



Jordana Zerpini Mecler

**A simple and effective hybrid genetic search for
the job sequencing and tool switching problem**

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-graduação em
Informática of PUC-Rio in partial fulfillment of the requirements
for the degree of Mestre em Informática.

Advisor : Prof. Thibaut Victor Gaston Vidal
Co-advisor: Prof. Anand Subramanian

Rio de Janeiro
April 2020



Jordana Zerpini Mecler

**A simple and effective hybrid genetic search for
the job sequencing and tool switching problem**

Dissertation presented to the Programa de Pós-graduação em
Informática of PUC-Rio in partial fulfillment of the requirements
for the degree of Mestre em Informática. Approved by the
Examination Committee.

Prof. Thibaut Victor Gaston Vidal

Advisor

Departamento de Informática – PUC-Rio

Prof. Anand Subramanian

Co-advisor

Departamento de Sistemas de Computação – UFPB

Prof. Marco Serpa Molinaro

Departamento de Informática – PUC-Rio

Prof. Marcus Vinícius Soledade Poggi de Aragão

Departamento de Informática – PUC-Rio

Rio de Janeiro, April 29th, 2020

All rights reserved.

Jordana Zerpini Mecler

Obtained a bachelor in Computer Engineering (2017) at the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). Earned a scholarship by CAPES for the masters.

Bibliographic data

Mecler, Jordana Zerpini

A simple and effective hybrid genetic search for the job sequencing and tool switching problem / Jordana Zerpini Mecler; advisor: Thibaut Victor Gaston Vidal; co-advisor: Anand Subramanian. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2020.

v., 43 f: il. color. ; 30 cm

Dissertação (Mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui bibliografia

1. Metaheurística. 2. Otimização combinatória. 3. Sequenciamento de tarefas. 4. Troca de ferramentas. 5. Busca genética híbrida. I. Vidal, Thibaut Victor Gaston. II. Subramanian, Anand. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Acknowledgments

Firstly, I would like to thank my incredible advisor, Thibaut Vidal, for all the discussions, help and support in the past years and my coadvisor, Anand Subramanian, for helping making it fun to write a paper and becoming a great friend of mine in the process. I would also like to thank the examination committee, Marco Molinaro and Marcus Poggi, for their insightful comments and suggestions.

I would like to thank my family, Elizabeth Zerpini, Ian Mecler, Katia Mecler, Rosinha Goldenstein, Nair Zerpini, Edir Semblano and specially my brother, Davi Mecler, who is also a great colleague in the area.

My sincere gratitude also goes to my fellow colleagues, specially Rafael França, Bianca Teixeira, Lucas Murtinho, Pedro Ferreira and Marcelo Paulon, for their amazing friendship and important moments shared in the past couple of years.

Last but not least, I would like to thank my best friends, Laura Becker and Gabriela Mariz, for all their love, always.

This study was financed in part by the Coordenação de Aperfeiçoamento Pessoal de Nível Superior (CAPES) – Finance Code 001, and I am very grateful for this scholarship.

Abstract

Mecler, Jordana Zerpini; Vidal, Thibaut Victor Gaston (Advisor); Subramanian, Anand (Co-Advisor). **A simple and effective hybrid genetic search for the job sequencing and tool switching problem**. Rio de Janeiro, 2020. 43p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

EN

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

Metaheuristic; Combinatorial optimization; Job sequencing; Tool switching; Hybrid genetic search

Resumo

Mecler, Jordana Zerpini; Vidal, Thibaut Victor Gaston; Subramanian, Anand. **Uma busca genética híbrida simples e efetiva para o problema de sequenciamento de tarefas e troca de ferramentas**. Rio de Janeiro, 2020. 43p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

