10**	3.10	0.00	0.22	**3.10**	3.10	0.00	3.41
15	20	6	26.50	**20.60**	1.94	**20.60**	20.90	0.41	3.62	20.80	21.66	0.08	0.56	**20.60**	20.87	0.32	8.18
15	20	8	21.60	**13.70**	3.04	**13.70**	13.70	0.11	2.86	**13.70**	14.12	0.05	0.58	**13.70**	13.72	0.13	8.88
15	20	10	20.00	**10.10**	60.79	**10.10**	10.10	0.03	2.80	**10.10**	10.23	0.02	0.61	**10.10**	10.10	0.09	11.20
15	20	12	19.60	**7.60**	95.41	**7.60**	7.60	0.00	2.68	**7.60**	7.60	0.00	0.63	**7.60**	7.60	0.01	18.05
30	40	15	113.60	96.10	3600.00	94.00	96.73	31.55	71.77	96.80	101.32	1.61	2.80	**91.80**	93.03	14.99	140.05
30	40	17	95.90	76.80	3600.00	74.00	76.38	29.32	65.11	76.60	81.00	1.28	2.56	**71.70**	72.98	15.66	127.09
30	40	20	76.80	56.90	3600.00	52.20	54.20	26.48	62.41	55.80	58.67	0.93	2.40	**50.70**	51.85	11.07	121.43
30	40	25	56.80	35.40	3600.01	28.80	30.18	17.23	50.80	31.00	33.42	0.65	2.04	**28.10**	28.97	10.94	104.02
40	60	20	211.60	192.60	3600.00	188.40	191.93	146.46	298.26	192.50	199.70	5.03	6.51	**179.80**	182.25	63.30	599.34
40	60	22	189.70	167.10	3600.01	161.00	164.34	145.01	276.31	164.50	171.52	4.55	6.31	**153.30**	155.28	60.39	557.97
40	60	25	160.50	137.70	3600.01	128.70	132.26	135.49	262.15	131.80	139.27	3.99	5.80	**122.50**	124.35	57.68	533.52
40	60	30	127.40	102.40	3600.01	90.70	93.07	114.41	233.97	93.90	99.58	2.96	5.17	**84.50**	86.94	50.54	473.56
Average			72.80	58.73	1810.08	55.78	57.13	40.41	83.41	57.36	60.06	1.32	2.30	53.57	54.42	17.82	169.56

Table 9
MTSP: comparison between ILS and CS+BRKGA using seven criteria.

N	*M*	*C*	*BK*	CS+BRKGA							ILS [16]						
				Min	Max	Mean	Median	StdDev	*Nopt* (%)	*Avd* (%)	Min	Max	Mean	Median	StdDev	*Nopt* (%)	*Avd* (%)
30	40	15	91.80	91.80	94.40	93.03	93	0.73	0.80	1.38	94.00	99.00	96.73	97	1.33	1.00	5.41
30	40	17	71.70	71.70	74.30	72.88	73	0.81	0.78	1.64	74.00	78.50	76.39	76	1.27	1.00	6.54
30	40	20	50.70	50.70	53.00	51.79	51	0.69	0.76	2.14	52.20	56.10	54.20	54	1.08	0.99	6.89
30	40	25	28.10	28.10	29.80	28.94	29	0.49	0.66	2.92	28.80	31.50	30.18	30	0.74	0.92	7.24
40	60	20	179.80	179.80	184.50	182.01	184	1.31	0.86	1.24	188.40	195.10	191.93	195	1.79	1.00	6.76
40	60	22	153.30	153.30	157.00	155.16	157	1.13	0.84	1.22	161.00	167.20	164.34	167	1.68	1.00	7.21
40	60	25	122.50	122.50	126.10	124.20	126	1.10	0.80	1.38	128.70	134.90	132.26	134	1.55	1.00	7.96
40	60	30	84.50	84.50	88.60	86.55	87	1.31	0.78	2.42	90.70	95.00	93.07	94	1.22	1.00	10.14
Average				**97.80**	**100.96**	**99.32**	**99.94**	**0.95**	**0.78**	**1.79**	102.23	107.16	104.88	105.75	1.33	0.99	7.27

solution with a cost which is at least as good as a given target value is found. In these experiments, we observe an integer value at most 5% greater than the best-known solution.

