A Simple and Effective Hybrid Genetic Search for the Job Sequencing and Tool Switching Problem

Jordana Mecler
Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
jmecler@inf.puc-rio.br

Anand Subramanian
Universidade Federal da Paraíba, Departamento de Sistemas de Computação
anand@ci.ufpb.br

Thibaut Vidal
Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)
vidalt@inf.puc-rio.br

October 2019
Technical Report – PUC-Rio

Abstract. The job sequencing and tool switching problem (SSP) has been extensively studied in the field of operations research, due to its practical relevance and methodological interest. Given a machine that can load a limited amount of tools simultaneously and a number of jobs that require a subset of the available tools, the SSP seeks a job sequence that minimizes the number of tool switches in the machine. To solve this problem, we propose a simple and efficient hybrid genetic search based on a generic solution representation, a tailored decoding operator, efficient local searches and diversity management techniques. To guide the search, we introduce a secondary objective designed to break ties. These techniques allow to explore structurally different solutions and escape local optima. As shown in our computational experiments on classical benchmark instances, our algorithm significantly outperforms all previous approaches while remaining simple to apprehend and easy to implement. We finally report results on a new set of larger instances to stimulate future research and comparative analyses.

Keywords. Combinatorial optimization; Job sequencing; Tool switches; Metaheuristics; Hybrid genetic search

1 Introduction

The *job sequencing and tool switching problem* (SSP) considers a set of jobs $J = \{1, ..., n\}$, a set of tools $T = \{1, ..., m\}$, and a single machine whose magazine can hold up to C tools simultaneously. Each job $j \in J$ requires a subset T_j of the available tools to be performed, where $|T_j| \leq C$. We assume that the tool setup times are identical and that every tool fits into one slot of the machine. In most applications, the total number of tools needed by all the jobs exceeds the machine magazine capacity. Therefore, the machine must switch some tools when finishing a job and starting another one. The goal of the SSP is to find a job sequence and a tool-loading strategy that minimizes the total number of switches.

The SSP arises in many applications, including circuit-board manufacturing [41], computer memory management [26], and pharmaceutical packaging [38]. In the first application, the jobs represent circuits which need to be placed on a board and the tools are the surface-mount component (SMC) feeders. In the second application, the jobs represent tasks which must be processed and the tools are the pages (memory fragments) that need to be transferred from the slow memory to the fast memory. In the last application, the jobs represent patient medicine boxes, and the tools are the pills.

The SSP can be generally decomposed into a *sequencing* problem which aims to find the best job sequence, and a *tooling problem* which seeks the best tool loading policy for a fixed sequence. For a fixed job sequence, the tooling problem can be solved in polynomial time in $\mathcal{O}(nm)$ with the "Keep Tools Needed Soonest" (KTNS) policy [47]. In contrast, for a free job sequence, the SSP is known to be $\mathcal{NP}$-hard [21].

Due to its rich structure and practical interest, the SSP has been the focus of extensive research, and most of the classical metaheuristic frameworks (e.g., tabu search, iterated local search, and genetic algorithms) have been adapted to this problem. However, the discrete nature of the objective (number of tool switches) and the problem symmetries tend to diminish the effectiveness of local searches, and most of the best methods tend to be complex and over-engineered. As a consequence, notable methodological breakthroughs remain achievable, even for medium-scale problems with a few dozens of jobs. Finally, the problem decision sets lend themselves to an effective application of solution representative- and decoder-based algorithms, consisting in searching the space of the permutations and systematically applying a decoder (the KTNS policy) to evaluate solution costs. Such a decision-set problem decomposition has been instrumental in designing efficient metaheuristics for a variety of permutation and set based problems [see, e.g., 30, 49, 51, 52, 55].

In this article, we present new methodological advances for the SSP and pursue the study of decision set decomposition-based metaheuristics. Inspired by the success of hybrid genetic searches (HGS) on vehicle routing problems —i.e., genetic algorithms with local search and population diversity management [54, 55]— we use and adapt this methodological paradigm with search operators tailored for the SSP. To perform a controlled exploration of solutions with identical cost, we also exploit a secondary objective which favors small 0-blocks to guide the search towards solution improvements. The main contributions of this article are fourfold:

- We introduce an efficient HGS for the SSP. This method exploits a simple permutation-based solution representation and measures solution quality in terms of cost and contribution to the population diversity during parents and survivors selections.
- We introduce a secondary objective which promotes short 0-blocks as a means to progress towards solutions with fewer switches.
- We conduct extensive computational experiments on classical SSP instances to evaluate the performance of our approach and measure the contribution of its principal components. These experiments show that the proposed method performs remarkably well in relation to previous approaches and that its main strategies (secondary objective and diversity management) are critical for a good performance.
- We finally report additional results on a set of larger-scale instances to stimulate future research and comparative analyses.

The remainder of the paper is organized as follows. Section 2 reviews the literature, while Section 3 describes the proposed algorithm. Section 4 presents our computational analyses, and Section 5 concludes.

2 Related Studies

The SSP was formally proposed in the late 1980s by Tang and Denardo [47]. Along with the problem, the authors introduced the KTNS policy, which determines the optimal number of tool switches in polynomial time given a predefined job sequence. The KTNS policy is a fundamental example of a greedy algorithm, which is also commonly used for interval scheduling, coloring and caching problems [13, 16]. Later, Gray et al. [29] discussed decision models for tooling management, including the SSP, while Crama et al. [21] proved that the job sequencing part of the problem is $\mathcal{NP}$-hard. A model using a network flow approach for the nonuniform case was proposed by Privault and Finke [41]. Crama [19] discussed optimization models and solutions for production planning and scheduling problems such as the SSP. An empirical study of the SSP in manufacturing companies was conducted by Shirazi and Frizelle [46], where the authors concluded that the heuristics available at the time outperformed the methods used within the companies. Reflecting the variety of application cases, several other studies have considered variant of the SSP, e.g., minimizing the tool switching instants [1, 34, 48], considering multiple objectives [33, 44], parallel machines [11, 23, 58], different tool sizes [37, 50], and other features [9, 24, 28, 32, 42, 44, 50].

Some mathematical programming algorithms have been proposed for the SSP. Tang and Denardo [47] formulated the problem as an integer program (IP). Later on, Laporte et al. [36], Catanzaro et al. [17] and Ghiani et al. [26] proposed branch-and-cut and branch-and-bound methods based on integer linear programs (ILP). On the other hand, Bard [10] developed a nonlinear IP and solved it using a dual-based relaxation heuristic, whereas Ghiani et al. [27] modeled the problem as a nonlinear least-cost Hamiltonian cycle problem and developed a branch-and-cut algorithm. Yanasse et al. [57] proposed an enumeration method based on a partial ordering of jobs. Finally, Crama and van de Klundert [20] studied the worst-case performance of approximation algorithms for various tool management problems, including the SSP.

Other SSP studies have been principally focused on heuristics and metaheuristics. Tang and Denardo [47] presented a job scheduling heuristic to find a short Hamiltonian path on a graph, where the nodes represent jobs and the edge weights represent the minimum number of tool switches obtained by processing two jobs consecutively. Salonen et al. [44] considered both job-grouping and tool-switch objectives but also reported heuristic results for the classical SSP. Burger et al. [14] developed job-grouping heuristics for the color print scheduling problem, which is an application of the SSP. Crama et al. [21] proposed heuristics based on the traveling salesman problem (TSP) for a graph representation similar to that of Tang and Denardo [47]. Hertz et al. [31] improved on earlier heuristics using methods such as GENIUS [25]. Privault and Finke [41] reduced the tool-switching part of the general nonuniform problem case into a search for a minimum cost flow of maximum value over an acyclic network, and they proposed a set of heuristics for the job sequencing. Djellab et al. [22] formulated the problem using a hypergraph representation and implemented a constructive method. Salonen et al. [43] presented a multi-start algorithm, creating the starting points by grouping jobs with similar tools.