PT

O problema de sequenciamento de tarefas e troca de ferramentas (job sequencing and tool switching problem - SSP) tem sido extensivamente estudado na área de pesquisa operacional, devido à sua relevância prática e interesse metodológico. Dada uma máquina que pode carregar uma quantidade limitada de ferramentas simultaneamente e um número de tarefas que requerem um subconjunto das ferramentas disponíveis, o SSP procura uma sequência de tarefas que minimize o número total de trocas de ferramentas na máquina. Para resolver este problema, é proposta uma busca genética híbrida simples e efetiva baseada em uma representação de solução genérica, um operador de decodificação sob medida, buscas locais eficientes e técnicas de gerenciamento de diversidade. Para orientar a busca, um objetivo secundário desenvolvido para tratar empates é introduzido. Essas técnicas permitem explorar soluções estruturalmente distintas e escapar de ótimos locais. Conforme apresentado nos experimentos computacionais em instâncias clássicas, o algoritmo proposto supera significativamente todas as abordagens anteriores, mesmo sendo de fácil entendimento e implementação. Por fim, resultados obtidos em um novo conjunto de instâncias maiores são reportados para estimular futuras pesquisas e análises comparativas.

Palavras-chave

Metaheurística; Otimização combinatória; Sequenciamento de tarefas; Troca de ferramentas; Busca genética híbrida

Table of contents

1	Introduction	10
2	Related studies	13
3	Proposed methodology	18
3.1	Solution evaluation	19
3.2	Generation of new solutions	21
3.3	Biased fitness and population diversity management	22
4	Computational experiments	23
4.1	Classical benchmark instances	23
4.2	Parameters calibration	24
4.3	Sensitivity analysis	25
4.4	Comparison with other algorithms on classical benchmark instances	27
4.5	Experiments on larger instances	33
5	Conclusions	37

List of figures

Figure 1.1	Example with 4 jobs, 5 tools and a machine capacity of 3. The tools which are kept in the machine magazine but not currently used are underlined. Tool switches are circled.	10
Figure 3.1	Illustration of the OX crossover on a small example with 10 jobs	21
Figure 4.1	Distribution of objective values in the population after first survivors selection, halfway through the execution and after last survivors selection.	27
Figure 4.2	Average number of tool switches over time for the instances of Group F_1	36

List of tables


Table 2.1	Summary of SSP studies	16
Table 3.1	Required tools matrix – Before KTNS	19
Table 3.2	Loaded tools matrix – After KTNS	20
Table 4.1	SSP instances	24
Table 4.2	Varying population size	25
Table 4.3	Varying μ^{elite}	25
Table 4.4	Varying μ^{close}	25
Table 4.5	Sensitivity Analysis	26
Table 4.6	Summary of the comparative analyses	29
Table 4.7	Performance comparison on Group <i>A</i> instances	30
Table 4.8	Performance comparison on Group <i>B</i> instances	30
Table 4.9	Performance comparison on Group <i>C</i> instances	30
Table 4.10	Performance comparison on Group <i>D</i> instances	31
Table 4.11	Performance comparison on Group <i>E</i> instances	31
Table 4.12	Performance comparison on Groups C_1 , C_2 , C_3 , and C_4	32
Table 4.13	Performance comparison on Groups <i>datA</i> , <i>datB</i> , <i>datC</i> and <i>datD</i>	32
Table 4.14	Performance comparison on Groups F_1 , F_2 and F_3	35

1

Introduction

The *job sequencing and tool switching problem* (SSP) considers a set of jobs $J = \{1, \dots, n\}$, a set of tools $T = \{1, \dots, m\}$, 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 may 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, calculated as the number of times a tool is *replaced* by another. A problem and solution example is provided in Figure 1.1.

Instance									
Jobs	1	2	3	4					
Tools	1	2	1	2					
	4	3	4	4					
	5	5							



Solution									
Jobs	2	4	3	1					
Tools	2	2	①	1					
	3	④	4	4					
	5	<u>5</u>	<u>5</u>	<u>5</u>					

Figure 1.1: Example with 4 jobs, 5 tools and a machine capacity of 3. The tools which are kept in the machine magazine but not currently used are underlined. Tool switches are circled.

