algorithms

MDPI

Article

Metaheuristics for the Minimum Time Cut Path Problem with Different Cutting and Sliding Speeds

Bonfim Amaro Junior [1], Marcio Costa Santos [1], Guilherme Nepomuceno de Carvalho [1], Luiz Jonatã Pires de Araújo [2] and Placido Rogerio Pinheiro [3,*]

[1] Núcleo de Estudo em Machine Learning e Otimização (NEMO), Federal University of Ceará, Russas 62900-000, Brazil; bonfimamaro@ufc.br (B.A.J.); marciocs@ufc.br (M.C.S.); guilhermenepomuceno46@gmail.com (G.N.d.C.)
[2] Machine Learning & Knowledge Representation (MlKr) Lab, Innopolis University, 420500 Innopolis, Russia; l.araujo@innopolis.university
[3] Graduate Program in Applied Informatics, University of Fortaleza (UNIFOR), Fortaleza 60811-905, Brazil
* Correspondence: placido@unifor.br

Abstract: The problem of efficiently cutting smaller two-dimensional pieces from a larger surface is recurrent in several manufacturing settings. This problem belongs to the domain of cutting and packing (C&P) problems. This study approached a category of C&P problems called the minimum time cut path (MTCP) problem, which aims to identify a sequence of cutting and sliding movements for the head device to minimize manufacturing time. Both cutting and slide speeds (just moving the head) vary according to equipment, despite their relevance in real-world scenarios. This study applied the MTCP problem on the practical scope and presents two metaheuristics for tackling more significant instances that resemble real-world requirements. The experiments presented in this study utilized parameter values from typical laser cutting machines to assess the feasibility of the proposed methods compared to existing commercial software. The results show that metaheuristic-based solutions are competitive when addressing practical problems, achieving increased performance regarding the processing time for 94% of the instances.

Keywords: cut & packing problems; evolutionary computation; cut determination problem

Citation: Amaro Junior, B.; Santos, M.C.; de Carvalho, G.N.; de Araújo, L.J.P.; Pinheiro, P.R. Metaheuristics for the Minimum Time Cut Path Problem with Different Cutting and Sliding Speeds. *Algorithms* **2021**, *14*, 305. https://doi.org/10.3390/a14110305

Academic Editors: Hsiang-Ling Chen, Yun-Chia Liang, Mehmet Fatih Tasgetiren and Quan-Ke Pan

Received: 8 October 2021
Accepted: 20 October 2021
Published: 23 October 2021

1. Introduction

Cutting and packing (C&P) are optimization problems that concern the efficient arrangement of items within a larger space with the same dimensionality. This type of problem appears in several real-world industrial settings, including manufacturing, logistics, and 3D printing [1–3]. Therefore, optimization methods for C&P can represent a valuable commercial advantage [4]. This study focused on the minimum time cut path (MTCP) problem, which aims to identify a sequence of instructions (cutting or moving the head device) that minimizes the total cutting time for a packing arrangement or layout.

Prior to the solution of MTCP, a packing procedure is executed to achieve maximal surface area utilization [5,6]. In other words, all the items must be arranged within a surface of fixed width and minimal length, as illustrated in Figure 1. According to Araújo et al.'s taxonomy [4], this is a $2|Si|Oo$ problem, that is, two-dimensional, with a single input minimization and an open-dimensional volume. Wäscher et al. used the term strip packing problem to refer to such [7]. In addition, the literature contains several constraints that resemble more realistic settings, including the requirement for guillotined cuts [8] or irregular objects [9].

MTCP problems have a primary objective: the minimal time required for cutting the pieces represented by an input layout [5,6]. This data concerns the packing of all the items within a surface that represent the raw material, as illustrated in Figure 1. The position of pieces in the layout stage can be performed by computer-aided design (CAD)

systems [10,11] or generated by algorithmic methods [8,12–14]. Wäscher et al. used the term strip packing problem to refer to them [7]. In addition, the literature contains several constraints that resemble more realistic settings, including the requirement for guillotined cuts [8] or irregular objects [9].

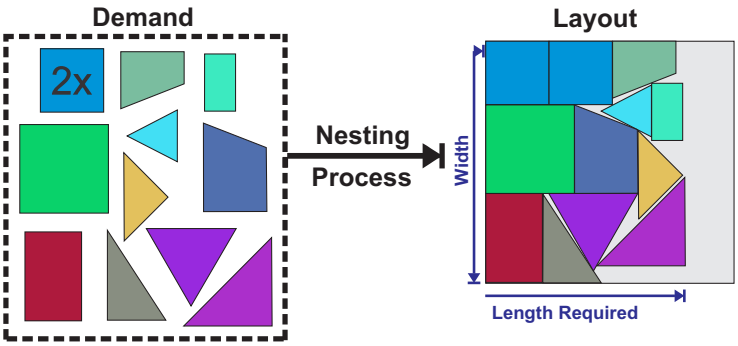

Figure 1. Example of a two-dimensional C&P problem with irregular items and a single open-dimensional surface of fixed width and variable length. This is a 2|*Si*|*Oo* problem according to Araújo et al.'s taxonomy [4].

The central aspect of MTCP, proposed in this study, is the minimal cutting time, which depends on two parameters of manufacturing laser cut machines: the cutting and the moving (or sliding) speed of the head device. The first parameter depends on the machine hardware, the shapes of pieces in the input layout, and the surface material. The second parameter refers to the speed to change the position of the cutting head without cutting. Thus, we tackled a generalization type of the cut determination problem (CPD) [15] adopting the above-mentioned restrictions differently from the studies developed by Lee et al. [16] and Derwil et al. [17], respectively.

From an optimization perspective, a solution obtained to an MTCP problem is a path through the layout area considering the equipment cutting and sliding moves and the resultant manufacturing cut time. This study tackled the minimum time cut path (MTCP) problem with different cutting and sliding speeds. For simplicity of notation, the remainder of this article will refer to this particular problem simply as MTCP.

This study presented two evolutionary metaheuristic-based approaches to tackle larger instances that resemble real-world scenarios. The metaheuristics used in the conducted experiments are a standard genetic algorithm (GA) and a biased random-key genetic algorithm (BRKGA). The experiments used an extensive set of instances from the literature and allowed one to gain valuable insights into the algorithms and their performance compared to specialized commercial software.

The remainder of this article is organized as follows. Section 2 presents the most common algorithmic approaches to MTCP problems and shows the definition and terminology for MTCP. Next, Section 3 introduces two evolutionary algorithms for the MTCP problem. Section 4 presents the results of the computational experiments using an extensive set of instances. Section 5 concludes by discussing the performance of the used algorithms and future works to tackle MTCP.

2. Literature Review

As mentioned previously, MTCP problems consider the minimization of the total cutting time to extract the pieces from the input layout, and it is often called cutting path determination (CPD) [18]. CPD aims to determine the sequence of moves for the cutting head necessary to separate all the smaller pieces from the surface (also referred to as the stage in an industrial setting). This section focuses on approaches for CPD problems. For comprehensive surveys on C&P problems, we refer to [13,19].

Hoeft and Palekar [20] categorized CPD problems according to the flexibility to choose an initial contour entry and whether a piece is only partially cut before the head

device moves to another object. The first category, according to [20], contains problems with *continuous cutting*, i.e., the cut is allowed to start at any point of the perimeter of pieces [21,22]. In such problems, the entry point should be the same for both entry and departure. The second category is called *endpoint cutting*, which contains problems in which the cut starts and ends at predefined vertices of the polygons [5,6]. The last category is *intermittent cutting*, in which there are no restrictions on the points that can be used for entry or exit of the cutting [23,24].

The literature contains several examples of algorithmic approaches for CPD. Dewil et al. [25] proposed grouping these methods according to the technique for traversing the vertices of the polygons. Three categories (here presented with examples of studies that employ such an approach) were identified by the authors: the touring polygons problem [26,27]), the traveling salesman problem (TSP) [28,29], the generalized TSP [30,31], and the TSP with neighborhoods [21]. Figure 2 presents a classification for CPD and the degree of generalization for this problem. The generalization level grows considering the flexibility of starting a path at any point of an item's layout. Besides, it is possible to cut only segments of a part, not necessarily starting and ending the cutting contour in the same item.

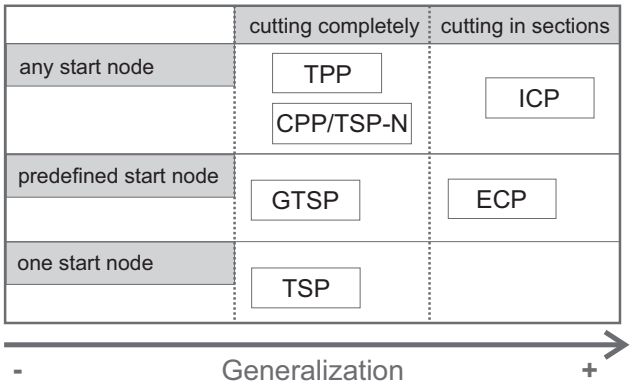

Figure 2. Classification for cutting path problems according to [18,20,25]. TPP: touring polygons problem; CCP: continuous cutting problem; TSP: traveling salesman problem; TSP-N: TSP with neighborhoods; ICP: intermittent cutting problem; GTSP: generalized TSP; ECP: endpoint cutting problem.

Laser cutting, as the designation suggests, applies a laser origin to cut the material. A benefit of laser cutting is that both the expanse of the cut and the heat-affected areas are tiny. Additionally, it is similar to both flame cutting and plasma cutting [32], and it is essential to highlight that the enclosed area detaches from the raw material after the cut of a complete piece contour. Depending on the supporting grid and air of the laser nozzle, it can shift its position, or if there is no supporting grid, it simply falls through. In both cases, it will be impracticable to continue cutting in this area.

The CPD problem aims to plan a path that minimizes the time required to cut all pieces regarding precedence constraints and is described in Dewil et al. [17].

Additional objectives include minimizing the cut across contours' path lengths and the effect of heat on the cutting path sequence [33]. A possible additional constraint is the requirement for a predefined cutting sequence for the items, which are to be cut without sliding movements [16]. Manber and Israni [6] tackled the problem of sequencing a torch (flame cutter machine) for cutting regular and irregular parts arranged on a surface. The main objective was to minimize the number of piercings, i.e., small holes made near each piece to improve the cutting process.

One of the approaches for CPD problems is the use of linear integer models to determine the sequence of moves that minimizes the overall time required to cut the demand of pieces [34].

Dewil et al. [17] extended [34] by assuming an additional set of constraints that resemble real-world requirements, for example, including inner–outer contours relations,

resulting from holes in parts, parts allocated in holes, or elements nested in enclosed waste areas. The consideration of the inner–outer contour means that an inner contour needs to be completely cut before the outer shape is cut. In summary, all pieces of an inner contour necessitate to be cut before the end element of its outer contour is cut. Additionally, it is possible to suggest a set of constraints about basic cuts. In some layouts, each typical cut is enveloped by a contour formed of both its two contours. Hence, no typical cut is permitted to connect both of its contours.

Another kit of priority constraints appears from the evidence that when one cut the contour of two contours in common cut with one another, the separated contour can slide, making the rest of the cut process unfeasible. To correctly cut the items of the residual contour, the laser has to move into the cut kerf. It is forbidden if a high part quality is required, and a pre-cut should have been placed earlier. When cutting a part, a tiny pre-cut can be made in a nearby element if the laser head has to begin cutting from this place later on. Several non-trivial practical extensions (additional practicalities) like collisions, bridges, and thermal effects also are present in Dewil et al. [25].

A similar approach consists of reducing CPD to graph-based problems such as the capacitated node routing problem (NRP), also known as the vehicle routing or dispatch problem [35], and then optimized through mathematical models [15,36]. These techniques address CPD by utilizing a mathematical formulation based on the NRP problem and a derived model for the traveling salesman problem (TSP). This approach has been demonstrated to be suitable for achieving optimal solutions, for instance, containing approximately 2000 edges in a reasonable time. The formulation in [15] achieved optimal results for larger instances with up to 712 edges and a maximum of 560 nodes.

It is noteworthy that the studies that use mathematical models have been impractical for more realistic instances with tens of thousands of edges and nodes. A strategy to mitigate possible limitations is using heuristics and metaheuristics for solving graph-based problems, which are equivalent to the original CPD problem. For example, Moreira et al. [5] employed this approach, also considering that the surface is at an elevation (height) and that items fall as they are cut. Despite the wide variety of CPD problems in the literature, problems with different cutting and sliding speeds have not been formally described to the best of the authors' knowledge.

3. Evolutionary Metaheuristics for MTCP

This section introduces the steps for building two evolutionary-based approaches for tackling MTCP, considering different moving and cutting speeds: a genetic algorithm (GA) and a biased random-key genetic algorithm (BRKGA).

3.1. A GA-Based Approach

As mentioned previously, the MTCP with different moving and cutting speeds can be seen as a generalization of the CPD problem [5,15]. The input data for the MTCP problem is a packing layout, i.e., a set of non-overlapping polygons, which are defined as a set of two-dimensional points and edges, as illustrated in Figure 3. It also includes, in our approach, the moving (μ) and cutting speeds (π), which are numerical parameters that vary according to the machinery and the raw material. Let $d(A, B)$ be the distance between the points A and B using a metric that respects the triangle inequality. This study adopted the Chebyshev metric (see Figure 4) since it abstracts aspects that are ignored: deceleration, acceleration, and effects from the cutting process, such as surface bending. The solution for the MTCP problem is a sequence of actions (being either moving or cutting) for the cutting head device.