A variety of local search-based metaheuristics have been applied to the problem, including tabu search [3, 35], iterated local search [40], and beam search [45, 59]. Population-based methods have also been successful. Hybrid methods combining genetic algorithms (GA) with other search procedures can be found in [2, 4, 5, 6, 7, 8, 18]. Amaya et al. [4, 6, 7] combined GA with local search-based procedures such as hill climbing, simulated annealing and tabu search, leading to hybrid methods which are also known under the name of memetic algorithms. Amaya et al. [5, 6, 8] combined genetic algorithms with a multi-agent approach or cross-entropy methods. Finally, Chaves et al. [18] combined clustering search (CS) with a biased random-key genetic algorithm (BRKGA), while Ahmadi et al. [2] combined a 2-TSP scheme with a novel learning-based GA. Table 1 summarizes the SSP studies in chronological order, lists their main contributions as well as the origin of the benchmark instances considered.

Most of the aforementioned GA-based implementations use traditional parent-selection strategies [56] such as roulette wheel [2] and binary tournament [4, 5, 7]. Survivor selections are solely based on individual objective values, typically obtained with KTNS [2, 4, 5, 6, 7, 18]. Ahmadi et al. [2] also take into account a similarity function for chromosomes to remove individuals from the population. The crossover operators applied are parameterized uniform crossover [18], uniform cycle crossover [7], alternating position (APX) [4, 5] and partially mapped (PMX) [2]. Finally, although some authors have developed local search procedures based on

Table 1: Summary of SSP studies

Work	**Year**	**Approach**	**Instances**
Tang and Denardo [47]	1988	Exact methods + heuristics	[47]
Bard [10]	1988	Exact methods + heuristics	[10]
Crama et al. [21]	1994	Heuristics	[21]
Privault and Finke [41]	1995	Network flow formulation + heuristics	[41]
Hertz et al. [31]	1998	Heuristics	[31]
Crama and van de Klundert [20]	1999	Worst-case analysis	–
Djellab et al. [22]	2000	Heuristics	[21]
Shirazi and Frizelle [46]	2001	Empirical study	Real life
Al-Fawzan and Al-Sultan [3]	2003	Tabu search	[3]
Laporte et al. [36]	2004	Exact methods	[31, 36]
Zhou et al. [59]	2005	Beam search	[10]
Salonen et al. [43]	2006	Heuristics	[21]
Salonen et al. [44]	2006	Exact methods + heuristics	[44]
Ghiani et al. [26]	2007	Exact methods	[31, 36]
Amaya et al. [4]	2008	Memetic algorithms	[4]
Konak et al. [35]	2008	Tabu search	[35]
Senne and Yanasse [45]	2009	Beam search	[45]
Yanasse et al. [57]	2009	Exact methods	[57]
Ghiani et al. [27]	2010	Exact methods	[27]
Amaya et al. [8]	2010	Hybrid cooperative	[8]
Amaya et al. [5]	2011	Memetic cooperative	[5]
Amaya et al. [7]	2012	Memetic algorithms	[7]
Amaya et al. [6]	2013	Cross entropy-based memetic algorithms	[6]
Burger et al. [14]	2015	Heuristics	[14] & Real life
Catanzaro et al. [17]	2015	Exact methods	[17]
Chaves et al. [18]	2016	Hybrid metaheuristics	[21, 57]
Paiva and Carvalho [40]	2017	Iterated local search	[21, 17, 57]
Ahmadi et al. [2]	2018	Hybrid metaheuristics	[21, 17]

problem-specific metrics such as 1-blocks and 0-blocks [21], ties arising during move evaluations are generally handled arbitrarily [5, 7].

We refer the reader to Calmels [15] for an extensive literature review and classification of SSP variants. To date, the best metaheuristic algorithms are those of Paiva and Carvalho [40], Chaves et al. [18] and Ahmadi et al. [2]. During our experimental comparisons, we will therefore compare our results with the results reported by these authors.

3 Proposed Methodology

To effectively solve the SSP, we exploit the hybrid generic search (HGS) paradigm [54, 55] outlined in Algorithm 1. Our algorithm starts with a population of size μ containing random solutions (permutations) improved by local search (line 1). Then, iteratively, it selects two parents by *binary tournament* (line 3) and crosses them via an order crossover (OX) [39] to produce a single offspring (line 4), which is improved by local search (line 5). The generation scheme is pursued until the population reaches a maximum size of $|\mathcal{P}| = \mu + \lambda$ individuals. At this point, a survivor selection procedure is applied to discard λ individuals. To that extent, the method considers not only the objective value of each individual, but also its contribution to the diversity of the population. The method terminates as soon as I_{MAX} consecutive iterations (individual generations) have been made without improvement of the best solution.

Algorithm 1 HGS with diversity management for the SSP

```
1: Generate an initial population $\mathcal{P}$ with $\mu$ random individuals subject to local search
2: while termination criterion is not attained do
3:    Select two parents $S_1$ and $S_2$ by binary tournament
4:    Generate a single child $S$ by order crossover (OX) on $S_1$ and $S_2$
5:    Apply local search on $S$
6:    Insert the resulting individual into the population $\mathcal{P}$
7:    if $|\mathcal{P}| = \mu + \lambda$ then Use a survivors selection procedure to discard $\lambda$ individuals, taking
      into account their quality and contribution to the population diversity
8: end while
9: return best solution found
```

As visible in Algorithm 1, the overall solution approach follows a simple scheme similar to [54, 55]. Nevertheless, each component and operator (solution evaluation, crossover, local search, diversity management) has been tailored to the SSP and plays a major role in the success of the method. These components will be detailed in the following sections.

3.1 Solution evaluation

Each solution (i.e., individual) in the population and local search is represented as a simple permutation of jobs. To evaluate it, we apply the KTNS algorithm [47] as a solution decoder to find an optimal tooling strategy. As illustrated in Table 2 on an example containing $n = 10$

jobs, $m = 10$ tools with a magazine of capacity $C = 4$, the job sequence can be represented as a binary matrix M in which columns are the jobs and rows are the tools. If the j^{th} job in the sequence requires tool t, then $M_{tj} = 1$, otherwise $M_{tj} = 0$. Since the number of tools needed by each job cannot exceed the magazine capacity, each column contains no more than C times the value 1.

Table 2: Required tools matrix – Before KTNS

		Jobs									
		1	2	3	4	5	6	7	8	9	10
Tools	1	0	1	0	0	0	0	0	0	0	0
	2	1	0	0	0	1	1	1	0	0	0
	3	0	1	0	0	1	1	0	0	0	0
	4	0	0	1	1	0	0	1	0	1	0
	5	0	0	0	0	1	1	0	0	0	0
	6	1	0	0	0	0	0	0	1	0	0
	7	0	0	1	0	1	0	0	0	0	1
	8	0	0	0	1	0	0	0	1	0	1
	9	0	1	0	0	0	1	1	1	1	0
	10	0	0	0	0	0	0	0	0	1	0

Evaluation of the primary objective. The KTNS algorithm determines the extra tools which may be maintained in the magazine when performing each job. It iterates over the job sequence from beginning to end and adheres to two simple rules:

(i) at each step, only the tools required for the current job are inserted; and

(ii) whenever new tools are loaded, the other tools kept in the machine are those which are needed soonest.

To efficiently implement this policy, we maintain for each job an auxiliary data structure which keeps track, at each instant, of the next instant in which this job is needed. Table 3 displays the loaded tools matrix obtained with KTNS on the previous example. For each job, the tools which are kept but not required are represented by an underscored 1.