The SSP arises in many applications, including circuit-board manufacturing (Privault and Finke, 1995), computer memory management (Ghiani et al., 2007) and pharmaceutical packaging (Mütze, 2014). In the first application, the jobs represent circuits which need to be placed on a board and the tools are the surface-mount component 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 any fixed job sequence, the tooling problem can be solved in time $\mathcal{O}(nm)$ with the “keep tools needed soonest”

(KTNS) policy (Tang and Denardo, 1988a). In contrast, finding an optimal job sequence (i.e., solving the SSP) is known to be \mathcal{NP} -hard (Crama et al., 1994).

Due to its 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 (solutions with the same cost obtained by simple reversals of job sequences) tend to diminish the effectiveness of local searches. 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., Gribel and Vidal, 2019; Toffolo et al., 2019; Vidal, 2017; Vidal et al., 2015, 2014).

In this work, 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 Vidal et al. (2012, 2014)— 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 (see definition in Section 3.1) to guide the search towards solution improvements. The main contributions of this dissertation 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 tool 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 and comparative analyses on a set of larger-scale instances to stimulate future research.

The remainder of the dissertation is organized as follows. We firstly review the literature in Chapter 2. Then, we describe the proposed algorithm in Chapter 3, present our experimental results in Chapter 4, and conclude in Chapter 5.

The SSP was formally proposed in the late 1980s by Tang and Denardo (1988a). 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 (Bouzina and Emmons, 1996; Carlisle and Lloyd, 1995).

Later, Gray et al. (1993) discussed decision models for tooling management, including the SSP, while Crama et al. (1994) proved that this problem is \mathcal{NP} -hard. A network flow approach for the nonuniform case was proposed by Privault and Finke (1995). Crama (1997) and Crama and van de Klundert (1999) introduced optimization models and studied the worst-case performance of some approximation algorithms for the SSP, as well as other production planning and scheduling problems. An empirical study of the SSP in manufacturing companies was conducted by Shirazi and Frizelle (2001). The authors concluded that the heuristics available at the time outperformed the methods used in companies. Reflecting the variety of application cases, other studies have considered variants of the SSP, e.g., minimizing the tool switching instants (Adjashvili et al., 2015; Konak and Kulturel-Konak, 2007; Tang and Denardo, 1988b), considering multiple objectives (Keung et al., 2001; Salonen et al., 2006b), parallel machines Beezão et al. (2017); Fathi and Barnette (2002); Zeballos (2010), different tool sizes (Matzliach, 1998; Tzur and Altman, 2004), and other features (Avci and Akturk, 1996; Furrer and Mütze, 2017; Ghrayeb et al., 2003; Hertz and Widmer, 1996; Raduly-Baka and Nevalainen, 2015; Salonen et al., 2006b; Tzur and Altman, 2004).

Several exact methods based on mathematical programming have been proposed for the SSP. Tang and Denardo (1988a) formulated the problem as an integer linear program (ILP). This line of study was followed by Laporte et al. (2004), Ghiani et al. (2007) and Catanzaro et al. (2015) who also proposed ILP-based branch-and-cut approaches. Ghiani et al. (2007) used the same formulation as Laporte et al. (2004) with additional symmetry-breaking strategies. The linear programming relaxation of the formulation of (Tang and Denardo, 1988a) is known to be weaker than that of (Laporte et al.,

2004), which is in turn weaker than that of (Catanzaro et al., 2015). The latter approach represents the best exact algorithm to date, but it cannot consistently solve instances with more than 10 jobs and tools. Branch-and-bound algorithms based on combinatorial bounds (Laporte et al., 2004) and enumerative approaches (Yanasse et al., 2009) can be competitive with branch-and-cut methods on some instances, especially when the number of required tools is close to the machine capacity. Bard (1988) finally developed a nonlinear integer program and solved it using a dual-based relaxation heuristic, while Ghiani et al. (2010) modeled and solved the problem as a nonlinear least-cost Hamiltonian cycle problem.