In these experiments, we want an integer value that the relative position of the curves implies that, given any fixed amount of running time, CS+BRKGA has a higher probability than does ILS of finding a solution whose objective function value is at least as good as the target objective function value. For example, in Fig. 4 the probability of the CS+BRKGA to find a solution at least as good as the target value in at most five seconds is about 80%, in at most 10 s is about 95%, and in at most 20 s is about 99%. The probability of the ILS to find a solution at least as good as the target value in at most 20 s is only 20%. For a probability of 90% over 150 s are required. Other tested instances also had this same behavior.

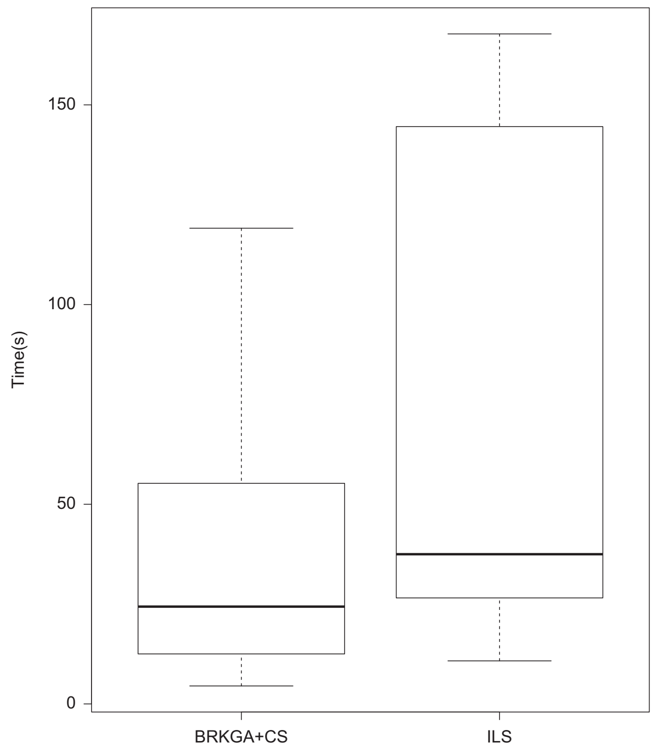

Fig. 4. Time to find the best solution with CS+BRKGA and ILS (time in seconds).

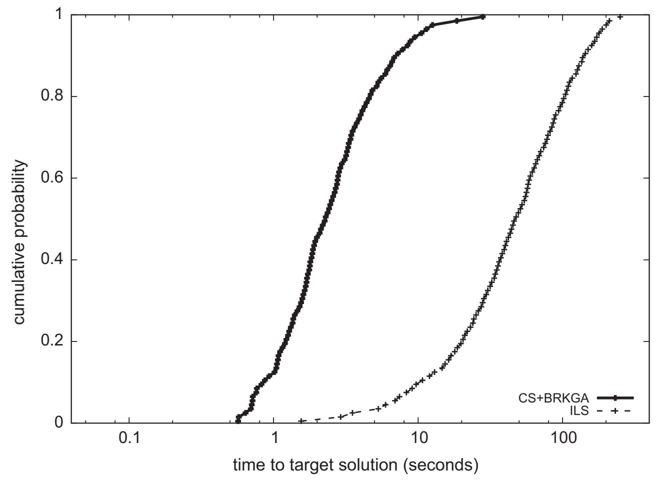

Fig. 5. Time to target distributions of CS+BRKGA and ILS for MTSP instances of type (40, 60, 20) (time in seconds).

5. Conclusions

This paper presents a new method based on the Biased Random Key Genetic Algorithms (BRKGA) and the Clustering Search (CS) to solve the Minimization of Tool Switches Problem (MTSP). The BRKGA and CS have been applied with success in many combinatorial optimization problems [21,16]. However, this work is the first approach where both methods are used to solve an optimization problem.

The idea of this paper was to simplify the clustering process of the CS based on the concept of random keys. With this, users have available an application in which one needs to only implement the decoder and local search heuristics.

Furthermore, this hybrid method detects the promising regions in the search space and applies local search only in these regions. The detection of promising areas becomes an interesting option and prevents the indiscriminate use of local search heuristics.