Table 3: Loaded tools matrix – After KTNS

		Jobs									
		1	2	3	4	5	6	7	8	9	10
Tools	1	0	1	0	0	0	0	0	0	0	0
	2	1	$\underline{1}$	$\underline{1}$	0	1	1	1	0	0	0
	3	0	1	$\underline{1}$	$\underline{1}$	1	1	0	0	0	0
	4	0	0	1	1	0	0	1	$\underline{1}$	1	0
	5	0	0	0	0	1	1	$\underline{1}$	0	0	0
	6	1	0	0	0	0	0	0	1	0	0
	7	0	0	1	$\underline{1}$	1	0	0	0	0	1
	8	0	0	0	1	0	0	0	1	$\underline{1}$	1
	9	0	1	0	0	0	1	1	1	1	$\underline{1}$
	10	0	0	0	0	0	0	0	0	1	$\underline{1}$

The number of tool switches can be obtained from Table 3 by computing the number of times a 1 turns into a 0 in every row, which is equivalent to replacing a tool by another in the machine.

Evaluation of the tie-breaking objective. Many solutions explored during the search, within the genetic algorithm and the local search, have the same number of tool switches. It is therefore necessary to guide the algorithm in "plateau" regions of the search space, where all solutions have the same primary objective. To that extent, we define a secondary objective, designed to break ties in favor of solutions which are more likely to lead to future improvements. We use the concept of 0-blocks, first introduced in Crama et al. [21]. A 0-block is a maximum sequence of consecutive zeros, preceded and followed by a one, in the loaded matrix representation. It represents an interval during which a tool is not needed, but which could be filled by maintaining the tool in the magazine. In Table 3, for example, the jobs in positions 5 to 7 represent a 0-block of size 3 for tool 8. Intuitively, short 0-blocks are more likely to be filled during the search process, leading to an effective reduction of the number of tool switches and to an improvement of the primary objective. For a solution S, and for each tool j, let $k_j(S)$ be the number of 0-blocks in the loaded tool matrix and let $b_j^x(S)$ be the size of the x^{th} 0-block, for $x \in \{1, \dots, k_j(S)\}$. The tie-breaking objective is then evaluated as:

$$\Phi'(S) = \sum_{j=1}^{m} \sum_{x=1}^{k_j(S)} \sqrt{b_j^x(S)}. \tag{1}$$

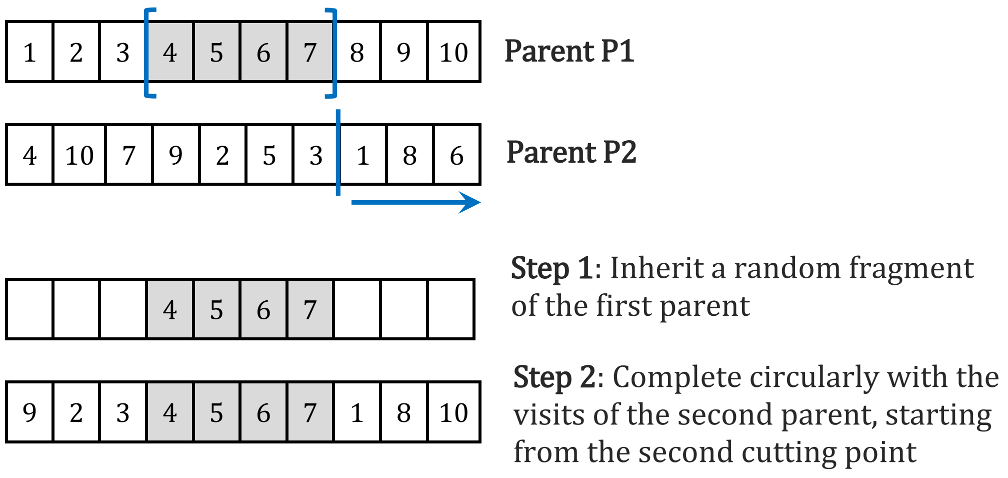
We use the square root function due to its concavity. By minimizing $\Phi'(S)$, we favor solutions with short and large 0-blocks (e.g., one block of size 2 and another of size 8) over solutions with balanced blocks (e.g., two blocks of size 5), with the aim of ultimately eliminating some of the shortest ones and reducing the number of tool switches.

3.2 Generation of new solutions

Firstly, two parents are selected by binary tournament. Each binary tournament selection consists in pickup randomly (with uniform probability) two individuals in the population and retaining the one with the best biased fitness, as defined in Section 3.3.

Secondly, the order crossover (OX) [39] is applied on the two parents to generate a single child. This crossover, illustrated in Figure 1, consists in (i) selecting a random substring from the first parent ; (ii) copying this substring into the child while leaving the rest of the positions empty; and (iii) sweeping through the second parent to fill the empty positions with the missing visits.

Finally, the resulting offspring is improved by means of three successive local searches, based on the 2-OPT, RELOCATE and SWAP neighborhoods, in this specific order. As discussed in Section 4.3, we performed sensitivity analysis for each alternative neighborhood ordering, and this setting led to solutions of significantly higher quality. In each local search, the neighborhood is explored in random order according to a first-improvement policy, i.e., any improving move (improving



1 2 3 4 5 6 7 8 9 10
Parent P1
4 10 7 9 2 5 3 1 8 6
Parent P2
4 5 6 7
Step 1: Inherit a random fragment of the first parent
9 2 3 4 5 6 7 1 8 10
Step 2: Complete circularly with the visits of the second parent, starting from the second cutting point

Figure 1: Illustration of the OX crossover on a small example with 10 jobs

the number of tool switches, or improving the tie-breaking objective for an equal number of tool switches) is immediately applied. Each local search stops as soon as all moves have been successively tested without improvement of the primary or auxiliary objective, therefore reaching a local minimum. The resulting solution is added into the population.

3.3 Biased fitness and population diversity management

The biased fitness $f_{\mathcal{P}}(p)$ of each individual p in the population $\mathcal{P}$ is calculated in a similar way as in [54, 55]. The individuals are kept ordered in terms of quality (considering primary and tie-breaking objectives) and diversity contribution. To calculate the diversity contribution of each individual, the algorithm computes its average distance to its μ^{close} closest ones, the distance between two individuals (represented as job permutations) being defined as the number of different edges between them (broken-pairs distance). Then, it sorts the population in decreasing order of diversity contribution, and in increasing order of objective function, associating to each individual a diversity rank $f^{div}_{\mathcal{P}}(p)$ and a quality rank $f^{obj}_{\mathcal{P}}(p)$. Finally, the biased fitness of each individual is calculated as:

$$f_{\mathcal{P}}(p) = f^{obj}_{\mathcal{P}}(p) + \left(1 - \frac{\mu^{elite}}{|\,\mathcal{P}\,|}\right) \times f^{div}_{\mathcal{P}}(p) \tag{2}$$

Small values of biased fitness correspond to promising individuals, with a small objective value and a large contribution to the population diversity.

The biased fitness measure is used when selecting parents by binary tournament (retaining the individual with the smallest biased fitness) and when selecting individuals to exclude from the population during survivor selections. In the latter case, the algorithm iteratively excludes the worst individual (i.e., with highest biased fitness) having a clone whenever duplicated solutions exist in the population, or the worst individual otherwise. This process is repeated until the desired population size of μ is attained. As discussed in [54], this survivor selection procedure preserves diversity and meanwhile guarantees that the best μ^{elite} individuals in the population remain preserved.

4 Computational Experiments

We implemented the proposed method in C++ and conducted our experiments on a single thread of an Intel Core i5-4288U 2.6 GHz processor with 8 GB of RAM running macOS High Sierra 10.13.6. It was compiled with g++ 8.1 using the -O3 flag. The source code and benchmark instances are also accessible at `https://github.com/jordanamecler/HGS-SSP`. [Password protected until publication].

We first describe the characteristics of the instances used in our experiments. Next, we explain how we calibrated the parameters of the algorithm and investigate the impact of several methodological choices. Finally, we compare our algorithm with the current state-of-the-art methods and provide solutions for a new set of larger instances.