Other SSP studies have been principally focused on heuristics and metaheuristics. Tang and Denardo (1988a) 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. (2006b) considered both job-grouping and tool-switch objectives but also reported heuristic results for the classical SSP. Burger et al. (2015) developed job-grouping heuristics for the color print scheduling problem, which is an application of the SSP. Crama et al. (1994) proposed heuristics based on the traveling salesman problem (TSP) for a graph representation similar to that of Tang and Denardo (1988a). Hertz et al. (1998) improved on earlier heuristics using methods such as GENIUS (Gendreau et al., 1992). Privault and Finke (1995) 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. (2000) formulated the problem using a hypergraph representation and implemented a constructive method. Salonen et al. (2006a) 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 (Al-Fawzan and Al-Sultan, 2003; Konak et al., 2008), iterated local search (ILS) (Paiva and Carvalho, 2017), and beam search (Senne and Yanasse, 2009; Zhou et al., 2005). Population-based methods have also been successful. Hybrid methods combining genetic algorithms (GA) with other search procedures can be found in (Ahmadi et al., 2018; Amaya et al., 2008, 2011, 2013, 2012, 2010; Chaves et al., 2016). Amaya et al. (2013, 2008, 2012) 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. (2011, 2013, 2010) combined GA with a multi-agent approach or cross-entropy methods. Finally,

Chaves et al. (2016) combined clustering search (CS) with a biased random-key genetic algorithm (BRKGA), while Ahmadi et al. (2018) solved the problem as a TSP of second order (Jäger and Molitor, 2008) with a learning-based GA. Table 2.1 summarizes the SSP studies in chronological order, lists their main contributions as well as the origin of the benchmark instances considered. We also refer to Calmels (2019) for a comprehensive literature review and classification of SSP variants.

Table 2.1: Summary of SSP studies

Work	Year	Approach	Instances
Tang and Denardo (1988a)	1988	Exact methods + heuristics	Tang and Denardo (1988a)
Bard (1988)	1988	Exact methods + heuristics	Bard (1988)
Crama et al. (1994)	1994	Heuristics	Crama et al. (1994)
Privault and Finke (1995)	1995	Network flow formulation + heuristics	Privault and Finke (1995)
Hertz et al. (1998)	1998	Heuristics	Hertz et al. (1998)
Crama and van de Klundert (1999)	1999	Worst-case analysis	–
Djellab et al. (2000)	2000	Heuristics	Crama et al. (1994)
Shirazi and Frizelle (2001)	2001	Empirical study	Real life
Al-Fawzan and Al-Sultan (2003)	2003	Tabu search	Al-Fawzan and Al-Sultan (2003)
Laporte et al. (2004)	2004	Exact methods	Hertz et al. (1998); Laporte et al. (2004)
Zhou et al. (2005)	2005	Beam search	Bard (1988)
Salonen et al. (2006a)	2006	Heuristics	Crama et al. (1994)
Salonen et al. (2006b)	2006	Exact methods + heuristics	Salonen et al. (2006b)
Ghani et al. (2007)	2007	Exact methods	Hertz et al. (1998); Laporte et al. (2004)
Amaya et al. (2008)	2008	Memetic algorithms	Amaya et al. (2008)
Konak et al. (2008)	2008	Tabu search	Konak et al. (2008)
Senne and Yanasse (2009)	2009	Beam search	Senne and Yanasse (2009)
Yanasse et al. (2009)	2009	Exact methods	Yanasse et al. (2009)
Ghani et al. (2010)	2010	Exact methods	Ghani et al. (2010)
Amaya et al. (2010)	2010	Hybrid cooperative	Amaya et al. (2010)
Amaya et al. (2011)	2011	Memetic cooperative	Amaya et al. (2011)
Amaya et al. (2012)	2012	Memetic algorithms	Amaya et al. (2012)
Amaya et al. (2015)	2013	Cross entropy-based memetic algorithms	Amaya et al. (2015)
Burger et al. (2015)	2015	Heuristics	Burger et al. (2015) & Real life
Catanzaro et al. (2015)	2015	Exact methods	Catanzaro et al. (2015)
Chaves et al. (2016)	2016	Hybrid metaheuristics	Crama et al. (1994); Yanasse et al. (2009)
Paiva and Carvalho (2017)	2017	Iterated local search	Crama et al. (1994); Catanzaro et al. (2015); Yanasse et al. (2009)
Almadi et al. (2018)	2018	Hybrid metaheuristics	Crama et al. (1994); Catanzaro et al. (2015)