This paper reports results found by BRKGA and CS on over 1510 instances for the MTSP. The CS+BRKGA found optimal solutions for 1360 instances (proven by [19,16]) and the best known solutions for the others. The results show that the CS+BRKGA is competitive for solving the MTSP. New different instances in size, difficulties and structure may be randomly generated to evaluate the performance of the CS+BRKGA.

For further work, it is intended to develop an API for the CS+BRKGA (coded in C++ and based on [28]). Then, it should be easy to apply this method to other combinatorial optimization problems. We also want to explore an automated algorithm for parameters tuning, known as iterated F-race [36]. Other aspects of the CS may be analyzed by parallelizing its various algorithmic components and by applying it for multiobjective optimization problem.

Acknowledgments

This work was supported by the FAPESP under Grant 2012/17523-3; and CNPq under Grants 482170/2013-1, 304979/2012-0, 476862/2012-4, 300692-2009-9 and 300692/2009-9. The research of Mauricio G.C. Resende was done while he was employed at AT&T Labs Research.

Appendix A. KTNS pseudo-code

Fig. A1 presents the pseudo-code of the Keep Tool Needed Soonest (KTNS) policy, proposed by Tang and Denardo [1], to minimize the total number of tool switches for a fixed job sequence.

Where

```
Step 1: Set $J_i = 1$ for C values of $i$ having minimal values of $L(i, 0)$.
        Break the ties arbitrarily. Set $J_i = 0$ for the remaining $M - C$
        values of $i$. Set $n = 1$.
Step 2: Set $W_n = J$. Stop if $n = N$.
Step 3: If each $i$ having $L(i, n) = n$ also has $J_i = 1$, set $n = n + 1$ and
        go to Step 2.
Step 4: Pick $i$ having $L(i, n) = n$ and $J_i = 0$. Set $J_i = 1$.
Step 5: Set $J_k = 0$ for a $k$ that maximizes $L(p, n)$ over $\{p: J_p = 1\}$. Go
        to Step 3.
```

Fig. A1. KTNS pseudo-code [1].

- J is the vector whose ith entry J_i is equal to 1 if tool i is on the machine at a given instant n, and 0 otherwise;
- $L(i, n)$ is the first instant at or after instant n at which tool i is needed.

References

[1] Tang CS, Denardo EV. Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. Oper Res 1988;36(5):767–77.

[2] Crama Y, Kolen AW, Oerlemans A, Spieksma FC. Minimizing the number of tool switches on a flexible machine. Int J Flex Manuf Syst 1994;6(1):33–54.