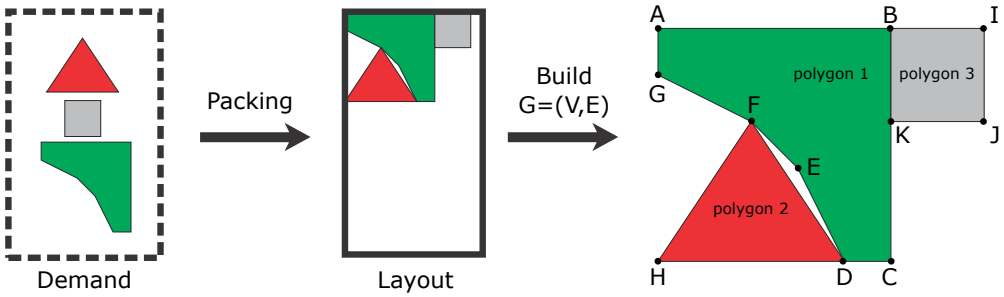

Figure 3. Example of the layout conversion to a set of points in the plane.

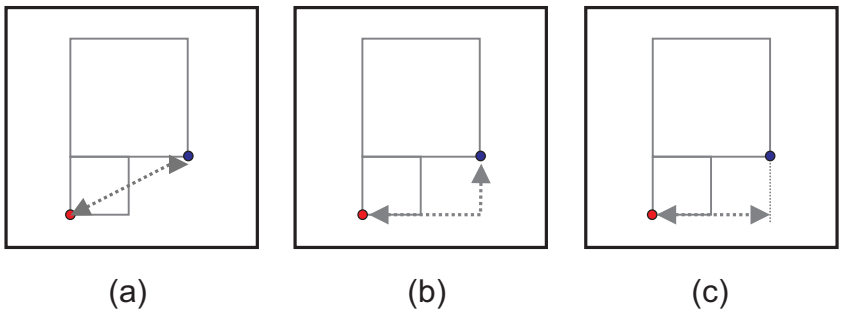

Figure 4. Distance metrics: Euclidean (**a**), Manhattan (**b**), and Chebyshev (**c**). The continuous lines represents the layout, and the points in red and blue represent the two points considered in our example of the three distances used. The dashed lines represent the value of the distance between these two points. The Euclidean distance is the usual shortest line that connect both points; the Manhattan distance is the horizontal euclidean distance plus the vertical Euclidean distance between them; and, finally, the Chebyshev distance is the maximum value between horizontal and vertical Euclidean distances.

The first step is to build an equivalent undirected graph $G = (V, E)$ containing the union of all the polygons' vertices and edges in the layout (see Figure 3). We applied the same strategy suggested by Silva et al. [15]. Figure 5 illustrates two possible paths in which cutting moves are represented as black edges, and simple moves are shown in red. The nodes and edges of the resultant graph portray the polygon points in surface space and the lines resulting from the meeting of two faces, respectively. In the final step, the solution is associated with a sequence of (cut or move) instructions processed by the cutting machine until the separation of the entire layout of pieces is finished.

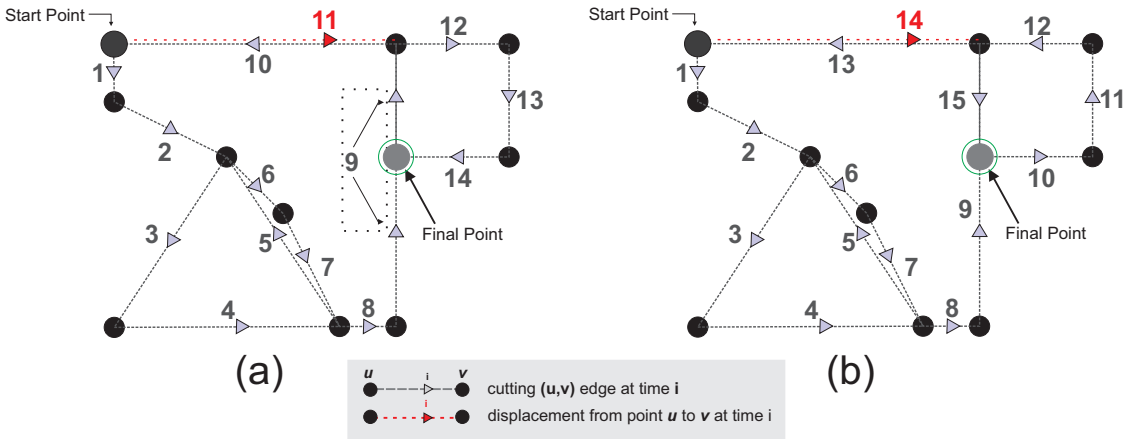

Figure 5. Examples of cut path for the layout in Figure 3. (**a**) A example of possible paths with 11 simple moves. (**b**) A example of possible paths with 14 simple moves.

In addition, this section presents a packing approach that uses a traditional genetic algorithm (GA) to tackle practical instances that exact models cannot address in a reasonable time. GAs are computational algorithms based on the principle of natural selection and survival through the fittest individuals in a similar way to the evolutionary processes in nature [37]. GA-based methods are among the prevalent approaches to cutting and packing problems [38,39] and cutting path problems [16,40].

The GA starts by generating a random group of individuals, which are represented by chromosomes or typically binary arrays. These structures are evaluated using a fitness function that measures the candidate solution's quality. A small percentage of the individuals with the highest fitness is copied into the next generation (elitism). The selection operator then chooses two individuals of the current generation to be combined with probability $txCross$ (crossover rate). Similarly, the mutation operator is applied to the offspring with probability $txMut$ (mutation rate) [41]. After the new population is generated, the stop condition (e.g., number of generations or genetic convergence) of the algorithm is tested. If it is not satisfied, the process is repeated.

In the proposed GA implementation, each individual is encoded by a chromosome represented by a vector $chrom(i) = (i = 1, 2, \ldots, n, n+1, \ldots, 2n)$, where n corresponds to the number of edges that must be cut from the input layout. The n's first positions represent the cutting order of each edge, and the remaining elements have binary values (0 or 1) expressing the direction of the process. Figure 6a illustrates an input layout. We emphasize that the entire cutting process tends to start from the original system of each device (Source) and return with the movement head at the end (Regress). In this work, we considered the point (0,0), because the machine where we applied the tests follows the same idea. Note that in Figure 6b, the coding process for each individual describes that the input layout has four edges and, therefore, the individuals' representation vector contains eight positions.

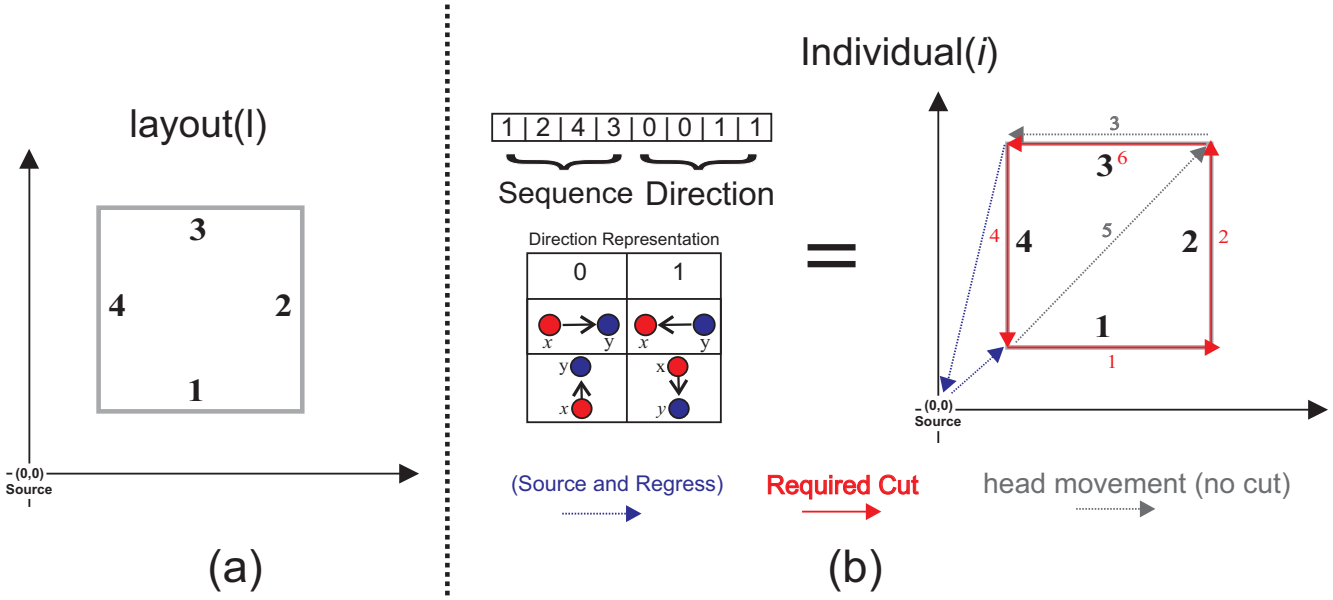

Figure 6. Example of an individual representation for input layout. (**a**) illustrates an input layout. (**b**) the coding process for each individual describes that the input layout has four edges and, therefore, the individuals' representation vector contains eight positions.

Moreover, the order ($i = 1, 2, 3, 4$) and direction ($i = 5, 6, 7, 8$) determine the plan of the complete cut. Therefore, when treating each element of the representation vector, we applied an offset assuming the shortest distance (Chebyshev) possible for cases in which the target node of one edge i does not match the origin node of the next $i + 1$. We assumed that 0 represents "left-to-right" or "bottom-to-up," in other words, for edge (u, v), we considered the initial point to visit u and the final point v, and 1 depicts the directions back.

The initial population was generated by shuffling a list of each cutting edge, while the rest of the positions with values 0s or 1s were drawn randomly under equal probabilities. The fitness of each individual was calculated by adding the time needed to cut, i.e., the time to cut the required edges (T_{cut}) and the motion of the head from the origin to the initial node of the layout and back from the head to the end of the complete cut ($t_{offsets}$). It was necessary to consider the speed π (cutting) and the other times μ (without cutting) to calculate the value of T_{cut}. Figure 7 illustrates the (i , $i+1$)-step of the fitness function. Note that (i) typifies each position of *chrom*, and the two squares represent the required cut edges (n), in other words, the input layout.

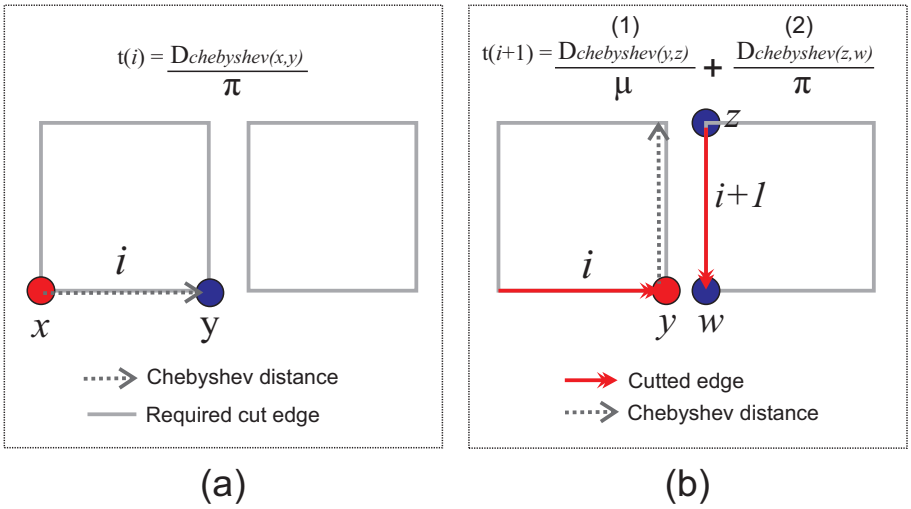

Figure 7. Fitness representation to $t(i)$ and $t(i+1)$. (**a**) illustrates the i-step to compute the fitness value for each individual. (**b**) supposed that the $chrom(i+1)$ was the edge (z,w) and that its direction position ($chrom(2n+(i+1))$) had value 1.

Figure 7a illustrates the i-step to compute the fitness value for each individual. The $chrom(i)$ is (x,y) edge and its position in the second part of the chromosome ($chrom(2n+i)$) contained value 0. The movement occured from node x to y (direction represented by 0), and, in this case, the edge (x,y) belonged to the set of required cutting edges. Thereby, to compute the time, $t(i)$ was divided by the Chebyshev distance for (x,y) to π (cut speed). The next stage, ($i+1$-step), supposed that the $chrom(i+1)$ was the edge (z,w) and that its direction position ($chrom(2n+(i+1))$) had value 1—see Figure 7b. In this condition, to account for the time $t(i+1)$, we needed to add the displacement time of the cutting device head from node y to z; for this, we applied the displacement speed (1) with the cutting time of the edge (z,w), and we used the cutting speed (2). Therefore, Equation (1) formalizes the fitness function.

$$fitness(chrom) = \sum_{i=1}^{n} t(chrom(i)) + t_{offsets} \quad (1)$$

The method for individual selection applied was q-Tournament [42]. In this procedure, a group of q individuals was randomly selected with population replacement. This group became part of a dispute in which the winner was determined according to the best fitness. The crossover operator, in general, recombines aspects of chromosomes and benefits the search for the solution space, directing the evolution process. The method applied to the proposed GA was partially matched crossover (PMX) [43]. The goal is to generate two children by combining pairs of values in a given range of the two parents and exchanging these indexes' values. This strategy was applied only to the n's first elements of the vector.