4.1 Benchmark instances

As indicated in Table 4, we use the classical benchmark instances of Crama et al. [21], Yanasse et al. [57], and Catanzaro et al. [17] to evaluate the performance of the proposed method. These instances are organized in different groups corresponding to different size parameters. Groups A, B, C, D, and E (from [57]) contain a total of 1350 instances with 8 to 25 jobs. Groups C_1, C_2, C_3, and C_4 (from [21]) as well as $datA$, $datB$, $datC$, and $datD$ (from [17]) contain 40 instances each and include 10 to 40 jobs.

Table 4: SSP instances

Group	**#Jobs**		**#Tools**		**Capacity**		**#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
$datA$	10	10	10	10	4	7	40
$datB$	15	15	20	20	6	12	40
$datC$	30	30	40	40	15	25	40
$datD$	40	40	60	60	20	30	40

4.2 Parameters Calibration

We first conducted preliminary experiments on a subset of instances to find suitable parameter values for our algorithm. We chose to perform the calibration tests on the benchmark of Crama et al. [21] since it contains instances with very diverse characteristics. This led to our *baseline configuration* with the following parameter values: $\mu = 20$, $\lambda = 40$, $\mu^{elite} = 10$, and $\mu^{close} = 3$. Subsequently, we performed additional analyses to investigate the impact of any deviation from this parameter setting. We modify the value of each parameter using a one-factor-at-a-time (OFAT) approach and report the results obtained by each configuration.

Tables 5, 6, and 7 show the average results of different method configurations for the benchmark instances of Crama et al. [21]. In these tables, **Avg** is the average objective value, **T** is the average CPU time in seconds, and **Gap (%)** is the percentage gap, calculated as $100 \times \left(\frac{\text{Avg}-\text{BKS}}{\text{BKS}}\right)$, where BKS is the best known solution collected from all previous studies. As visible in these experiments, modifying μ^{close} has only a minor impact on the solution quality and CPU time. On the other hand, as μ^{elite} increases, the CPU time tends to increase, and the best results are obtained when it is set to half of the base population size. The CPU time increases with the population size, but a mid-sized population yielded the best results in terms of solution quality.

Table 5: Varying population size

μ	λ	μ^{elite}	μ^{close}	Avg	T
10	10	5	2	53.07	105.35
10	20	5	2	53.04	112.70
20	20	10	3	53.01	127.12
20	40	10	3	**53.00**	133.09
40	40	20	5	53.00	152.58
40	80	20	5	53.02	163.27
80	80	40	10	53.04	181.49
80	160	40	10	53.06	190.17

Table 6: Varying μ^{elite}

μ	λ	μ^{elite}	μ^{close}	Avg	T
20	40	2	3	53.03	160.65
20	40	5	3	53.02	144.80
20	40	8	3	53.01	139.42
20	40	10	3	**53.00**	133.09
20	40	12	3	53.00	129.29
20	40	15	3	53.02	124.60
20	40	20	3	53.03	114.59

Table 7: Varying μ^{close}

μ	λ	μ^{elite}	μ^{close}	Avg	T
20	40	10	1	53.02	132.28
20	40	10	2	53.00	133.16
20	40	10	3	**53.00**	133.09
20	40	10	5	53.00	138.38
20	40	10	7	53.00	138.33
20	40	10	10	**53.00**	138.22

4.3 Sensitivity analysis

We performed a sensitivity analysis for the main components of the algorithm. The following modifications of the method were tested: changing the order of the neighborhoods, deactivating some or several neighborhoods, deactivating the diversity management component, and deactivating the secondary objective. The results of these alternative configurations are reported in Table 8 and compared to our baseline configuration.

Table 8: Sensitivity Analysis

Configuration	Avg	Gap(%)	T
Baseline	53.00	**-0.01**	135.31
2OPT-SWAP-RELOCATE	53.01	0.02	145.42
SWAP-2OPT-RELOCATE	53.06	0.09	178.56
SWAP-RELOCATE-2OPT	53.06	0.10	183.41
RELOCATE-2OPT-SWAP	53.06	0.09	160.73
RELOCATE-SWAP-2OPT	53.08	0.12	183.61
without 2OPT	53.61	0.67	130.95
without SWAP	53.02	0.02	118.97
without RELOCATE	53.11	0.14	105.70
without 2OPT and SWAP	53.80	0.88	74.93
without 2OPT and RELOCATE	53.66	0.72	71.03
without RELOCATE and SWAP	53.17	0.20	74.02
without diversity management	53.04	0.18	111.63
without secondary objective	53.16	0.22	87.20

These results confirm our methodological design and baseline parameter choices. Indeed, all these alternative configurations obtained solutions of significantly worse quality for only moderate time gains. The 2OPT neighborhood appears to be the most important for a good performance, followed by the RELOCATE and SWAP neighborhoods. This difference of impact may also explain why the exploration of the neighborhoods by order of importance 2OPT-RELOCATE-SWAP (as done in our baseline configuration) leads to generally better results in shorter time. The use of the secondary objective contributes to attain solutions of significantly higher quality, and the diversity management strategy has a smaller but still beneficial impact on solution quality.

To better visualize the impact of the secondary objective, we conducted another experimental analysis of the distribution of the individuals in the objective space during a typical GA run, on the first instance of group C_3. This analysis is displayed in Figure 2 at three stages of the solution process: after the first survivors selection, half way through the execution, and after the last survivors selection. On this figure, dot sizes are proportional to the number of solutions sharing the same objective value. Visibly, numerous solutions share the same primary (KTNS) objective value, especially at intermediate or late stages of the search. Fortunately, secondary objective values are better spread, therefore allowing to rank solutions and guide the search towards more promising regions by selection pressure.

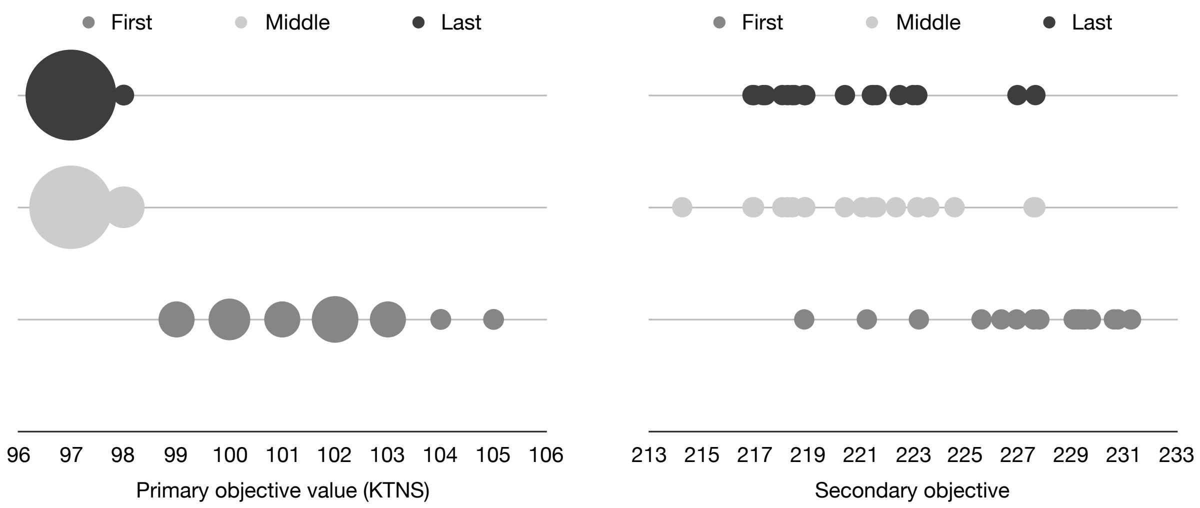


Figure 2: Distribution of objective values in the population, at different stages of the search

4.4 Comparison with other algorithms

The tables presented hereafter report the best solution (**Best**), the average solution over 10 runs (**Avg**), and the average CPU time in seconds (**T**) of each method. The best solutions, for each group of instances, are highlighted in boldface. It is worth mentioning that new random seeds were used in these experiments to eliminate any possibility of overtuning.