Most of the aforementioned GA-based implementations use traditional parent-selection strategies (Whitley, 2019) such as roulette wheel (Ahmadi et al., 2018) and binary tournament (Amaya et al., 2008, 2011, 2012). Survivor selections are solely based on individual objective values, typically obtained with KTNS (Ahmadi et al., 2018; Amaya et al., 2008, 2011, 2012; Chaves et al., 2016). Ahmadi et al. (2018) also take into account a chromosome similarity function to remove individuals from the population. The crossover operators applied are parameterized uniform crossover (Chaves et al., 2016), uniform cycle crossover (Amaya et al., 2012), alternating position (APX) (Amaya et al., 2008, 2011) and partially mapped (PMX) (Ahmadi et al., 2018). Finally, although some authors have developed local search procedures based on problem-specific metrics such as 0-blocks and 1-blocks (Crama et al., 1994), ties arising during move evaluations are generally handled arbitrarily (Amaya et al., 2011, 2012).

In terms of performance, the current best metaheuristics are those of Chaves et al. (2016), Paiva and Carvalho (2017) and Ahmadi et al. (2018). The ILS of Paiva and Carvalho (2017) achieves the best average solution quality so far, but at the expense of a computational time which is larger on large-scale instances. Over multiple runs, the method of Ahmadi et al. (2018) also produced several best known solutions. During our experimental comparisons, we will therefore compare our results with those of the three aforementioned methods.

3

Proposed methodology

To solve the SSP, we exploit the hybrid generic search (HGS) paradigm (Vidal et al., 2012, 2014) 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) (Oliver et al., 1987) 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 μ random individuals subject to local search
 - 2: **while** the 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 λ 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 the HGS of Vidal et al. (2012, 2014). This search paradigm, combining an adaptive population-diversity management with efficient local searches and solution decoders, has been applied with great success to the family of vehicle routing problems, but has rarely been tested on other permutation-based problems. In our application of HGS, each component and operator (solution evaluation, crossover, local search) has been tailored

to the SSP and plays a major role in the success of the method. Moreover, compared to previous studies in the domain, our genetic algorithm is the first to systematically use a hierarchical objective for solution evaluations and a fitness function accounting for population diversity. 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 (Tang and Denardo, 1988a) as a solution decoder to find an optimal tooling strategy. As illustrated in Table 3.1 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 3.1: Required tools matrix – Before KTNS

		Jobs									
Tools		1	2	3	4	5	6	7	8	9	10
	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 3.2 displays the loaded tools matrix obtained with KTNS

on the previous example. For each job, the tools which are kept in the magazine but not required are represented by an underscored 1.