On the other hand, the remaining positions of the representation were recombined through the two-point crossover. This adaptation facilitates the manipulation of each part of the applied model's scopes, sequencing edges, and direction choices, respectively.

The mutation operator was verified to ensure an expansive scan of the state space and to contain the premature convergence (optimal locations) of the GA. Similar to the idea conceived for the crossover, we applied the *Shuffle Indexes* method to the first part of the chromosome and the *flip bit* mutation strategy for the directions portion.

Finally, we detailed all values for GA parameters like population size, crossover and mutation rate, stop criteria, and others in the results Section 4.

3.2. A BRKGA-Based Approach

Genetic algorithms with random keys (RKGA) were introduced by [44] to deal with combinatorial optimization problems involving representation adapted to sequencing. In an RKGA, each chromosome is represented as a vector of real numbers in the range $[0, 1[^n$, where n is the problem's dimension. A decoder receives a chromosome and maps it into a feasible problem solution. Resende [45] proposed the Biased Random-Key Genetic Algorithm (BRKGA), in which the individuals are selected in the recombination process, and it also contains the highlighted function of the probability of inheriting information from the parents (p_a).

This study used the BRKGA due to its tolerance to several optimization problems, assuming that two elements of its framework must be represented [46]: a compatible decoder, featured in this section, and a representative fitness function, trying to minimize Equation (1), for this problem. Figure 8 illustrates the BRKGA heuristic and its parameters: the size of the population (P), the proportion of the population in the elite (P_e), and the number of new random individuals that will be included in the new population (P_m).

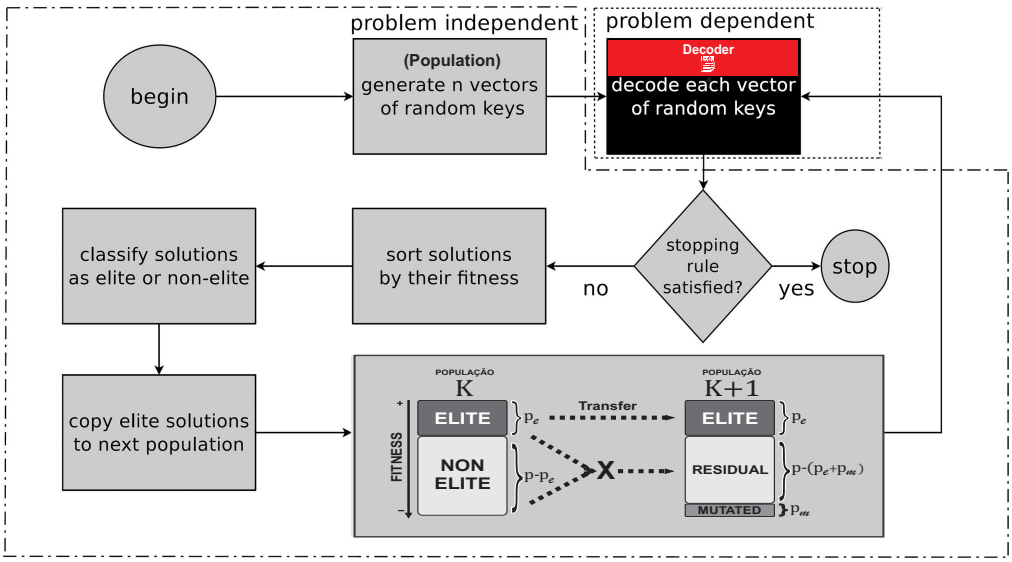

Figure 8. A flowchart of BRKGA and parameters.

An important characteristic of BRKGA is the parameterized uniform crossover [47]. In this crossover type, one of the parents is extracted from the elite group, while the second parent is extracted from the non-elite set. This enables the BRKGA heuristic to mitigate a premature genetic convergence. On the other hand, it requires the calibration of the p_a parameter that can affect the overall performance. This peculiar parameter of the BRKGA ensures that the child individual inherits more characteristics of an elite parent. One parent is always selected (with replacement) from the elite solutions group, and the probability that the child inherits the key of the elite parent >0.5. According to Resende [45], this is a differentiating factor in favor of faster convergence when compared to RKGA [44].

Each gene in the chromosome representation for individuals in the BRKGA contains a real value in the range [0.1). For example, Figure 9 illustrates how random keys represent alleles for each chromosome. In our proposed strategy, the value of each gene $chrom(i)$ was used by the decoder to determine the order and the cuts and movements' directions.

Therefore, the decoder operated as follows: first, the initial n genes for each chromosome are sorted upwards, corresponding to the cut order. Then, for each of the remaining genes $(n+1,\ldots,2n)$, the following function is used to determine the cuts' direction.

$$f(chrom(i)) = \begin{cases} 0, & \text{if } chrom(i) < 0.5 \\ 1, & \text{otherwise} \end{cases}$$

Figure 9 presents the encoding process from random keys to process an individual. Note that this process depends on problem specifications. For this reason, we illustrated this process separately from Figure 8. The input module saves the SVG file (layout) information like the edges for a cut. The dimension of the array depends on such a value. It is noteworthy that the part of the vector representing the directions does not participate in the keys' ordering, being decoded only by the function *f(chrom)*. In this way, sorting random keys (ascending order of keys) results in the sequencing of edges visits. After that, the decoder presents a solution containing a sequence of movements for the input layout and the fitness function defined by Equation (1), which can calculate the time required for each individual.

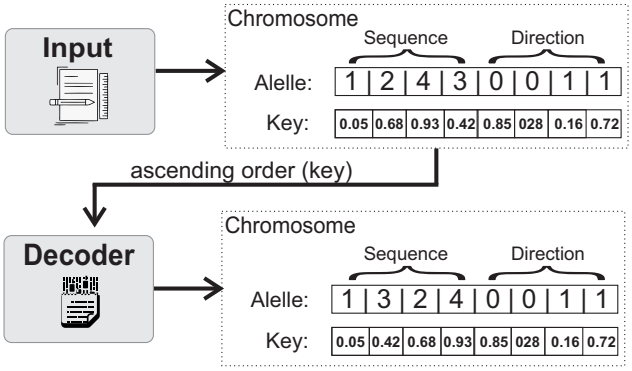

Figure 9. Decoder process to BRKGA proposed.

In this context, we use Section 4 to present our GA and BRKGA approaches' parameters and the cutting(π) and moving speeds(μ), inherent in MTCP, for all computational tests applied.

4. Results and Discussion

The computational experiments were conducted in an Intel Xeon 3.80 GHz machine with eight cores, 8GB RAM, and an Ubuntu 18.08 operational system. The GA and the BRKGA algorithm were implemented in Python 3.7. As input data, all dataset instances were layouts extracted through the algorithm applied by Amaro et al. [12]. The result was a file in SVG format. The following sub-sections present the characteristics of the benchmark instances (Section 4.1), a comparison between GA and BRKGA (Section 4.2), and a comparison between BRKGA (best approach considered) and a commercial laser cut software in practical situations (Section 4.3).The reader can access the complete information of the data used and constructed in this section through the link presented in the Data Availability Statement.

4.1. Instances

A set of 50 problem instances was used to evaluate the presented approaches. These instances can be categorized according to the possible presence of empty space between the pieces in the input layout, being either connected (C) (Figures A1 and A2 in the Appendix A.1) or separated (S) (Figures A3–A5 in the Appendix A.2). The presence of space between items in the latter group aimed to enable the use of support, which is quite common in some material cutting applications. The dataset was generated by the nesting approach presented in [12].

Several hyper-parameter settings (27) were tested for the GA and the BRKGA heuristics, which was executed ten times for each instance. Table 1 presents the instances, the number of vertices, both the number of edges (original and adapted layouts with the joint of the edges), and the number of polygons.

Table 1. Characteristics of instances used in the computational experiments.

Instance	Vertices		Edges				Items	
			Original		Adapted			
	(C)	(S)	(C)	(S)	(C)	(S)	(C)	(S)
albano	156	164	164	164	173	164	24	24
blaz1	39	44	44	44	46	44	7	7
blaz2	70	80	88	80	89	80	14	13
blaz3	97	132	132	132	130	132	21	21
dighe1	20	54	46	54	38	54	15	15
dighe2	20	46	38	46	30	46	10	10
fu	37	43	43	43	51	43	12	12
inst_01_10pol	20	40	39	40	29	40	10	10
inst_01_16pol	27	128	64	128	42	128	16	32
inst_01_2pol	7	8	8	8	8	8	2	2
inst_01_3pol	8	12	12	12	10	12	3	3
inst_01_4pol	10	16	16	16	13	16	4	4
inst_01_5pol	12	20	19	20	16	20	5	5
inst_01_6pol	13	24	23	24	18	24	6	6
inst_01_7pol	15	28	27	28	21	28	7	7
inst_01_8pol	16	32	31	32	23	32	8	8
inst_01_9pol	18	36	35	36	26	36	9	9
inst_01_26pol	210	264	264	264	237	264	66	66
rco1	33	36	36	36	40	36	7	7
rco2	62	72	72	72	81	72	14	14
rco3	82	108	108	108	116	108	21	21
shapes2	68	70	70	70	78	70	8	8
shapes4	127	140	140	140	147	140	16	16
spfc_instance	55	55	55	55	63	55	11	11
trousers	350	388	388	388	424	388	64	64

Some instances present a different number of vertices and edges due to the application of the join procedure and split edges in the original input layout. It is necessary to highlight that joining segments are treated as particular cases of splitting edges. Figure 10 illustrates the input data format (SVG file) and the output obtained by this methodology. The algorithm of preprocessing data, extracted by [15], was used to convert the input file into a graph before addressing the MTCP through the proposed adapted metaheuristic approaches. Then, the sequence of cuts and moves wasvalidated (output).

4.2. GA and BRKGA Hyper-Parameter Configuration

GA and BRKGA hyper-parameter configurations used in the experiments were selected after preliminary tests: population of 10,000, 5000, and 1000 individuals; crossover of 70%, 75%, and 80%; and mutation of 10%, 15%, and 20%. For the elite group in BRKGA, values of 30%, 20%, and 10%; mutated population of 10%, 15%, and 20%; p_a of 70%. The stopping criteria wre met when there was no improvement in the best solution found for 100 generations or when the execution time exceeded 300 s.

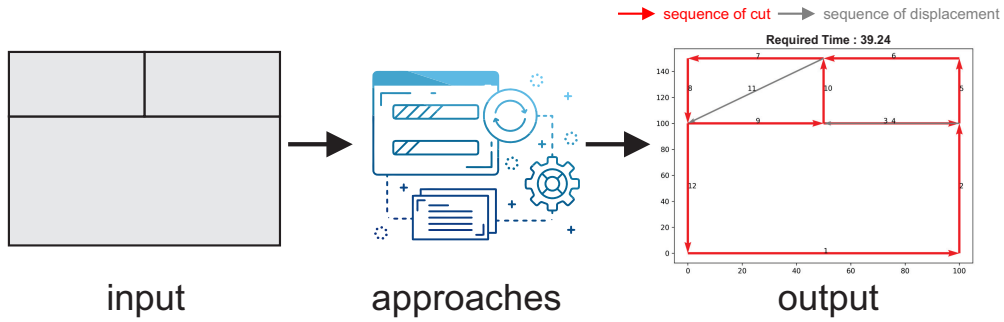

Figure 10. Input and output representation.

Tables A1 (see Appendix B.1) and A2 (see Appendix B.2) summarize the computational results. For every connect and separate instance, the results include the objective function value (FO), the elapsed time for the best execution, the average objective values, and the elapsed computational time. It is possible to observe that the BRKGA is more suitable than the standard GA for addressing the tested instances. Every hyper-parameter setting was tested for both GA and BRKGA, and each type of layout, connected (C) and separated (S).

Figure 11 presents the number of times that a BRKGA hyper-parameter setting achieved the best solution found. For connected layouts (C), the best GA configuration addressing *relation-1* (instances/executions) was obtained using the following hyper-parameters after 49 times: P = 10,000, $txCross = 0.8$, and $txMut = 0.1$. In the separated instances group (S), the best solution was obtained after 20 executions, and the configuration was the following: P = 5000, $txCross = 0.75$, and $txMut = 0.2$.

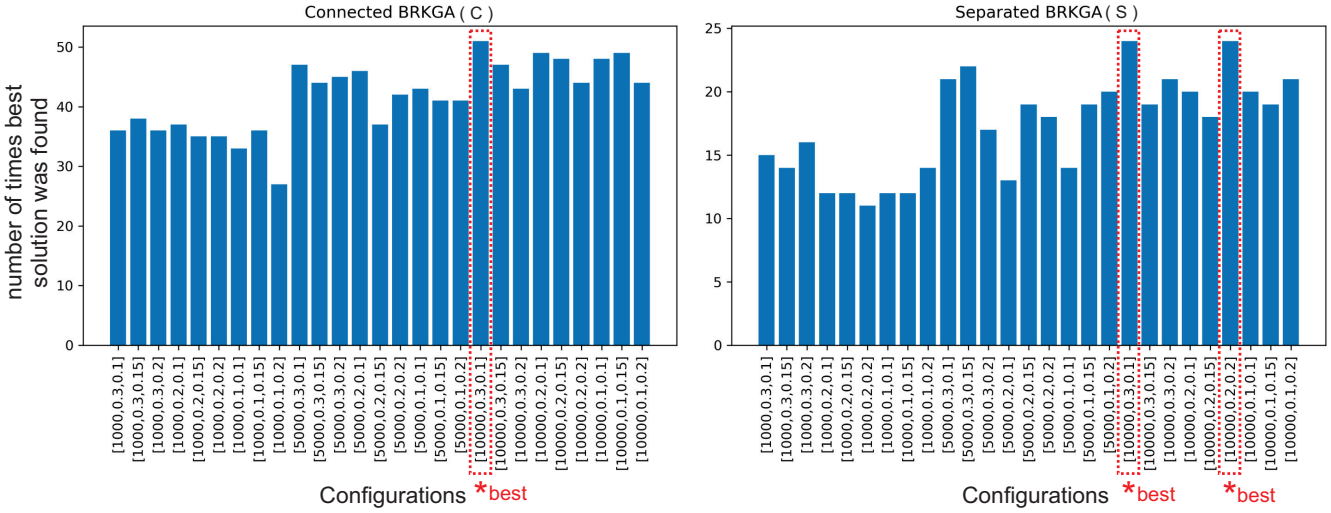

Figure 11. Analysis of the proposed configurations for BRKGA (*relation-1*).