Comparison with previous heuristics. We compare our results with those found by the best methods from the literature, namely the CS+BRKGA of Chaves et al. [18], the ILS of Paiva and Carvalho [40] and the DQGA of Ahmadi et al. [2]. The following experimental setup has been used in previous studies, based on the data reported in previous papers and additional information sent to us by the authors:

- Chaves et al. [18]: average of 20 runs on an Intel i7-2600 3.4 GHz with 16 GB of RAM.
- Paiva and Carvalho [40]: average of 20 runs on an Intel i5-3330 3.2 GHz with 8 GB of RAM.
- Ahmadi et al. [2]: average of 10 runs on an Intel i7 3.4 GHz with 16 GB of RAM.

For a fair comparison with these studies, we converted all CPU times into their equivalent time on our i5-4288U 2.6 GHz processor. To that extent, we used the single thread rating from Passmark benchmark (`www.cpubenchmark.net`), leading to conversion factors of 1.0977, 0.9709 and 1.1394 for the machines used in Chaves et al. [18], Paiva and Carvalho [40] and Ahmadi et al. [2], respectively. Moreover, since the precise processor model used in Ahmadi et al. [2] is not documented, we used the average rating of two popular Intel i7 3.4 GHz models of the same generation as a good approximation.

Table 9 presents a summary of the results in which the results are aggregated per instance group. The proposed HGS-SSP found the solutions of best quality for all groups, in a computational time which is equal of smaller than existing algorithms. The difference of solution quality and speed is especially visible on the larger and more challenging instances of groups C_3, C_4, *datC* and *datD*.

Table 9: Summary of results

	CS+BRKGA			ILS			DQGA			HGS-SSP		
Group	Best	Avg	T	Best	Avg	T	Best	Avg	T	Best	Avg	T
A	**12.63**	**12.63**	4.07	**12.63**	**12.63**	0.09	-	-	-	**12.63**	**12.63**	0.06
B	**13.32**	13.32	4.45	**13.32**	13.32	0.15	-	-	-	**13.32**	**13.32**	0.10
C	**17.46**	17.62	10.80	**17.46**	17.46	1.40	-	-	-	**17.46**	17.46	0.91
D	13.40	13.68	30.37	**13.38**	13.39	5.07	-	-	-	**13.38**	13.38	4.28
E	**9.64**	9.70	7.18	**9.64**	**9.64**	0.39	-	-	-	**9.64**	9.64	0.43
C_1	**5.68**	5.68	2.42	**5.68**	**5.68**	0.07	**5.68**	**5.68**	0.26	**5.68**	**5.68**	0.07
C_2	**13.00**	13.07	11.58	**13.00**	13.01	0.83	**13.00**	13.03	1.38	**13.00**	**13.00**	0.78
C_3	60.58	61.71	123.15	60.25	60.54	130.74	60.15	60.99	213.22	**60.08**	60.17	67.45
C_4	135.03	137.21	541.10	133.70	134.19	874.98	133.63	136.28	940.01	**132.78**	132.99	472.55
datA	-	-	-	**5.35**	**5.35**	0.07	**5.35**	**5.35**	0.21	**5.35**	**5.35**	0.06
datB	-	-	-	**12.78**	**12.78**	1.05	**12.78**	12.86	1.69	**12.78**	**12.78**	0.74
datC	-	-	-	55.53	55.79	132.54	55.53	56.62	201.21	**55.43**	55.48	63.77
datD	-	-	-	134.00	134.38	917.35	133.98	136.19	736.50	**132.85**	133.11	472.33

Tables 10 to 16 now compare the results of our HGS with those of the best methods from the literature for each particular group. We therefore compare with CS+BRKGA and ILS for the first five groups (A, B, C, D and E), and with DQGA and ILS for the remaining groups.

On the first five groups of instances, we observe that HGS-SSP finds, in general, solutions of similar quality as ILS. Group A contains very small instances, and thus all methods find the BKSs. For group B, our algorithm obtained the BKS for all instances, with a CPU time slightly smaller than that of ILS, and much faster than that of CS+BRKGA. For groups C, D and E, HGS-SSP found in most cases the same solution as ILS with similar CPU times. Since these instances are relatively small, however, very little differences between methods can be generally observed, and we should turn towards larger problem instances with better discriminating power.

Tables 15 and 16 now compare the performance of existing methods on the larger instances of Crama et al. [21] and Catanzaro et al. [17]. On these instances, very significant differences of performance can be noticed.

Table 10: Group *A*

				CS+BRKGA			ILS			HGS-SSP		
n	m	C	i	Best	Avg	T	Best	Avg	T	Best	Avg	T
8	15	5	10	**12.00**	**12.00**	2.62	**12.00**	**12.00**	0.04	**12.00**	**12.00**	0.04
8	15	10	30	**6.83**	**6.83**	2.91	**6.83**	**6.83**	0.07	**6.83**	**6.83**	0.04
8	20	5	10	**16.80**	**16.80**	3.26	**16.80**	**16.80**	0.06	**16.80**	**16.80**	0.06
8	20	10	30	**13.07**	**13.07**	3.89	**13.07**	**13.07**	0.13	**13.07**	**13.07**	0.06
8	20	15	60	**7.08**	**7.08**	3.92	**7.08**	**7.08**	0.10	**7.08**	**7.08**	0.05
8	25	5	10	**20.10**	**20.10**	4.99	**20.10**	**20.10**	0.03	**20.10**	**20.10**	0.07
8	25	10	30	**18.20**	**18.20**	4.80	**18.20**	**18.20**	0.12	**18.20**	**18.20**	0.08
8	25	15	60	**12.95**	**12.95**	5.06	**12.95**	**12.95**	0.14	**12.95**	**12.95**	0.08
8	25	20	100	**6.61**	**6.61**	5.18	**6.61**	**6.61**	0.11	**6.61**	**6.61**	0.06

Table 11: Group *B*

				CS+BRKGA			ILS			HGS-SSP		
n	m	C	i	Best	Avg	T	Best	Avg	T	Best	Avg	T
9	15	5	10	**12.20**	**12.20**	2.99	**12.20**	**12.20**	0.07	**12.20**	**12.20**	0.07
9	15	10	30	**7.37**	**7.37**	3.46	**7.37**	**7.37**	0.11	**7.37**	**7.37**	0.07
9	20	5	10	**17.40**	**17.40**	3.44	**17.40**	**17.40**	0.08	**17.40**	**17.40**	0.09
9	20	10	30	**14.17**	**14.17**	4.30	**14.17**	**14.17**	0.18	**14.17**	**14.17**	0.10
9	20	15	60	**7.60**	**7.60**	4.38	**7.60**	**7.60**	0.18	**7.60**	**7.60**	0.08
9	25	5	10	**20.40**	**20.40**	4.48	**20.40**	**20.40**	0.05	**20.40**	**20.40**	0.11
9	25	10	30	**18.77**	**18.77**	5.58	**18.77**	18.77	0.20	**18.77**	**18.77**	0.13
9	25	15	50	**14.74**	14.75	5.60	**14.74**	**14.74**	0.32	**14.74**	**14.74**	0.13
9	25	20	100	**7.19**	**7.19**	5.78	**7.19**	7.19	0.19	**7.19**	**7.19**	0.10