Table 3.2: 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	<u>1</u>	<u>1</u>	0	1	1	1	0	0	0
	3	0	1	<u>1</u>	<u>1</u>	1	1	0	0	0	0
	4	0	0	1	1	0	0	1	<u>1</u>	1	0
	5	0	0	0	0	1	1	<u>1</u>	0	0	0
	6	1	0	0	0	0	0	0	1	0	0
	7	0	0	1	<u>1</u>	1	0	0	0	0	1
	8	0	0	0	1	0	0	0	1	<u>1</u>	1
	9	0	1	0	0	0	1	1	1	1	<u>1</u>
	10	0	0	0	0	0	0	0	0	1	<u>1</u>

The number of tool switches can be obtained from Table 3.2 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 GA 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. (1994). 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 3.2, the 0-blocks have been highlighted in gray. 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 $x \in \{1, \dots, k_j(S)\}$. For example, the number of 0-blocks for tool 4 in Table 3.2 is $k_4(S) = 1$, and the size of this block is $b_4^1(S) = 2$. The tie-breaking objective is then evaluated as:

$$\Phi'(S) = \sum_{j=1}^m \sum_{x=1}^{k_j(S)} \sqrt{b_j^x(S)}. \quad (3-1)$$

In this equation, 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 picking up 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) (Oliver et al., 1987) is applied on the two parents to generate a single child. This crossover, illustrated in Figure 3.1, 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, starting from the second cutting point, to fill the empty positions with the missing visits.

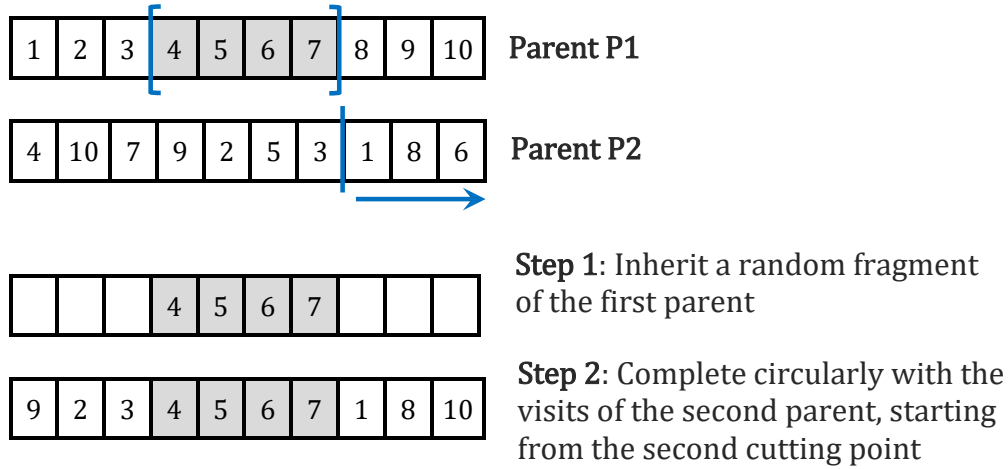


Figure 3.1: Illustration of the OX crossover on a small example with 10 jobs

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 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 Vidal et al. (2012, 2014). It depends on two parameters: the number μ^{close} of closest individuals in the population considered in the distance measure, and the number of elite individuals μ^{elite} one wishes to preserve. 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 individuals, the distance between two individuals (represented as job permutations) being defined as the number of job pairs in the first individual which are broken in the second one (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 $f_{\mathcal{P}}^{div}(p)$ and a quality rank $f_{\mathcal{P}}^{obj}(p)$. Finally, the biased fitness of each individual is calculated as:

$$f_{\mathcal{P}}(p) = f_{\mathcal{P}}^{obj}(p) + \left(1 - \frac{\mu^{elite}}{|\mathcal{P}|}\right) \times f_{\mathcal{P}}^{div}(p). \quad (3-2)$$