Figure 12 presents the minimum time required to reach the best value in each execution. It is possible to observe that, for each instance, the objective function value was above the average best. A consideration of the results exposes that it is not suitable to define the best configuration for *relation-1*. While it achieves the best result in all executions for some instances, it led to the below-average value for the remaining.

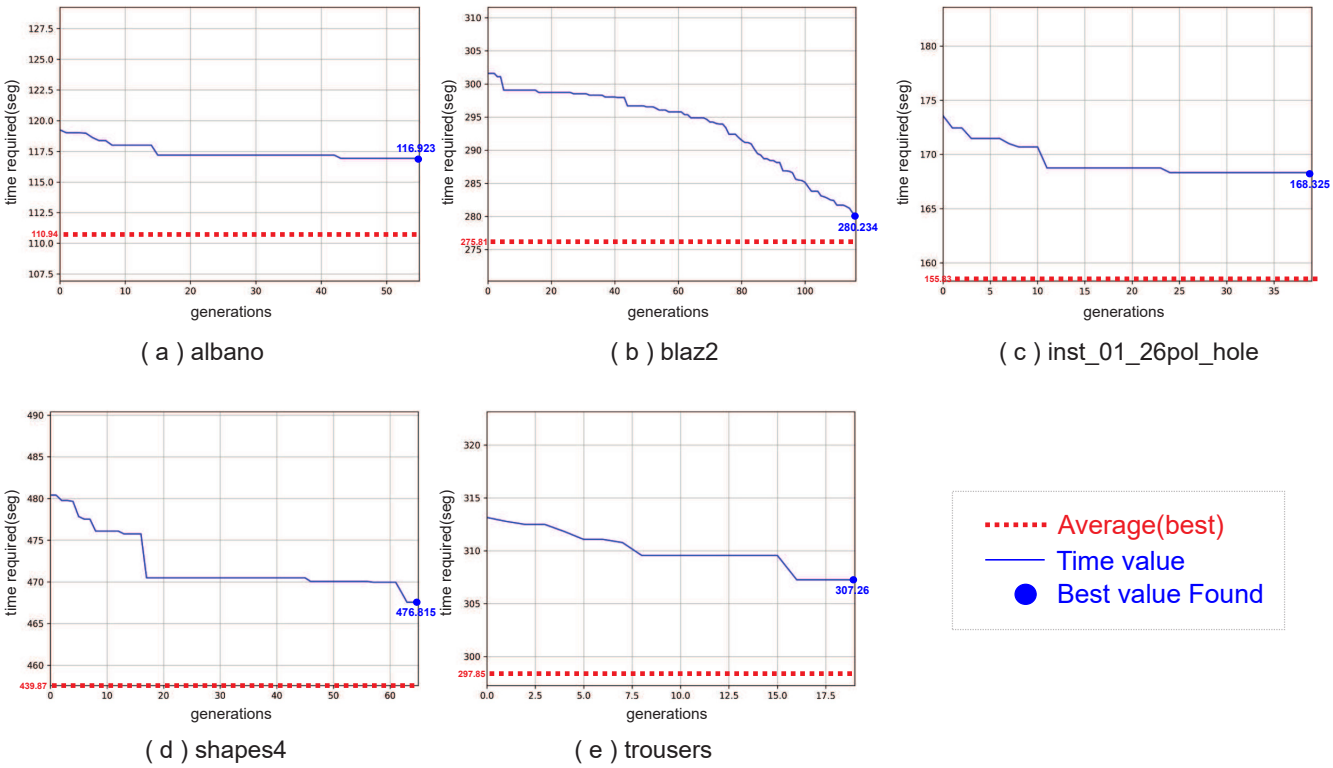

Figure 12. Progression of the objective values through generations when addressing connected instances. BRKGA hyper-parameter: P = 10,000, P_e = 0.30, and P_m = 0.1.

Next, we defined *relation-2* considering the number of instances each configuration achieved the best FO value for at least one execution. Figure 13 presents the best hyper-parameter configuration according to the highest average value for connected and separate layouts. Therefore, the best parameters to connected instances were P = 5000, P_e = 0.30, P_m = 0.1; P = 10,000, P_e = 0.10, P_m = 0.1; P = 10,000, P_e = 0.10, and P_m = 0.2. For separate instances, the best hyper-parameters were P = 10,000, P_e = 0.20, and P_m = 0.2. Figure 14 presents the progression of the BRKGA optimal value at each generation.

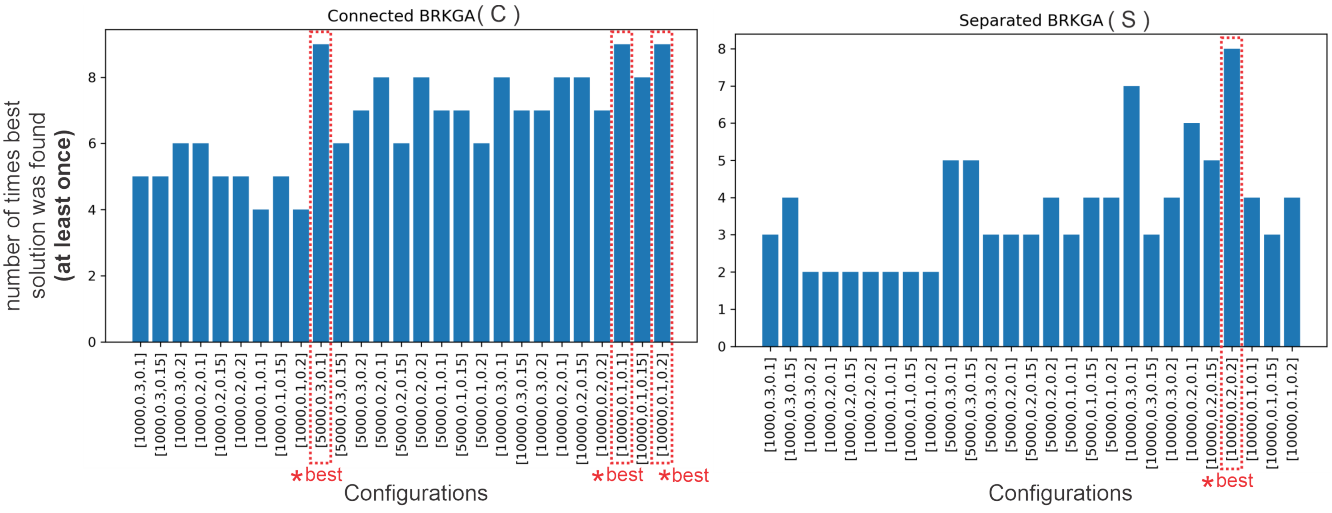

Figure 13. Analysis of the proposed configurations for BRKGA (*relation-2*).

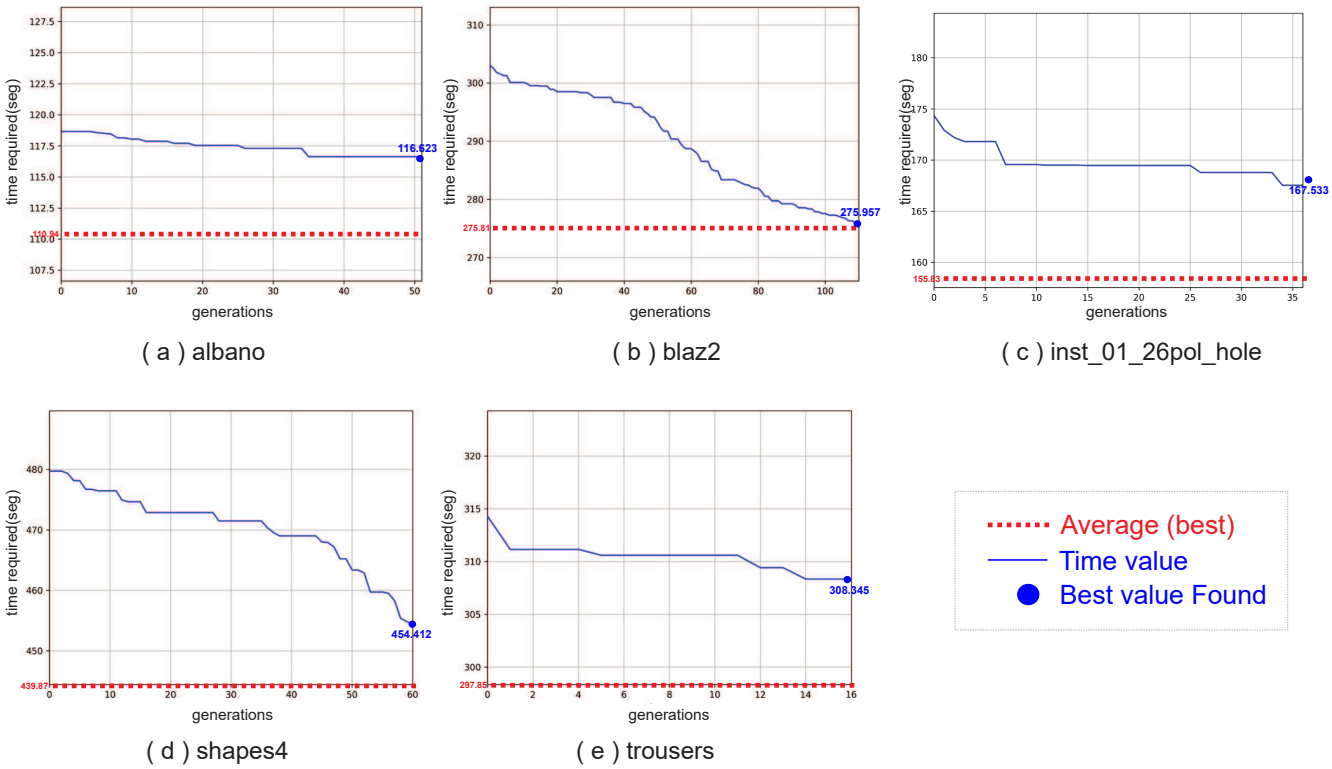

Figure 14. Evolution process defined for time limit reached instances in C-group. Configuration of BRKGA: [P = 10,000, P_e = 0.20, and P_m = 0.2].

Lastly, we defined *relation-3* setting the configuration using each parameter option that occurred more frequently in the best solutions (modal value): P = 10,000, P_e = 0.3, and P_m = 0.10. A comparison between vertex numbers (V) and population size (P) exposed that the configuration that hyper-parameter settings with population equal to 10,000 failed to achieve the best results for instances with $V \geq 70$. For example, for the instances *inst_01_26pol* ($V = 210$) and *trousers* ($V = 350$), the value of P in the configuration that presented the best solution was 1000 for both cases. Therefore, the following hyper-parameter was selected for further experiments: P = 5000, $P_e = 0.30$, and $P_m = 0.1$. The progression of the BRKGA is shown in Figure 15 and Table A3 (see Appendix C.1) and Table A4 (see Appendix C.2). Since the configuration failed to reach the average result for both connected and separate instances, it is possible to affirm that increasing the timeout would enable better solutions.

4.3. Comparing BRKGA and a Commercial Laser Cut Machine Software

This section presents a practical case from industry comparing BRKGA using the best hyper-parameter setting (see Section 4.2) and commercial software for laser cutting machines. The characteristics of the device are shown as follows. PRISMA machine (https://www.automatisa.com.br/en/ (accessed on 8 October 2021)) produced *Automatisa Laser Solutions*, was 60 W, and had a maximum working area of 900 × 600 mm. The speeds π (cut) and μ (displacement) were 16.67 mm/s and 400 mm/s, respectively. Then, BRKGA was executed for all the instances, and the generated outputs (sequence of edges for cutting) were exported as SVG files. After, a similar process was repeated using the commercial software. Finally, we used the proposed instances and assigned them as input for execution to compute the machine's software (LaserCut).