Table 12: Group *C*

				CS+BRKGA			ILS			HGS-SSP		
n	m	C	i	Best	Avg	T	Best	Avg	T	Best	Avg	T
15	15	5	10	**16.60**	16.69	5.83	**16.60**	**16.60**	0.51	**16.60**	**16.60**	0.71
15	15	10	30	**9.80**	9.88	7.80	**9.80**	**9.80**	0.86	**9.80**	**9.80**	0.57
15	20	5	10	**20.60**	20.77	7.97	**20.60**	**20.60**	0.65	**20.60**	**20.60**	0.83
15	20	10	30	**18.33**	18.52	9.81	**18.33**	**18.33**	1.54	**18.33**	**18.33**	0.96
15	20	15	60	**10.52**	10.65	10.55	**10.52**	10.52	1.66	**10.52**	10.52	0.76
15	25	5	10	**27.50**	27.7	10.21	**27.50**	**27.50**	0.63	**27.50**	27.51	1.02
15	25	10	30	**25.07**	25.30	14.85	**25.07**	25.07	1.86	**25.07**	**25.07**	1.28
15	25	15	60	**19.07**	19.27	14.97	**19.07**	**19.07**	2.93	**19.07**	19.07	1.18
15	25	20	100	**9.66**	9.79	15.17	**9.66**	9.66	2.00	**9.66**	9.66	0.90

Table 13: Group D

				CS+BRKGA			ILS			HGS-SSP		
n	m	C	i	Best	Avg	T	Best	Avg	T	Best	Avg	T
20	15	5	10	21.10	21.58	11.84	**20.90**	**20.90**	1.35	**20.90**	**20.90**	2.59
20	15	10	20	**8.20**	8.44	13.55	**8.20**	8.21	2.13	**8.20**	**8.20**	1.83
20	20	5	10	24.30	24.93	16.29	**24.20**	24.24	1.80	**24.20**	**24.20**	3.23
20	20	10	10	**10.60**	10.76	17.66	**10.60**	**10.60**	2.79	**10.60**	**10.60**	2.52
20	20	15	30	**6.67**	6.79	27.08	**6.67**	**6.67**	3.22	**6.67**	**6.67**	2.27
20	25	5	10	**30.10**	30.74	21.04	**30.10**	**30.10**	2.04	**30.10**	30.11	3.99
20	25	10	10	**15.40**	15.47	23.60	**15.40**	**15.40**	3.62	**15.40**	**15.40**	3.37
20	25	15	40	**21.25**	21.75	30.87	**21.25**	21.26	7.19	**21.25**	21.25	3.88
20	25	20	40	**6.15**	6.28	39.01	**6.15**	**6.15**	3.35	**6.15**	**6.15**	2.69
25	15	10	10	**5.90**	6.00	23.12	**5.90**	**5.90**	3.85	**5.90**	**5.90**	3.78
25	20	10	10	**11.60**	12.05	30.17	**11.60**	11.61	9.44	**11.60**	**11.60**	6.58
25	20	15	10	**7.60**	7.82	28.18	**7.60**	**7.60**	9.63	**7.60**	**7.60**	6.74
25	25	10	10	**16.60**	17.06	40.49	**16.60**	16.67	11.33	**16.60**	16.67	8.60
25	25	15	10	**10.00**	**10.00**	60.06	**10.00**	**10.00**	7.60	**10.00**	**10.00**	6.21
25	25	20	30	**5.50**	5.59	72.58	**5.50**	**5.50**	6.68	**5.50**	**5.50**	5.97

Table 14: Group E

				CS+BRKGA			ILS			HGS-SSP		
n	m	C	i	Best	Avg	T	Best	Avg	T	Best	Avg	T
10	10	4	10	**9.50**	**9.50**	1.70	**9.50**	**9.50**	0.08	**9.50**	**9.50**	0.07
10	10	5	10	**6.20**	6.21	2.15	**6.20**	**6.20**	0.08	**6.20**	**6.20**	0.06
10	10	6	10	**4.30**	**4.30**	3.16	**4.30**	**4.30**	0.05	**4.30**	**4.30**	0.06
10	10	7	10	**3.00**	**3.00**	3.79	**3.00**	**3.00**	0.03	**3.00**	**3.00**	0.06
15	20	6	10	**21.40**	21.71	7.79	**21.40**	**21.40**	0.75	**21.40**	**21.40**	0.92
15	20	8	10	**14.20**	14.33	8.43	**14.20**	**14.20**	0.87	**14.20**	14.21	0.84
15	20	10	10	**10.30**	10.34	13.96	**10.30**	**10.30**	0.67	**10.30**	**10.30**	0.72
15	20	12	10	**8.20**	**8.20**	16.44	**8.20**	**8.20**	0.60	**8.20**	**8.20**	0.69

Table 15: Groups C_1, C_2, C_3, and C_4

				ILS			DQGA			HGS-SSP		
n	m	C	i	Best	Avg	T	Best	Avg	T	Best	Avg	T
10	10	4	10	**9.10**	**9.10**	0.08	**9.10**	**9.10**	0.23	**9.10**	**9.10**	0.08
10	10	5	10	**6.20**	**6.20**	0.08	**6.20**	**6.20**	0.26	**6.20**	**6.20**	0.07
10	10	6	10	**4.30**	**4.30**	0.06	**4.30**	**4.30**	0.29	**4.30**	**4.30**	0.06
10	10	7	10	**3.10**	**3.10**	0.04	**3.10**	**3.10**	0.25	**3.10**	**3.10**	0.06
15	20	6	10	**20.60**	20.63	0.82	**20.60**	20.70	2.27	**20.60**	**20.60**	0.93
15	20	8	10	**13.70**	**13.70**	1.06	**13.70**	**13.70**	1.55	**13.70**	**13.70**	0.82
15	20	10	10	**10.10**	**10.10**	0.84	**10.10**	**10.10**	1.07	**10.10**	**10.10**	0.73
15	20	12	10	**7.60**	**7.60**	0.58	**7.60**	**7.60**	0.64	**7.60**	**7.60**	0.64
30	40	15	10	91.40	91.70	84.21	91.30	91.88	184.54	**91.10**	**91.10**	75.85
30	40	17	10	71.30	71.59	128.05	**71.20**	72.08	254.18	**71.20**	**71.20**	66.94
30	40	20	10	50.40	50.71	168.27	50.40	51.36	224.38	**50.20**	50.37	64.40
30	40	25	10	27.90	28.14	142.41	**27.70**	28.64	189.77	27.80	28.02	62.61
40	60	20	10	178.40	179.00	428.41	178.40	180.92	599.65	**177.20**	177.41	512.09
40	60	22	10	151.50	152.04	645.19	151.30	154.00	792.20	**150.50**	150.67	490.88
40	60	25	10	121.00	121.52	985.88	120.90	123.90	979.43	**120.20**	120.44	478.33
40	60	30	10	83.90	84.18	1440.42	83.90	86.30	1388.76	**83.20**	83.44	408.90

Table 16: Groups *datA*, *datB*, *datC* and *datD*

				ILS			DQGA			HGS-SSP		
n	m	C	i	Best	Avg	T	Best	Avg	T	Best	Avg	T
10	10	4	10	**8.50**	**8.50**	0.09	**8.50**	**8.50**	0.19	**8.50**	**8.50**	0.07
10	10	5	10	**5.80**	**5.80**	0.09	**5.80**	**5.80**	0.19	**5.80**	**5.80**	0.06
10	10	6	10	**4.10**	**4.10**	0.05	**4.10**	**4.10**	0.22	**4.10**	**4.10**	0.06
10	10	7	10	**3.00**	**3.00**	0.04	**3.00**	**3.00**	0.25	**3.00**	**3.00**	0.05
15	20	6	10	**20.50**	**20.50**	1.06	**20.50**	20.72	1.85	**20.50**	**20.50**	0.88
15	20	8	10	**13.70**	**13.70**	1.34	**13.70**	13.74	2.42	**13.70**	**13.70**	0.79
15	20	10	10	**9.70**	**9.70**	1.09	**9.70**	9.76	1.58	**9.70**	**9.70**	0.69
15	20	12	10	**7.20**	**7.20**	0.69	**7.20**	**7.20**	0.90	**7.20**	**7.20**	0.60
30	40	15	10	83.90	84.09	96.49	83.80	84.82	165.49	**83.50**	**83.50**	74.71
30	40	17	10	65.50	65.82	132.97	65.50	66.74	223.35	**65.40**	65.43	71.77
30	40	20	10	**46.60**	46.78	167.41	**46.60**	47.82	250.95	**46.60**	46.66	60.58
30	40	25	10	26.30	26.47	133.27	**26.20**	27.10	165.03	**26.20**	26.32	48.01
40	60	20	10	178.00	178.36	474.00	178.00	180.16	243.10	**176.50**	176.71	501.21
40	60	22	10	151.60	152.05	684.95	151.50	153.68	552.95	**150.30**	150.45	437.85
40	60	25	10	121.20	121.68	1037.20	121.20	123.68	853.43	**120.30**	120.61	466.12
40	60	30	10	85.20	85.42	1473.24	85.20	87.22	1296.50	**84.30**	84.66	484.14