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 Vidal et al. (2012), 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 compiled it with g++ 8.1 using the -O3 flag. The source code and benchmark instances are accessible at <https://github.com/jordanamecler/HGS-SSP>. We first describe the characteristics of the classical instances used in our experiments in Section 4.1. Next, we explain in Section 4.2 how we calibrated the parameters of the algorithm and investigate the impact of several methodological choices in Section 4.3. Finally, we compare our algorithm with other state-of-the-art methods for the existing instances in Section 4.4 as well as new larger ones in Section 4.5. For the existing instances, we ran our code on a single thread of an Intel Core i5-4288U 2.6 GHz processor with 8 GB of RAM using macOS High Sierra 10.13.6 and compare our solutions with published results. For the new instances, no results were available and therefore we contacted the authors to obtain source codes. Chaves et al. (2016) and Paiva and Carvalho (2017) kindly provided their original code to us. As discussed in Section 4.5, we could therefore compare these two algorithms with the proposed HGS-SSP using the same computational environment: a single thread of an Intel Gold 6148 Skylake 2.4 GHz processor with 256 MB of reserved RAM running CentOS 7.7.1908.

4.1

Classical benchmark instances

We first consider the classical benchmark instances of Crama et al. (1994), Yanasse et al. (2009), and Catanzaro et al. (2015) to evaluate the performance of the proposed method. As summarized in Table 4.1, these instances are organized in different groups corresponding to different size parameters. Groups A , B , C , D , and E (from Yanasse et al. (2009)) contain a total of 1350 instances with 8 to 25 jobs. Groups C_1 , C_2 , C_3 , and C_4 (from Crama et al. (1994)) as well as $datA$, $datB$, $datC$, and $datD$ (from Catanzaro et al. (2015)) contain 40 instances each and include 10 to 40 jobs.

Table 4.1: SSP instances

Group	#Jobs		#Tools		Capacity		#Instances
	Min	Max	Min	Max	Min	Max	
<i>A</i>	8	8	15	25	5	20	340
<i>B</i>	9	9	15	25	5	20	330
<i>C</i>	15	15	15	25	5	20	340
<i>D</i>	20	25	15	25	5	20	260
<i>E</i>	10	15	10	20	4	12	80
<i>C</i> ₁	10	10	10	10	4	7	40
<i>C</i> ₂	15	15	20	20	6	12	40
<i>C</i> ₃	30	30	40	40	15	25	40
<i>C</i> ₄	40	40	60	60	20	30	40
<i>datA</i>	10	10	10	10	4	7	40
<i>datB</i>	15	15	20	20	6	12	40
<i>datC</i>	30	30	40	40	15	25	40
<i>datD</i>	40	40	60	60	20	30	40

4.2

Parameters calibration

We conducted preliminary experiments on a subset of the instances to find suitable parameter values for our algorithm. We opted to perform the calibration tests on the benchmark of Crama et al. (1994) (groups C_1, C_2, C_3 , and C_4) 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 4.2, 4.3, and 4.4 show the average results of different method configurations for the benchmark instances of Crama et al. (1994). In these tables, **Avg** is the average objective value over 10 runs and over all instances and **T** is the average CPU time in seconds. 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} decreases, 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 4.2: 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 4.3: 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 4.4: 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 on the same subset of instances as in Section 4.2. 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 4.5 and compared to our baseline configuration. This table also indicates the p-values of pairwise Wilcoxon signed-rank tests between the solution value of each method configuration and the baseline.

These results confirm our methodological design and baseline parameter choices. Indeed, at significance level 0.05, all alternative configurations but

Table 4.5: Sensitivity Analysis

Configuration	Avg	T	p-value
Baseline	53.00	135.31	–
2OPT-SWAP-RELOCATE	53.01	145.42	0.52
SWAP-2OPT-RELOCATE	53.06	178.56	1.31×10^{-5}
SWAP-RELOCATE-2OPT	53.06	183.41	6.17×10^{-7}
RELOCATE-2OPT-SWAP	53.06	160.73	8.75×10^{-7}
RELOCATE-SWAP-2OPT	53.08	183.61	1.91×10^{-6}
without 2OPT	53.61	130.95	7.85×10^{-15}
without SWAP	53.02	118.97	0.02
without RELOCATE	53.11	105.70	2.00×10^{-11}
without 2OPT and SWAP	53.80	74.93	5.36×10^{-15}
without 2OPT and RELOCATE	53.66	71.03	7.85×10^{-15}
without RELOCATE and SWAP	53.17	74.02	6.65×10^{-12}
without diversity management	53.16	111.63	9.16×10^{-11}
without secondary objective	53.16	87.20	1.31×10^{-11}