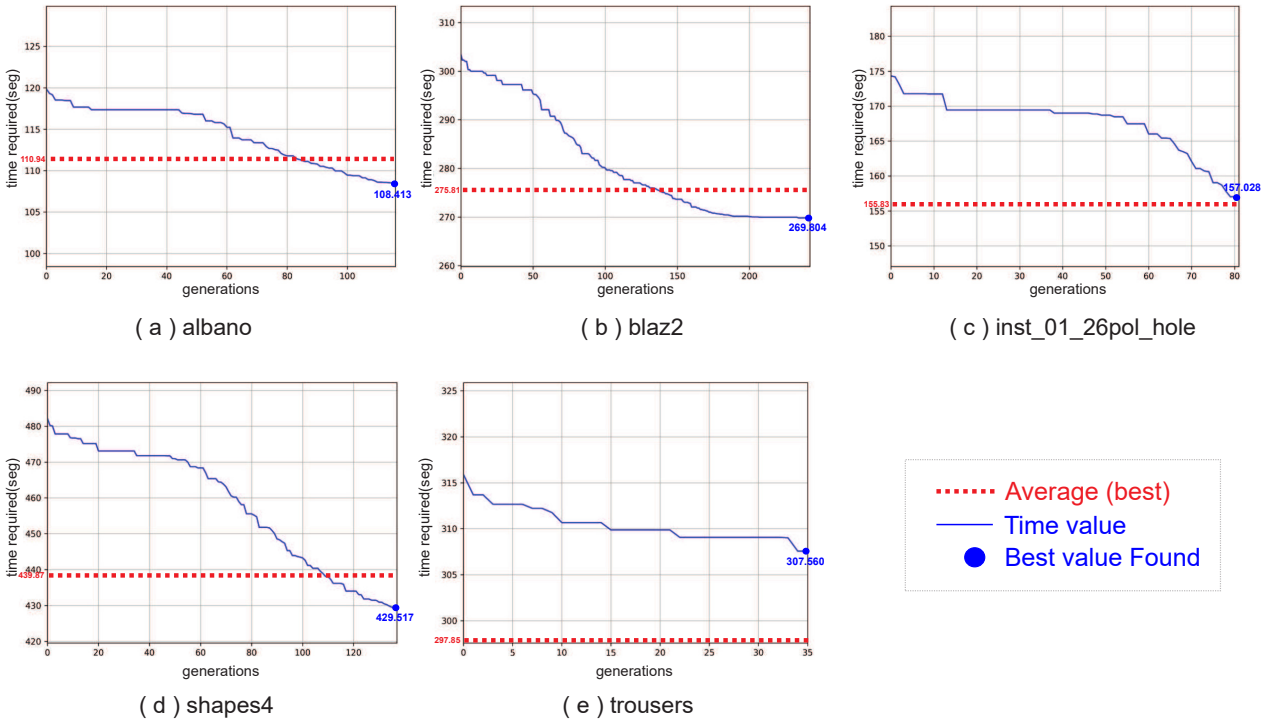

Figure 15. Evolution process defined for time limit reached instances in C-group. Configuration of BRKGA: [P = 5000, $P_e = 0.30$, and $P_m = 0.1$].

Table A5 (see Appendix D.1) presents a comparison between the introduced BRKGA approach with parameters P = 5000, $P_e = 0.30$, and $P_m = 0.1$, and LaserCut. For 50 instances analyzed, BRKGA achieved a gained time in 47 s, i.e., an improvement of 94%. It is noteworthy that for instances with separated items, the improvement was inferior to those for instances with connected items.

It is possible to notice that the BRKGA approach achieved a time of 256.56 s for the connected group and 258.76 s for the separated group—see Tables A1 and A2. The respective settings applied to these results were: P = 1000, $P_e = 0.30$, and $P_m = 0.1$ and P = 1000, $P_e = 0.30$, and $P_m = 0.15$. It is noteworthy that, for several instances, BRKGA performed better than LaserCut. Interestingly, the results indicate an apparent relationship between the population size and the number of vertices (Table 1) in the input layout.

5. Conclusions

This study introduced a case of the MTCP problem, which more closely resembles real-world scenarios by distinguishing cutting and moving speeds. The cutting and the sliding rates for the equipment are essential parameters and, despite their relevance in real-world machinery, such features are often ignored in the literature. This study presented the minimum time cut path problem, including two presented evolutionary approaches, i.e., the genetic algorithm (GA) and biased random-key genetic algorithm (BRKGA) methods, which are suitable for addressing larger instances.

Computational tests presented extensive hyper-parameter tuning using a large set of instances extracted from the literature, which can be leveraged in future research. For all instances analyzed, the proposed methodology achieved a gain in time of 47 s and an improvement in 94% of the tests performed.

However, we noticed a decrease in performance when the input layouts had a relatively large number of vertices. This fact indicates a correlation between this parameter and the population size. Moreover, a suitable approach would consist of splitting the layout into parts, considering cutoff distances, so that the individual's representation can also be segmented. Hence, we would build N subgroups of edges, and the evaluation would be limited in finding the best sequence for each N. Lastly, we would add the endpoint

times to the beginning of all predefined N by finding a feasible solution and enabling parallel processing.

Future work will investigate how clustering methods such as K-Means can be integrated into the current approach to find feasible and quality solutions when addressing instances with a more significant number of parts positioned in a layout. This strategy will be integrated into the mathematical model. Another research direction is the investigation of an alternative representation for candidate solutions in the BRKGA algorithm. In the recent literature, there have been examples of methodology that allow the use of a purely discrete representation and the possibility of combining the discrete solutions using vector-like operations, similar to what is usually done for continuous solutions as in the random-key encoding [48–50].

Author Contributions: Conceptualization, B.A.J.; data curation, M.C.S. and G.N.d.C.; methodology, L.J.P.d.A., B.A.J. and M.C.S.; software, G.N.d.C.; validation, B.A.J.; writing—original draft, L.J.P.d.A., B.A.J. and P.R.P.; writing—review & editing, L.J.P.d.A. and B.A.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://github.com/GUIKAR741/metaheuristics-minimun-path (accessed on 8 October 2021). The folder "algorithms" includes the codes of the two approaches developed for this study. The folder "instances" holds all input SVG layouts. The folder "results" presents the numerical experimental data. The folder "result-evolution process" contains the evolution graph of the best solution (Y) with the number of generations (X), and the folder "results-final draw" shows the ten images of the path sequence found (each instance × execution), just for the BRKGA approach with hyper-parameters [P = 5000, $P_e = 0.30$, and $P_m = 0.1$].

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BRKGA	Biased random-key genetic algorithm
CCP	Continuous cutting problem
C&P	Cutting and packing
CPD	Cut path determination
ECP	Endpoint cutting problem
GA	Genetic algorithm
GTSP	Generalized traveling salesman problem
ICP	Intermittent cutting problem
MTCP	Minimum time cut path
NRP	Node routing problem
SVG	Scalable vector graphics
TPP	Touring polygons problem
TSP	Traveling salesman problem
TSP-N	Traveling salesman problem with neighborhoods

Appendix A

Appendix A.1

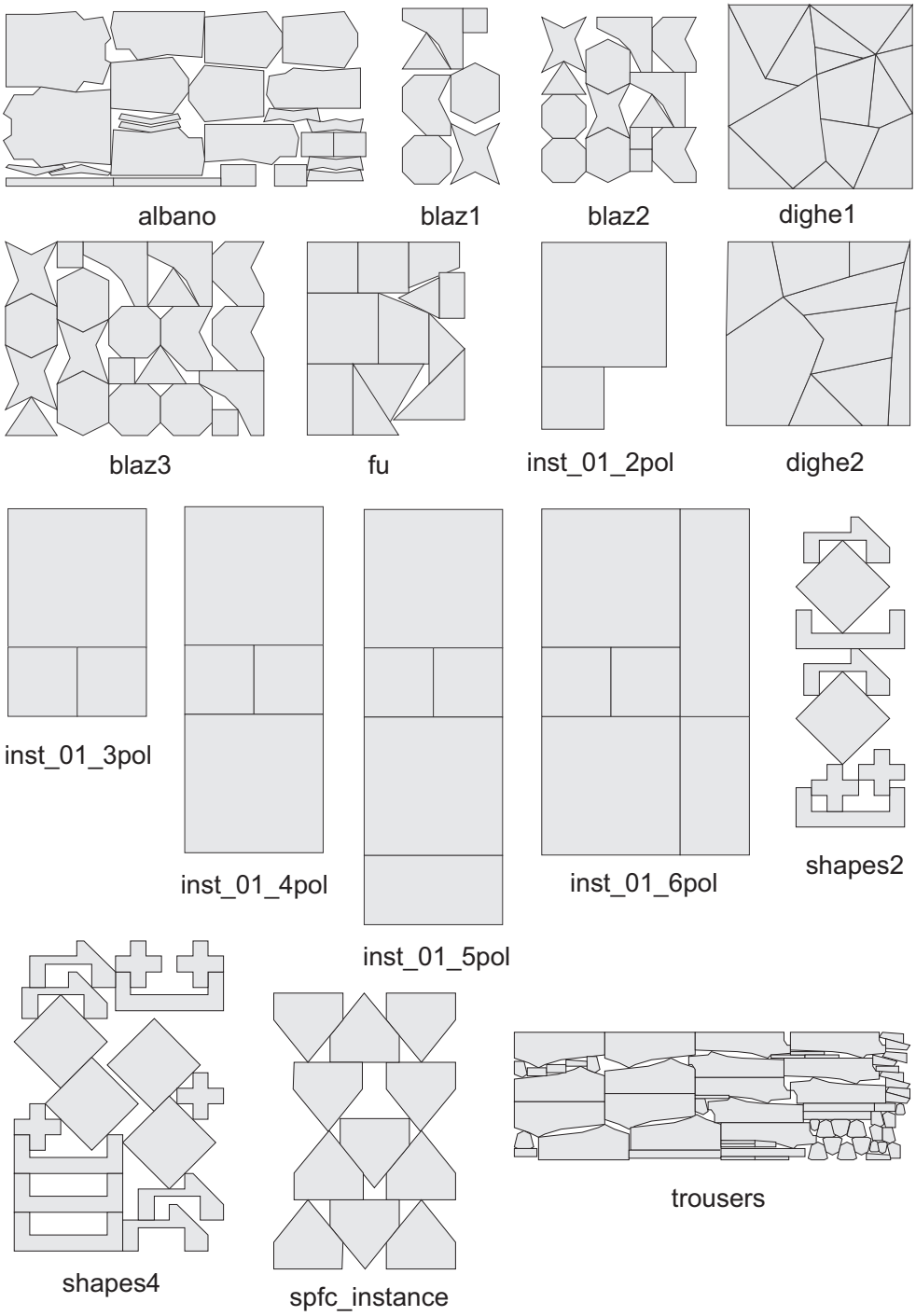

Figure A1. First batch of instances with connected items.

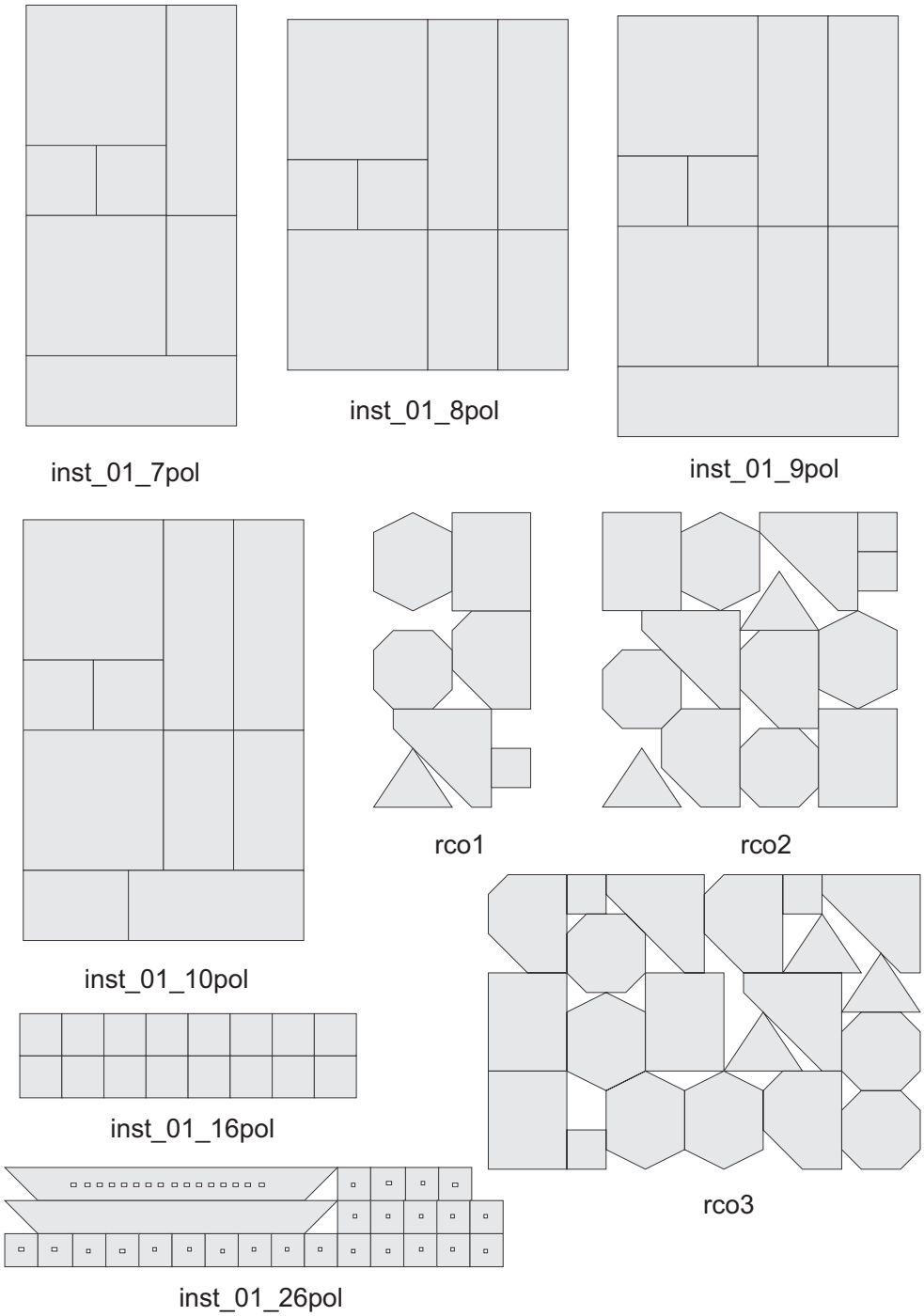

Figure A2. Second batch of instances with connected items.