For the first two groups of each table (C_1 and C_2 from Table 15 and *datA* and *datB* from Table 16), detailed in the first eight lines, the average solutions of HGS-SSP match the BKS on all instances. HGS-SSP consumes a CPU time similar to that of ILS, and much shorter than that of DQGA. For the last two groups of each table (C_3 and C_4 from Table 15 and *datC* and *datD* from Table 16), HGS-SSP finds the same or new BKSs, with only one exception, and produces significantly better average results than all other methods for a CPU which is two to three times smaller. These significant improvements, on larger instances, show that the proposed method performs a more sustained and diversified search in the solution space.

Comparison with known optimal solutions. Finally, some optimal solutions are known for those instances from previous studies. In particular, Yanasse et al. [57] solved the small instances of groups A, B and E to optimality, as well as some instances of groups C and D. Moreover, Catanzaro et al. [17] solved to optimality some instances with 10 jobs and 10 tools from group *datA*. For all these instances, we verified that the proposed HGS-SSP retrieved the optimal solutions on every run.

4.5 Experiments on larger instances

The benchmark instances of Crama et al. [21], Yanasse et al. [57], and Catanzaro et al. [17] have been widely used in the literature. However, these instances remain limited in terms of number of jobs and tools, and start to lose their discriminative power: state-of-the-art heuristics now find solutions which are close to each other. Moreover, as discussed in Shirazi and Frizelle [46], companies are regularly confronted with problems that contain over sixty jobs. To stimulate future research on the SSP and allow comparisons on more challenging instances, we generated a set of additional instances of a size comparable to that reported in Shirazi and Frizelle [46], and provide in Table 17 (in Appendix) the results of our method for future reference. These datasets are also available at `https://w1.cirrelt.ca/~vidalt/en/research-data.html`.

The new instances can be divided into three groups (F_1, F_2 and F_3), each with the same number of jobs and tools, but different capacities. For each capacity, we generated five instances (each line from Table 17). The groups have a number of jobs ranging from 50 to 70 and a number of tools ranging from 75 to 105. Despite the higher CPU time required to solve these larger instances, the objective values obtained by HGS-SSP over multiple runs remained very stable. This shows that the method is robust in the sense that it consistently achieves a similar solution quality. Future research on these instances will allow more extensive algorithm comparisons.

5 Conclusions

In this article, we proposed a simple and efficient metaheuristic for the well-known job sequencing and tool switching problem (SSP). The proposed approach contrasts with previous algorithms, which had a tendency to be over-engineered and generally complex. Our HGS-SSP finds a good balance between aggressive *intensification*, achieved by an efficient local search in the space of permutations, and *diversification* [12, 53], obtained via a simple crossover, population diversity management strategy, and tie-breaking objective to guide the search towards potential tool switches reductions. Through extensive experiments on several sets of benchmark instances, we observed that our algorithm significantly outperforms existing methods in terms of solution quality and CPU time. To guide future research, we also evaluated the impact of each neighborhood and methodological choice in the method. We observed that adopting a defined order of neighborhood exploration in the local search is beneficial for this problem since 2OPT tends to operate larger structural changes than SWAP or RELOCATE. The tie-breaking objective was also essential for a good performance, along with our population diversity management techniques.

Many promising research perspectives are open. The proposed HGS-SSP could be extended and tested on other SSP variants, e.g., with multiple machines [11] or with different objective functions [15]. As the proposed method exploits an indirect solution representation as a job permutation with a solution decoder (the KTNS algorithm), it conducts the search on a much smaller space at the price of more computationally expensive solution evaluations. Such a disciplined analysis of metaheuristics and structural problem decompositions should be pursued and extended to other problems, seeking to achieve a search space reduction which is as large as possible for a decoder which is a fast as possible, opening the way to a variety of complexity analyses and algorithmic contributions. Finally, the HGS framework could be extended to efficiently solve other difficult permutation-based optimization problems.

Acknowledgments

This research was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), grants 428549/2016-0, 307843/2018-1 and 308528/2018-2, by FAPERJ (grant E-26/203.310/2016), and by Comissão de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Finance Code 001.

References

[1] Adjiashvili, D., Bosio, S., Zemmer, K., 2015. Minimizing the number of switch instances on a flexible machine in polynomial time. Operations Research Letters 43, 317–322.

[2] Ahmadi, E., Goldengorin, B., Süer, G.A., Mosadegh, H., 2018. A hybrid method of 2-TSP and novel learning-based GA for job sequencing and tool switching problem. Applied Soft Computing 65, 214–229.

[3] Al-Fawzan, M., Al-Sultan, K., 2003. A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. Computers & Industrial Engineering 44, 35–47.

[4] Amaya, J.E., Cotta, C., Fernández, A.J., 2008. A memetic algorithm for the tool switching problem, in: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics, Springer, Berlin, Heidelberg. pp. 190–202.

[5] Amaya, J.E., Cotta, C., Fernández-Leiva, A.J., 2011. Memetic cooperative models for the tool switching problem. Memetic Computing 3, 199–216.

[6] Amaya, J.E., Cotta, C., Fernández-leiva, A., 2013/05. Cross entropy-based memetic algorithms: An application study over the tool switching problem. International Journal of Computational Intelligence Systems 6, 559–584.

[7] Amaya, J.E., Cotta, C., Fernández-Leiva, A.J., 2012. Solving the tool switching problem with memetic algorithms. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 26, 221–235.

[8] Amaya, J.E., Cotta, C., Leiva, A.J.F., 2010. Hybrid cooperation models for the tool switching problem. Springer, Berlin, Heidelberg. pp. 39–52.

[9] Avci, S., Akturk, M., 1996. Tool magazine arrangement and operations sequencing on CNC machines. Computers & Operations Research 23, 1069–1081.

[10] Bard, J.F., 1988. A Heuristic for Minimizing the Number of Tool Switches on a Flexible Machine. IIE Transactions 20, 382–391.

[11] Beezão, A.C., Cordeau, J.F., Laporte, G., Yanasse, H.H., 2017. Scheduling identical parallel machines with tooling constraints. European Journal of Operational Research 257, 834–844.

[12] Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys (CSUR) 35, 268–308.

[13] Bouzina, K., Emmons, H., 1996. Interval scheduling on identical machines. Journal of Global Optimization 9, 379–393.

[14] Burger, A.P., Jacobs, C.G., van Vuuren, J.H., Visagie, S.E., 2015. Scheduling multi-colour print jobs with sequence-dependent setup times. Journal of Scheduling 18, 131–145.

[15] Calmels, D., 2019. The job sequencing and tool switching problem: State-of-the-art literature review, classification, and trends. International Journal of Production Research 57, 5005–5025.

[16] Carlisle, M., Lloyd, E., 1995. On the k-coloring of intervals. Discrete Applied Mathematics 59, 225–235.

[17] Catanzaro, D., Gouveia, L., Labbé, M., 2015. Improved integer linear programming formulations for the job Sequencing and tool Switching Problem. European Journal of Operational Research 244, 766–777.