[3] Hertz A, Laporte G, Mittaz M, Stecke KE. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE Trans 1998;30(8):689–94.
[4] Matzliach B, Tzur M. The online tool switching problem with non-uniform tool size. Int J Prod Res 1998;36:3407–20.
[5] Shirazi R, Frizelle GDM. Minimizing the number of tool switches on a flexible machine: an empirical study. Int J Prod Res 2001;39:3547–60.
[6] Fathi Y, Barnette KW. Heuristic procedures for the parallel machine problem with tool switches. Int J Prod Res 2002;40(1):151–64.
[7] Song CY, Hwang H. Optimal tooling policy for a tool switching problem of a flexible machine with automatic tool transporter. Int J Prod Res 2002;40:873–83.
[8] Ghrayeb OA, Phojanamongkolkij N, Finch PR. A mathematical model and heuristic procedure to schedule printed circuit packs on sequencers. Int J Prod Res 2003;41(16):3849–60.
[9] Al-Fawzan M, Al-Sultan K. A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. Comput Ind Eng 2003;44(1):35–47.
[10] Senne ELF, Yanasse HH. Beam search algorithms for minimizing tool switches on a flexible manufacturing system. In: Proceedings of the 11th WSEAS international conference on mathematical and computational methods in science and engineering, MACMESE'09. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS); 2009. p. 68–72.
[11] Konak A, Kulturel-Konak S. An ant colony optimization approach to the minimum tool switching instant problem in flexible manufacturing system. In: 2007 IEEE symposium on computational intelligence in scheduling (CISched 2007). Honolulu, Hawaii, USA; 2–4 April 2007. p. 43–8.
[12] Konak A, Kulturel-Konak S, Azizoglu M. Minimizing the number of tool switching instants in Flexible Manufacturing Systems. Int J Prod Econ 2008;116(2):298–307.
[13] Amaya J, Cotta C, Fernández A. A memetic algorithm for the tool switching problem. In: Blesa M, Blum C, Cotta C, Fernandez A, Gallardo J, Roli A, et al., editors. Hybrid metaheuristics, Lecture notes in computer science, vol. 5296. Malaga, Spain: Springer Berlin Heidelberg; 2008. p. 190–202.
[14] Amaya J, Cotta C, Leiva A. Hybrid cooperation models for the tool switching problem. In: González J, Pelta D, Cruz C, Terrazas G, Krasnogor N, editors. Nature inspired cooperative strategies for optimization (NICSO 2010), Studies in computational intelligence, vol. 284. Malaga, Spain: Springer Berlin Heidelberg; 2010. p. 39–52.
[15] Amaya JE, Cotta C, Fernandez-Leiva AJ. Solving the tool switching problem with memetic algorithms. Artif Intell Eng Des Anal Manuf 2012;26:221–35.
[16] Chaves AA, Senne ELF, Yanasse HH. Uma nova heurística para o problema de minimização de trocas de ferramentas. Gest Prod 2012;19:17–30.
[17] Laporte G, Salazar-González JJ, Semet F. Exact algorithms for the job sequencing and tool switching problem. IIE Trans 2004;36(1):37–45.
[18] Yanasse HH, Rodrigues RCM. A partial ordering enumeration scheme for solving the minimization of tool switches problem. In: Proceedings of INFORMS annual meeting Seattle 2007. Seattle, Washington; 2007. p. 299 (Book of Abstracts).
[19] Yanasse HH, Rodrigues RCM, Senne ELF. Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. Gest Prod 2009;16(3):370–81.
[20] Oliveira ACM, Chaves AA, Lorena LAN. Clustering search. Pesqui Oper 2013;33:105–21.
[21] Gonçalves J, Resende M. Biased random-key genetic algorithms for combinatorial optimization. J Heuristics 2011;17:487–525.
[22] Chaves AA. A hybrid metaheuristic with clustering search applied to combinatorial optmization problems [thesis], Applied Computing; 2009.
[23] Glover F, Martí R. Fundamentals of scatter search and path relinking. Control Cybern 2000;39:653–84.
[24] Gonçalves JF, Resende MG. A biased random key genetic algorithm for 2D and 3D bin packing problems. Int J Prod Econ 2013;145(2):500–10.
[25] Bean JC. Genetic algorithms and random keys for sequencing and optimization. ORSA J Comput 1994;6(2):154–60.
[26] Spears WM, Jong KAD. On the virtues of parameterized uniform crossover. In: Proceedings of the fourth international conference on genetic algorithms; 1991. p. 230–6.
[27] Buriol LS, Hirsch MJ, Pardalos PM, Querido T, Resende MG, Ritt M. A biased random-key genetic algorithm for road congestion minimization. Optim Lett 2010;4(4):619–33.
[28] Toso R, Resende M. A C++application programming interface for biased random-key genetic algorithms. Optim Methods Softw 2015;30(1):81–93.
[29] Glover F. Tabu search and adaptive memory programing? Advances, applications and challenges. In: Interfaces in computer science and operations research. Kluwer; 1996. p. 1–75.
[30] Hansen P, Mladenovic N. A tutorial on variable neighborhood search. Technical Report, Les Cahiers du GERAD, HEC Montreal and GERAD; 2003.
[31] Lourenco H, Martin O, Stützle T. Iterated local search. In: Glover F, Kochenberger GA, editors. Handbook of metaheuristics, International series in operations research & management science, vol. 57. New York: Springer; 2003. p. 320–53.
[32] Rey D, Neuhauser M. Wilcoxon-signed-rank test. In: Lovric M, editor. International encyclopedia of statistical science. Springer Berlin Heidelberg; 2014. p. 1658–9.
[33] Amorim FMS. Metaheurísticas aplicadas ao problema das p-medianas [Master's thesis], Centro Federal de Educaç ao Tecnológica de Minas Gerais, Brazil, Belo Horizonte; 2011.
[34] Roque L, Fontes D, Fontes F. A hybrid biased random key genetic algorithm approach for the unit commitment problem. J Comb Optim 2014:1–27.
[35] Aiex R, Resende M, Ribeiro C. TTT plots: a perl program to create time-to-target plots. Optim Lett 2007;1(4):355–66.
[36] López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium; 2011.