Appendix A.2

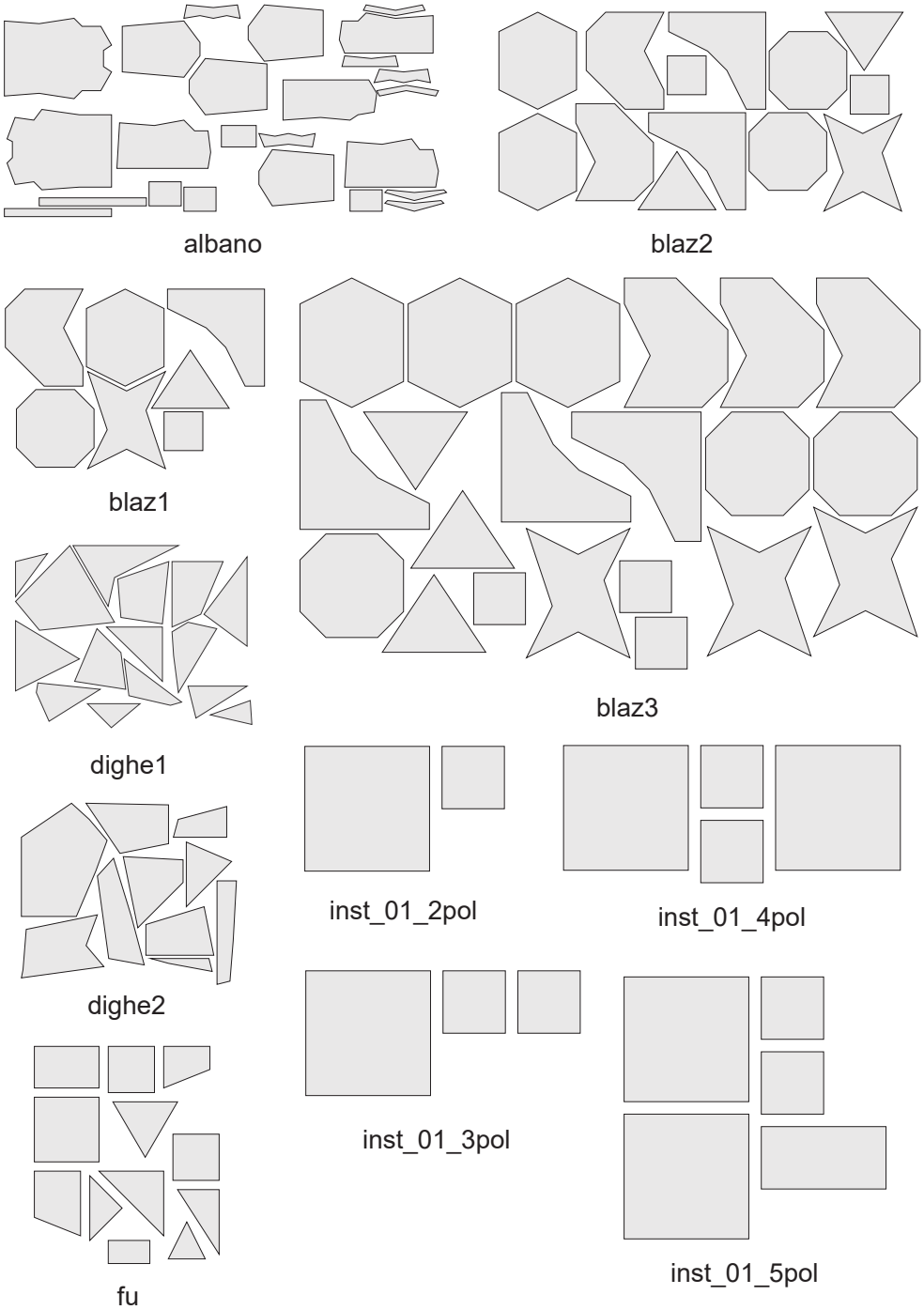

Figure A3. First batch of instances with separated items.

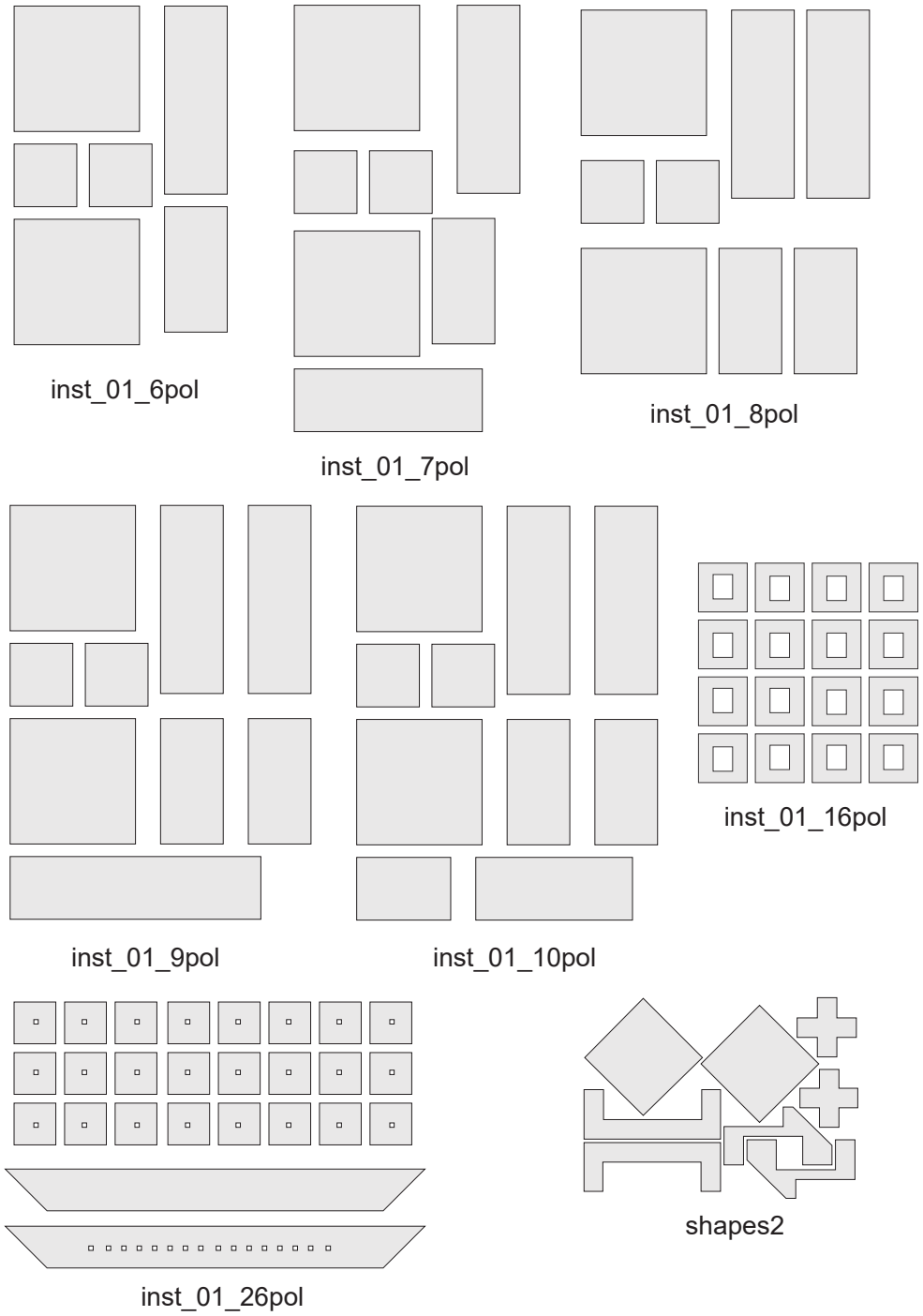

Figure A4. Second batch of instances with separated items.

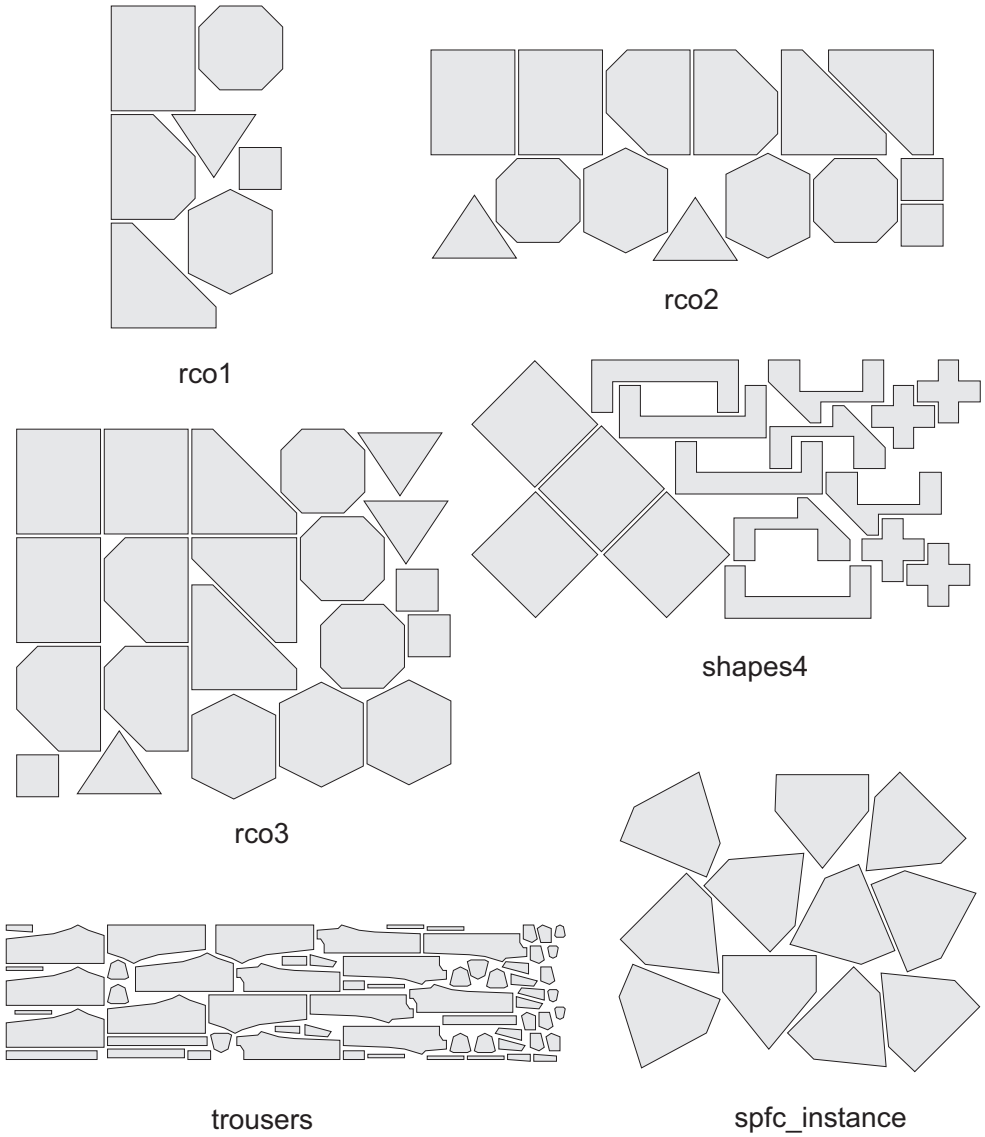

Figure A5. Third batch of instances with separated items.

Appendix B

Appendix B.1

Table A1. GA and BRKGA configurations for the connected instances (C).

Instances		GA		BRKGA	
		FO	TIME	FO	TIME
albano	Best	106.38	300 s	104.62	300 s
	Average	114.5	300 s	110.94	300 s
blaz1	Best	154.88	300 s	154.22	294 s
	Average	158.98	300 s	155.2	300 s
blaz2	Best	275.27	189 s	269.72	300 s
	Average	290.93	300 s	275.81	300 s
blaz3	Best	428.38	300 s	419.32	300 s
	Average	454.98	300 s	435.34	72 s
dighe1	Best	70.47	300 s	70.53	124 s
	Average	71.09	25 s	70.87	19 s
dighe2	Best	53.87	205 s	53.87	100 s
	Average	54.42	34 s	54.03	15 s
fu	Best	23.86	300 s	23.82	164 s
	Average	24.23	61 s	23.98	26 s
inst_01_10pol	Best	126.97	131 s	126.97	158 s
	Average	127.47	188 s	127.34	14 s
inst_01_16pol	Best	76.73	211 s	76.58	300 s
	Average	78.08	26 s	77.48	20 s
inst_01_2pol	Best	33.11	4 s	33.11	4 s
	Average	33.11	20 s	33.11	23 s
inst_01_3pol	Best	39.24	5 s	39.24	4 s
	Average	39.24	25 s	39.24	27 s
inst_01_4pol	Best	57.36	5 s	57.36	6 s
	Average	57.36	32 s	57.36	34 s
inst_01_5pol	Best	69.61	8 s	69.61	41 s
	Average	69.73	7 s	69.61	86 s
inst_01_6pol	Best	81.73	139 s	81.73	103 s
	Average	81.85	62 s	81.85	48 s
inst_01_7pol	Best	96.85	11 s	96.85	9 s
	Average	96.98	69 s	97.23	8 s
inst_01_8pol	Best	105.72	229 s	105.72	127 s
	Average	106.10	289 s	106.10	10 s
inst_01_9pol	Best	123.97	210 s	123.97	136 s
	Average	124.47	85 s	124.47	10 s
inst_01_26pol_hole	Best	142.79	300 s	137.50	300 s
	Average	162.66	300 s	155.83	300 s
rco1	Best	140.52	300 s	140.17	248 s
	Average	143.11	32 s	141.01	233 s
rco2	Best	274.43	264 s	270.16	300 s
	Average	288.92	300 s	275.06	231 s
rco3	Best	403.65	300 s	393.92	300 s
	Average	429.06	300 s	407.85	64 s
shapes2	Best	218.02	210 s	214.73	300 s
	Average	228.10	300 s	218.98	52 s
shapes4	Best	425.87	300 s	419.83	300 s
	Average	461.70	300 s	439.87	300 s
spfc_instance	Best	144.94	300 s	143.53	201 s
	Average	150.18	300 s	145.56	183 s
trousers	Best	271.78	300 s	256.56	300 s
	Average	301.36	300 s	297.85	300 s

Appendix B.2

Table A2. GA and BRKGA configurations for the separated instances (S).