[18] Chaves, A., Lorena, L., Senne, E., Resende, M., 2016. Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. Computers & Operations Research 67, 174–183.

[19] Crama, Y., 1997. Combinatorial optimization models for production scheduling in automated manufacturing systems. European Journal of Operational Research 99, 136–153.

[20] Crama, Y., van de Klundert, J., 1999. Worst-case performance of approximation algorithms for tool management problems. Naval Research Logistics (NRL) 46, 445–462.

[21] Crama, Y., Kolen, A.W.J., Oerlemans, A.G., Spieksma, F.C.R., 1994. Minimizing the number of tool switches on a flexible machine. International Journal of Flexible Manufacturing Systems 6, 33–54.

[22] Djellab, H., Djellab, K., Gourgand, M., 2000. A new heuristic based on a hypergraph representation for the tool switching problem. International Journal of Production Economics 64, 165–176.

[23] Fathi, Y., Barnette, K.W., 2002. Heuristic procedures for the parallel machine problem with tool switches. International Journal of Production Research 40, 151–164.

[24] Furrer, M., Mütze, T., 2017. An algorithmic framework for tool switching problems with multiple objectives. European Journal of Operational Research 259, 1003–1016.

[25] Gendreau, M., Hertz, A., Laporte, G., 1992. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. Operations Research 40, 1086–1094.

[26] Ghiani, G., Grieco, A., Guerriero, E., 2007. An exact solution to the TLP problem in an NC machine. Robotics and Computer-Integrated Manufacturing 23, 645–649.

[27] Ghiani, G., Grieco, A., Guerriero, E., 2010. Solving the Job Sequencing and Tool Switching Problem as a nonlinear least cost Hamiltonian cycle problem. Networks 55, 379–385.

[28] Ghrayeb, O.A., Phojanamongkolkij, N., Finch, P.R., 2003. A mathematical model and heuristic procedure to schedule printed circuit packs on sequencers. International Journal of Production Research 41, 3849–3860.

[29] Gray, A.E., Seidmann, A., Stecke, K.E., 1993. A Synthesis of Decision Models for Tool Management in Automated Manufacturing. Management Science 39, 549–567.

[30] Gribel, D., Vidal, T., 2019. HG-means: A scalable hybrid metaheuristic for minimum sum-of-squares clustering. Pattern Recognition 88, 569–583.

[31] Hertz, A., Laporte, G., Mittaz, M., Stecke, K.E., 1998. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE Transactions 30, 689–694.

[32] Hertz, A., Widmer, M., 1996. An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. Discrete Applied Mathematics 65, 319–345.

[33] Keung, K.W., Ip, W.H., Lee, T.C., 2001. The Solution of a Multi-Objective Tool Selection Model Using the GA Approach. The International Journal of Advanced Manufacturing Technology 18, 771–777.

[34] Konak, A., Kulturel-Konak, S., 2007. 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, pp. 43–48.

[35] Konak, A., Kulturel-Konak, S., Azizoğlu, M., 2008. Minimizing the number of tool switching instants in Flexible Manufacturing Systems. International Journal of Production Economics 116, 298–307.

[36] Laporte, G., Salazar-Gonzáles, J.J., Semet, F., 2004. Exact algorithms for the job sequencing and tool switching problem. IIE Transactions 36, 37–45.

[37] Matzliach, B., 1998. The online tool switching problem with non-uniform tool size. International Journal of Production Research 36, 3407–3420.

[38] Mütze, T., 2014. Scheduling with few changes. European Journal of Operational Research 236, 37–50.

[39] Oliver, I., Smith, D., Holland, J., 1987. A study of permutation crossover operators on the traveling salesman problem., in: Grefenstette, J. (Ed.), Genetic Algorithms and their Applications: Proceedings of the Second International Conference, pp. 224–230.

[40] Paiva, G.S., Carvalho, M.A.M., 2017. Improved heuristic algorithms for the Job Sequencing and Tool Switching Problem. Computers & Operations Research 88, 208–219.

[41] Privault, C., Finke, G., 1995. Modelling a tool switching problem on a single NC-machine. Journal of Intelligent Manufacturing 6, 87–94.

[42] Raduly-Baka, C., Nevalainen, O.S., 2015. The modular tool switching problem. European Journal of Operational Research 242, 100–106.

[43] Salonen, K., Raduly-Baka, C., Nevalainen, O.S., 2006a. A note on the tool switching problem of a flexible machine. Computers & Industrial Engineering 50, 458–465.

[44] Salonen, K., Smed, J., Johnsson, M., Nevalainen, O., 2006b. Grouping and sequencing PCB assembly jobs with minimum feeder setups. Robotics and Computer-Integrated Manufacturing 22, 297–305.

[45] Senne, E.L.F., Yanasse, H.H., 2009. 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, World Scientific and Engineering Academy and Society, Stevens Point, USA. pp. 68–72.

[46] Shirazi, R., Frizelle, G.D.M., 2001. Minimizing the number of tool switches on a flexible machine: An empirical study. International Journal of Production Research 39, 3547–3560.

[47] Tang, C.S., Denardo, E.V., 1988a. Models arising from a flexible manufacturing machine, Part I: Minimization of the number of tool switches. Operations Research 36, 767–777.

[48] Tang, C.S., Denardo, E.V., 1988b. Models arising from a flexible manufacturing machine, Part II: Minimization of the number of switching instants. Operations Research 36, 778–784.

[49] Toffolo, T.A., Vidal, T., Wauters, T., 2019. Heuristics for vehicle routing problems: Sequence or set optimization? Computers & Operations Research 105, 118–131.

[50] Tzur, M., Altman, A., 2004. Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. IIE Transactions 36, 95–110.

[51] Vidal, T., 2017. Node, edge, arc routing and turn penalties: Multiple problems – One neighborhood extension. Operations Research 65, 992–1010.

[52] Vidal, T., Battarra, M., Subramanian, A., Erdogan, G., 2015. Hybrid metaheuristics for the clustered vehicle routing problem. Computers & Operations Research 58, 87–99.

[53] Vidal, T., Crainic, T., Gendreau, M., Prins, C., 2013. Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. European Journal of Operational Research 231, 1–21.

[54] Vidal, T., Crainic, T.G., Gendreau, M., Lahrichi, N., Rei, W., 2012. A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems. Operations Research 60, 611–624.

[55] Vidal, T., Crainic, T.G., Gendreau, M., Prins, C., 2014. A unified solution framework for multi-attribute vehicle routing problems. European Journal of Operational Research 234, 658–673.

[56] Whitley, D., 2019. Next Generation Genetic Algorithms: A User's Guide and Tutorial. Springer International Publishing. pp. 245–274.

[57] Yanasse, H.H., Rodrigues, R.d.C.M., Senne, E.L.F., 2009. Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. Gestão & Produção 16, 370–381. In Portuguese.

[58] Zeballos, L., 2010. A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. Robotics and Computer-Integrated Manufacturing 26, 725–743.

[59] Zhou, B.H., Xi, L.F., Cao, Y.S., 2005. A beam-search-based algorithm for the tool switching problem on a flexible machine. The International Journal of Advanced Manufacturing Technology 25, 876–882.

Appendix A Results on a the new set of larger instances

Table 17: Groups F_1, F_2 and F_3

				HGS-SSP		
n	m	C	i	Best	Avg	T
50	75	25	5	268.40	268.80	2349.09
50	75	30	5	196.60	197.22	2064.81
50	75	35	5	147.00	148.14	1848.54
50	75	40	5	109.80	110.62	1607.07
60	90	35	5	414.80	415.54	7607.10
60	90	40	5	320.20	321.08	7108.39
60	90	45	5	247.00	248.34	5253.31
60	90	50	5	191.20	192.86	5595.93
70	105	40	5	576.80	577.82	16353.88
70	105	45	5	459.00	460.18	14264.06
70	105	50	5	369.20	371.00	12360.99
70	105	55	5	298.20	299.80	11756.43