Instances		GA		BRKGA	
		FO	TIME	FO	TIME
albano	Best	109.83	300 s	107.23	300 s
	Average	119.43	300 s	113.52	300 s
blaz1	Best	162.57	279 s	161.86	300 s
	Average	166.06	300 s	163.13	300 s
blaz2	Best	296.30	300 s	291.57	300 s
	Average	309.95	300 s	295.52	243 s
blaz3	Best	499.18	300 s	489.88	300 s
	Average	531.40	300 s	506.38	77 s
dighe1	Best	96.53	300 s	96.03	300 s
	Average	98.65	37s	97.47	35s
dighe2	Best	78.91	90 s	78.47	300 s
	Average	80.18	56 s	79.65	23 s
fu	Best	28.04	297 s	27.95	295 s
	Average	28.69	300 s	28.09	23 s
inst_01_10pol	Best	193.18	300 s	192.61	149 s
	Average	196.74	300 s	194.47	27 s
inst_01_16pol	Best	173.04	300 s	171.16	300 s
	Average	180.86	83 s	176.69	68 s
inst_01_2pol	Best	36.04	4 s	36.04	4 s
	Average	36.04	21 s	36.04	25 s
inst_01_3pol	Best	48.09	6 s	48.09	33 s
	Average	48.09	29 s	48.09	33 s
inst_01_4pol	Best	72.13	44 s	72.13	90 s
	Average	72.33	7 s	72.38	7 s
inst_01_5pol	Best	90.13	60 s	90.13	56 s
	Average	90.38	13 s	90.73	9 s
inst_01_6pol	Best	114.2	193 s	114.2	151 s
	Average	114.65	78 s	114.62	72 s
inst_01_7pol	Best	138.31	250 s	138.31	180 s
	Average	139.23	25 s	139.59	12 s
inst_01_8pol	Best	156.49	264 s	156.44	200 s
	Average	157.74	272 s	157.81	21 s
inst_01_9pol	Best	186.61	300 s	186.41	273 s
	Average	189.36	27 s	187.83	16 s
inst_01_26pol_hole	Best	198.97	300 s	192.89	295 s
	Average	216.82	300 s	205.02	300 s
rco1	Best	162.74	300 s	162.59	219 s
	Average	164.99	25 s	163.12	191 s
rco2	Best	322.09	300 s	319.31	300 s
	Average	334.81	300 s	321.63	300 s
rco3	Best	489.66	175 s	480.36	300 s
	Average	515.63	300 s	492.81	59 s
shapes2	Best	229.47	222 s	227.79	300 s
	Average	238.31	52 s	230.85	42 s
shapes4	Best	455.2	300 s	447.99	300 s
	Average	481.66	152 s	461.88	300 s
spfc_instance	Best	148.79	300 s	147.33	300 s
	Average	153.36	300 s	149.28	157 s
trousers	Best	303.51	300 s	285.76	300 s
	Average	338.13	300 s	330.06	300 s

Appendix C

Appendix C.1

Table A3. Connected instances (C) for considered configurations. Relation 1: P = 10,000; Pe = 30%; Pm = 10%. Relation 2: P = 10,000; Pe = 20%; Pm = 20%. Relation 3: P = 5000; Pe = 30%; Pm = 10%.

Instances		Relation 1		Relation 2		Relation 3	
		FO	TIME	FO	TIME	FO	TIME
albano	Best	116.92	300 s	116.62	300 s	**108,41**	**300 s**
	Average	117.12	300 s	116.92	300	**111.41**	**300 s**
blaz1	Best	**154.29**	**300 s**	**154.29**	**300 s**	154.64	300 s
	Average	**154.71**	**300 s**	154.74	277 s	**154.71**	**141 s**
blaz2	Best	280.23	300 s	275.95	300 s	**269.8**	**300 s**
	Average	281.7	300 s	277.85	300 s	**270.91**	**300 s**
blaz3	Best	457.75	300 s	450.12	300 s	**426.21**	**300 s**
	Average	460.13	300 s	453.97	300 s	**428.21**	**300 s**
dighe1	Best	**70.54**	**281 s**	**70.53**	**269 s**	**70.53**	**124 s**
	Average	70.73	277 s	70.69	300 s	**70.69**	**128 s**
dighe2	Best	**53.88**	**222 s**	53.90	202 s	53.90	92 s
	Average	**53.96**	**191 s**	54	186 s	54	93 s
fu	Best	23.84	300 s	23.83	300 s	**23.82**	**164 s**
	Average	**23.88**	**300 s**	23.90	300 s	23.91	168 s
instance_01_10pol	Best	**126.97**	**202 s**	**126.97**	**204 s**	**126.97**	**85 s**
	Average	127.22	175 s	**127.09**	**183 s**	127.22	90 s
instance_01_16pol	Best	**76.66**	**300 s**	**76.66**	**279 s**	76.73	135 s
	Average	76.81	300 s	**76.73**	**294 s**	76.88	130 s
instance_01_2pol	Best	**33.11**	**48 s**	**33.11**	**50 s**	**33.11**	**23 s**
	Average	**33.11**	**48 s**	**33.11**	**50 s**	**33.11**	**23 s**
instance_01_3pol	Best	**39.24**	**55 s**	**39.24**	**57 s**	**39.24**	**25 s**
	Average	**39.24**	**57 s**	**39.24**	**58 s**	**39.24**	**26 s**
instance_01_4pol	Best	**57.36**	**69 s**	**57.36**	**69 s**	**57.36**	**32 s**
	Average	**57.36**	**72 s**	**57.36**	**72 s**	**57.36**	**34 s**
instance_01_5pol	Best	**69.61**	**86 s**	**69.61**	**84 s**	**69.61**	**40 s**
	Average	**69.61**	**91 s**	**69.61**	**90 s**	**69.61**	**40 s**
instance_01_6pol	Best	**81.73**	**105 s**	**81.73**	**102 s**	**81.73**	**45 s**
	Average	**81.73**	**107 s**	**81.73**	**108 s**	**81.73**	**52 s**
instance_01_7pol	Best	**96.85**	**156 s**	**96.85**	**165 s**	**96.85**	**84 s**
	Average	**96.98**	**152 s**	**96.98**	**148 s**	**96.98**	**152 s**
instance_01_8pol	Best	**105.72**	**138 s**	**105.72**	**140 s**	**105.72**	**60 s**
	Average	105.85	148 s	**105.72**	**142 s**	105.85	68 s
instance_01_9pol	Best	**123.97**	**165 s**	**123.97**	**249 s**	**123.97**	**70 s**
	Average	**124.10**	**173 s**	**124.10**	**175 s**	**124.10**	**74 s**
instance_artificial_01_26pol_hole	Best	168.32	300 s	167.53	300 s	**157.02**	**300 s**
	Average	169.03	300 s	168.31	300 s	**165.78**	**300 s**
rco1	Best	**140.17**	**248 s**	140.66	251 s	140.52	126 s
	Average	**140.59**	**264 s**	140.94	263 s	140.73	113 s
rco2	Best	276.67	300 s	274.08	300 s	**270.86**	**300 s**
	Average	278.63	300 s	274.85	300 s	**271.63**	**300 s**
rco3	Best	421.50	300 s	417.93	300 s	**395.32**	**300 s**
	Average	427.17	300 s	420.59	300 s	**398.33**	**300 s**
shapes2	Best	216.77	300 s	215.17	300 s	**215.27**	**274 s**
	Average	218.61	300 s	216.53	300 s	**216.40**	**281 s**
shapes4	Best	467.54	300 s	454.41	300 s	**429.51**	**300 s**
	Average	469.71	300 s	464.67	300 s	**432.10**	**300 s**
spfc_instance	Best	143.90	300 s	**143.63**	**300 s**	143.69	244 s
	Average	144.32	300 s	144.17	300 s	**144.16**	**271 s**
trousers	Best	307.26	300 s	308.34	300 s	**307,56**	**300 s**
	Average	309.44	300 s	309.46	300 s	**308.75**	**300 s**

Appendix C.2

Table A4. Separated instances (S) for considered configurations. Relation 1: P = 10,000; Pe = 30%; Pm = 10%. Relation 2: P = 10,000; Pe = 20%; Pm = 20%. Relation 3: P = 5000; Pe = 30%; Pm = 10%.

Instances		Relation 1		Relation 2		Relation 3	
		FO	TIME	FO	TIME	FO	TIME
albano	Best	121.43	300 s	121.23	300 s	**111.91**	**300 s**
	Average	122.29	300 s	122.13	300 s	**115.86**	**300 s**
blaz1	Best	162.19	300 s	**161.86**	**300 s**	162.31	171 s
	Average	162.64	300 s	**162.66**	**300 s**	162.80	173 s
blaz2	Best	297.44	300 s	293.96	300 s	**291.57**	**300 s**
	Average	299.09	300 s	295.32	300 s	**292.68**	**300 s**
blaz3	Best	528.67	300 s	522.09	300 s	**492.80**	**300 s**
	Average	532.83	300 s	525.17	300 s	**496.80**	**300 s**
dighe1	Best	96.03	300 s	96.25	300 s	**96.21**	**194 s**
	Average	**96.22**	**300 s**	96.45	300 s	96.30	202 s
dighe2	Best	**78.47**	**300 s**	78.78	300 s	78.64	152 s
	Average	78.92	300 s	**78.91**	**300 s**	79.07	159 s
fu	Best	**27.96**	**300 s**	27.97	248 s	28.03	151 s
	Average	**28.03**	**291 s**	28.04	263 s	**28.03**	**151 s**
instance_01_10pol	Best	**192.64**	**298 s**	192.83	260 s	192.79	129 s
	Average	**193.07**	**300 s**	193.48	269 s	193.16	138 s
instance_01_16pol	Best	182.66	300 s	181.72	300 s	**173.45**	**300 s**
	Average	183.25	300 s	182.56	300 s	**174.99**	**300 s**
instance_01_2pol	Best	**36.04**	**49 s**	**36.04**	**52 s**	**36.04**	**23 s**
	Average	**36.04**	**50 s**	**36.04**	**50 s**	**36.04**	**23 s**
instance_01_3pol	Best	**48.09**	**68 s**	**48.09**	**71 s**	**48.09**	**32 s**
	Average	**48.09**	**69 s**	**48.09**	**71 s**	**48.09**	**33 s**
instance_01_4pol	Best	**72.13**	**104 s**	**72.13**	**96 s**	**72.13**	**46 s**
	Average	**72.13**	**105 s**	**72.13**	**96 s**	**72.13**	**47 s**
instance_01_5pol	Best	**90.13**	**135 s**	**90.13**	**122 s**	**90.13**	**53 s**
	Average	**90.15**	**119 s**	90.18	118 s	90.18	56 s
instance_01_6pol	Best	114.37	158 s	**114.20**	**156 s**	114.37	72 s
	Average	114.47	146 s	**114.37**	**140 s**	114.47	69 s
instance_01_7pol	Best	**138.39**	**184 s**	**138.39**	**186 s**	**138.39**	**82 s**
	Average	**138,55**	**172 s**	**138.55**	**179 s**	138.60	95 s
instance_01_8pol	Best	**156.49**	**218 s**	156.54	224 s	**156.49**	**121 s**
	Average	156.69	223 s	**156.66**	**244 s**	**156.66**	**137 s**
instance_01_9pol	Best	**186.41**	**273 s**	186.66	248 s	186.78	118 s
	Average	**186.88**	**250 s**	187.16	236 s	187.18	103 s
instance_artificial_01_26pol_hole	Best	218.87	300 s	219	300 s	**218.27**	**300 s**
	Average	219.71	300 s	219.55	300 s	**219.12**	**300 s**
rco1	Best	162.96	276 s	**162.63**	**221 s**	162.71	111 s
	Average	163.33	232 s	**163.12**	**249 s**	163.21	113 s
rco2	Best	321.37	300 s	**317.31**	**300 s**	317.60	273 s
	Average	322.51	300 s	**318.73**	**300 s**	318.79	265 s
rco3	Best	507.25	300 s	492.88	300 s	**480.36**	**300 s**
	Average	509.64	300 s	500.92	300 s	**482.03**	**300 s**
shapes2	Best	228.83	300 s	**227.79**	**300 s**	227.93	243 s
	Average	229.98	300 s	**228.18**	**300 s**	228.57	267 s
shapes4	Best	478.50	300 s	475.20	300 s	**454.27**	**300 s**
	Average	479.27	300 s	477.46	300 s	**455.44**	**300 s**
spfc_instance	Best	**147.33**	**300 s**	147.38	300 s	148.06	180 s
	Average	148.10	300 s	**148.09**	**300 s**	148.36	202 s
trousers	Best	347.41	300 s	346.45	300 s	**345.94**	**300 s**
	Average	348.69	300 s	347.65	300 s	**346.61**	**300 s**

Appendix D

Appendix D.1

Table A5. BRKGA X LaserCut Software.

Instances	BRKGA		LASERCUT		GAIN	
	(C)	(S)	(C)	(S)	(C)	(S)
albano	111.41	115.86	118.66	118.71	**+ftg7.25**	**+2.85**
blaz1	154.71	162.80	177.51	178.05	**+22.8**	**+15.25**
blaz2	270.91	292.68	357.40	323.54	**+86.49**	**+30.86**
blaz3	428.21	496.80	537.55	538.97	**+109.34**	**+42.17**
dighe1	70.69	96.30	113.83	114.51	**+43.14**	**+18.21**
dighe2	54	79.07	88.22	88.58	**+34.22**	**+9.51**
fu	23.91	28.03	35.05	35.81	**+11.14**	**+7.78**
inst_01_10pol	127.22	193.16	200.13	200.59	**+72.91**	**+7.43**
inst_01_16pol	76.88	174.99	124.91	185.11	**+48.03**	**+10.12**
inst_01_2pol	33.11	36.04	37	37.15	**+3.89**	**+1.11**
inst_01_3pol	39.24	48.09	49	50.04	**+9.76**	**+1.95**
inst_01_4pol	57.36	72.13	74.57	74.64	**+24.37**	**+2.51**
inst_01_5pol	69.61	90.18	93.98	94.61	**+24.37**	**+4.43**
inst_01_6pol	81.73	114.47	118.35	119.51	**+36.62**	**+5.04**
inst_01_7pol	96.98	138.60	143.74	144.06	**+46.76**	**+5.46**
inst_01_8pol	105.85	156.66	162.04	162.48	**+56.19**	**+5.82**
inst_01_9pol	124.10	187.18	193.30	193.85	**+69.20**	**+6.67**
inst_01_26pol_hole	165.78	219.12	213.88	213.03	**+48.10**	**−6.09**
rco1	140.73	163.21	174.21	174.61	**+33.48**	**+11.40**
rco2	271.63	318.79	349.94	352.05	**+78.31**	**+33.26**
rco3	398.33	482.03	526.18	526.90	**+127.85**	**+44.87**
shapes2	216.40	228.57	245.12	246.72	**+28.70**	**+18.15**
shapes4	432.10	455.44	485.45	486.69	**+53.35**	**+31.25**
spfc_instance	144.16	148.36	175.76	165.90	**+31.60**	**+17.54**
trousers	308.75	346.61	302.59	303.54	**−6.16**	**−43.07**

References

1. Júnior, B.A.; Pinheiro, P.R. Approaches to tackle the nesting problems. In *Artificial Intelligence Perspectives in Intelligent Systems*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 285–295.
2. Pisinger, D. Heuristics for the container loading problem. *Eur. J. Oper. Res.* **2002**, *141*, 382–392. [CrossRef]
3. Araújo, L.J.; Panesar, A.; Özcan, E.; Atkin, J.; Baumers, M.; Ashcroft, I. An experimental analysis of deepest bottom-left-fill packing methods for additive manufacturing. *Int. J. Prod. Res.* **2020**, *58*, 6917–6933. [CrossRef]
4. Araújo, L.J.; Özcan, E.; Atkin, J.A.; Baumers, M. Analysis of irregular three-dimensional packing problems in additive manufacturing: A new taxonomy and dataset. *Int. J. Prod. Res.* **2019**, *57*, 5920–5934. [CrossRef]
5. Moreira, L.M.; Oliveira, J.F.; Gomes, A.M.; Ferreira, J.S. Heuristics for a dynamic rural postman problem. *Comput. Oper. Res.* **2007**, *34*, 3281–3294. [CrossRef]
6. Manber, U.; Israni, S. Pierce point minimization and optimal torch path determination in flame cutting. *J. Manuf. Syst.* **1984**, *3*, 81–89. [CrossRef]
7. Wäscher, G.; Haußner, H.; Schumann, H. An improved typology of cutting and packing problems. *Eur. J. Oper. Res.* **2007**, *183*, 1109–1130. [CrossRef]
8. Araújo, L.J.; Pinheiro, P.R. Applying backtracking heuristics for constrained two-dimensional guillotine cutting problems. In Proceedings of the International Conference on Information Computing and Applications, Ho Chi Minh City, Vietnam, 5–7 December 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 113–120.
9. Hopper, E.; Turton, B.C. A review of the application of meta-heuristic algorithms to 2D strip packing problems. *Artif. Intell. Rev.* **2001**, *16*, 257–300. [CrossRef]
10. Raggenbass, A.; Reissner, J. Automatic Generation of NC Production Plans in Stamping and Laser Cutting. *CIRP Ann.* **1991**, *40*, 247–250. [CrossRef]
11. Raggenbass, A.; Reissner, J. Stamping—Laser Combination in Sheet Processing. *CIRP Ann.* **1989**, *38*, 291–294. [CrossRef]

12. Amaro Júnior, B.; Pinheiro, P.R.; Coelho, P.V. A Parallel Biased Random-Key Genetic Algorithm with Multiple Populations Applied to Irregular Strip Packing Problems. Available online: https://www.hindawi.com/journals/mpe/2017/1670709/ (accessed on 8 October 2021).
13. Leao, A.A.; Toledo, F.M.; Oliveira, J.F.; Carravilla, M.A.; Alvarez-Valdés, R. Irregular packing problems: A review of mathematical models. *Eur. J. Oper. Res.* **2020**, *282*, 803–822. [CrossRef]
14. Labrada-Nueva, Y.; Cruz-Rosales, M.H.; Rendón-Mancha, J.M.; Rivera-López, R.; Eraña-Díaz, M.L.; Cruz-Chávez, M.A. Overlap Detection in 2D Amorphous Shapes for Paper Optimization in Digital Printing Presses. *Mathematics* **2021**, *9*, 1033. [CrossRef]
15. Silva, E.F.; Oliveira, L.T.; Oliveira, J.F.; Toledo, F.M.B. Exact approaches for the cutting path determination problem. *Comput. Oper. Res.* **2019**, *112*, 104772. [CrossRef]
16. Lee, M.K.; Kwon, K.B. Cutting path optimization in CNC cutting processes using a two-step genetic algorithm. *Int. J. Prod. Res.* **2006**, *44*, 5307–5326. [CrossRef]
17. Dewil, R.; Vansteenwegen, P.; Cattrysse, D. Construction heuristics for generating tool paths for laser cutters. *Int. J. Prod. Res.* **2014**, *52*, 5965–5984. [CrossRef]
18. Dewil, R.; Vansteenwegen, P.; Cattrysse, D.; Laguna, M.; Vossen, T. An improvement heuristic framework for the laser cutting tool path problem. *Int. J. Prod. Res.* **2015**, *53*, 1761–1776. [CrossRef]
19. Zhao, X.; Bennell, J.A.; Bektaş, T.; Dowsland, K. A comparative review of 3D container loading algorithms. *Int. Trans. Oper. Res.* **2016**, *23*, 287–320. [CrossRef]
20. Hoeft, J.; Palekar, U.S. Heuristics for the plate-cutting traveling salesman problem. *IIE Trans.* **1997**, *29*, 719–731. [CrossRef]
21. Arkin, E.M.; Hassin, R. Approximation algorithms for the geometric covering salesman problem. *Discret. Appl. Math.* **1994**, *55*, 197–218. [CrossRef]
22. Veeramani, D.; Kumar, S. Optimization of the nibbling operation on an NC turret punch press. *Int. J. Prod. Res.* **1998**, *36*, 1901–1916. [CrossRef]
23. Garfinkel, R.S.; Webb, I.R. On crossings, the Crossing Postman Problem, and the Rural Postman Problem. *Networks* **1999**, *34*, 173–180.:3<173::AID-NET1>3.0.CO;2-W. [CrossRef]
24. Rodrigues, A.; Soeiro Ferreira, J. Cutting path as a Rural Postman Problem: Solutions by Memetic Algorithms. *IJCOPI* **2012**, *3*, 31–46.
25. Dewil, R.; Vansteenwegen, P.; Cattrysse, D. A review of cutting path algorithms for laser cutters. *Int. J. Adv. Manuf. Technol.* **2016**, *87*, 1865–1884. [CrossRef]
26. Chvátal, V.; Cook, W.; Dantzig, G.B.; Fulkerson, D.R.; Johnson, S.M. Solution of a large-scale traveling-salesman problem. In *50 Years of Integer Programming 1958–2008*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 7–28.
27. Ahadi, A.; Mozafari, A.; Zarei, A. Touring a sequence of disjoint polygons: Complexity and extension. *Theor. Comput. Sci.* **2014**, *556*, 45–54. [CrossRef]
28. Khan, W.; Hayhurst, D. Two and Three-Dimensional Path Optimization for Production Machinery. *J. Manuf. Sci. Eng. Trans. Asme J. Manuf. Sci. Eng.* **2000**, *122*. [CrossRef]
29. Erdos, G.; Kemény, Z.; Kovacs, A.; Váncza, J. Planning of Remote Laser Welding Processes. *Procedia CIRP* **2013**, *7*, 222–227. [CrossRef]
30. Xie, S.; Tu, Y.; Liu, J.; Zhou, Z. Integrated and concurrent approach for compound sheet metal cutting and punching. *Int. J. Prod. Res.* **2010**, *39*, 1095–1112. [CrossRef]
31. Yu, W.; Lu, L. A route planning strategy for the automatic garment cutter based on genetic algorithm. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014, Beijing, China, 6–11 July 2014; pp. 379–386. [CrossRef]
32. Bobade, S.; Badgujar, T.Y.B. A State Of Art In A Sheet Metal Stamping Forming Technology-An Overview. *Int. J. Adv. Res. Innov. Ideas Educ.* **2017**, 3760–3770.
33. Han, G.; Na, S. A study on torch path planning in laser cutting processes part 2: Cutting path optimization using simulated annealing. *J. Manuf. Syst.* **1999**, *18*, 62–70. [CrossRef]
34. Dewil, R.; Vansteenwegen, P.; Cattrysse, D. Cutting path optimization using tabu search. In *Key Engineering Materials*; Trans Tech Publ: Zurich, Switzerland 2011; Volume 473, pp. 739–748.
35. Golden, B.L.; Wong, R.T. Capacitated arc routing problems. *Networks* **1981**, *11*, 305–315. [CrossRef]
36. Usberti, F.L.; França, P.M.; França, A.L.M. The open capacitated arc routing problem. *Comput. Oper. Res.* **2011**, *38*, 1543–1555. [CrossRef]
37. Holland, J. *Adaptation In Natural And Artificial Systems*; University of Michigan Press: Ann Arbor, MI, USA, 1975.
38. Hopper, E.; Turton, B. A genetic algorithm for a 2D industrial packing problem. *Comput. Ind. Eng.* **1999**, *37*, 375–378. [CrossRef]
39. Onwubolu, G.; Mutingi, M. A genetic algorithm approach for the cutting stock problem. *J. Intell. Manuf.* **2003**, *14*, 209–218.:1022955531018. [CrossRef]
40. Park, J.Y.; Seo, J.J. A Study on Cutting Path Optimization Using Genetic Algorithm. *J. Ocean Eng. Technol.* **2009**, *23*, 67–70.
41. Deshpande, A.S.; Kelkar, R.B. Advanced genetic operators and techniques: An analysis of dominance & diploidy, reordering operator in genetic search. In Proceedings of the Ninth WSEAS International Conference on Evolutionary Computing, Sofia, Bulgaria, 2–4 May 2008; pp. 27–33.
42. Blickle, T.; Thiele, L. A Mathematical Analysis of Tournament Selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1995; pp. 9–16.

43. Goldberg, D.E.; Lingle, R. AllelesLociand the Traveling Salesman Problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*; L. Erlbaum Associates Inc.: Mahwah, NJ, USA, 1985; pp. 154–159.
44. Bean, J.C. Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA J. Comput.* **1994**, *6*, 154–160. [CrossRef]
45. Resende, M. Biased random-key genetic algorithms with applications in telecommunications. *TOP* **2010**, *20*, 130–153. [CrossRef]
46. Gonçalves, J.F.; Resende, M.G.; Toso, R.F. An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesqui. Oper.* **2014**, *34*, 143–164. [CrossRef]
47. Spears, V.M.; Jong, K.A.D. On the virtues of parameterized uniform crossover. In Proceedings of the Fourth International Conference on Genetic Algorithms, San Diego, CA, USA, 13–16 July 1991; pp. 230–236.
48. Baioletti, M.; Milani, A.; Santucci, V. Variable neighborhood algebraic differential evolution: An application to the linear ordering problem with cumulative costs. *Inf. Sci.* **2020**, *507*, 37–52. [CrossRef]
49. Santucci, V.; Baioletti, M.; Milani, A. An algebraic framework for swarm and evolutionary algorithms in combinatorial optimization. *Swarm Evol. Comput.* **2020**, *55*, 100673. [CrossRef]
50. Baioletti, M.; Milani, A.; Santucci, V. An algebraic approach for the search space of permutations with repetition. In Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar), Seville, Spain, 15–17 April 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 18–34.