one obtained solutions of significantly worse quality than the baseline for only moderate time gains. The remaining configuration (SWAP-2OPT-RELOCATE) had a similar performance but a longer CPU time. Based on these results, 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 as well as the diversity management strategy also contributes to attain solutions of significantly higher 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 4.1 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.

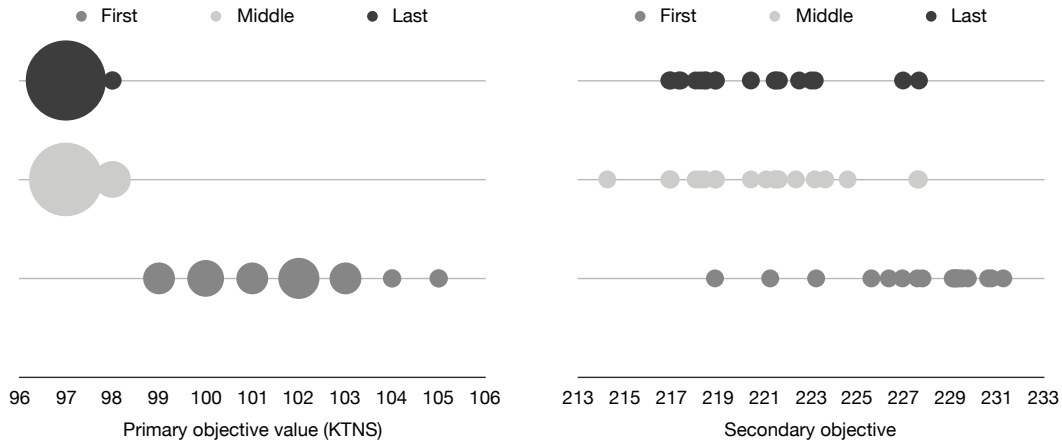


Figure 4.1: Distribution of objective values in the population after first survivors selection, halfway through the execution and after last survivors selection.

4.4

Comparison with other algorithms on classical benchmark instances

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. (2016), the ILS of Paiva and Carvalho (2017) and the DQGA of Ahmadi et al. (2018). 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. (2016): average of 20 runs on a single thread of an Intel i7-2600 3.4 GHz with 16 GB of RAM;
- Paiva and Carvalho (2017): average of 20 runs on a single thread of Intel i5-3330 3.2 GHz with 8 GB of RAM;
- Ahmadi et al. (2018): average of 10 runs on an Intel i7 3.4 GHz with 16 GB of RAM.

The processors used in CS+BRKGA (Chaves et al., 2016) and ILS (Paiva and Carvalho, 2017) experiments are from a similar generation as ours, with a similar speed, such that the *magnitude* of the CPU times reported by

these authors remain comparable. The time measurements of DQGA (Ahmadi et al., 2018), however, are not directly comparable with the other works, since only the time taken to *attain* the final solutions in each run was counted (i.e., excluding the remaining search time until termination), and a different programming language was used (MATLAB instead of C++).

Table 4.6 presents a summary of the results. Each line reports average values for one instance group. HGS-SSP found the solutions of best quality for all groups, in a computational time which is equal or 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 4.6: Summary of the comparative analyses

Group	CS+BRKGA			ILS			DQGA			HGS-SSP		
	Best	Avg	T	Best	Avg	T	Best	Avg	T	Best	Avg	T
<i>A</i>	12.63	12.63	3.71	12.63	12.63	0.09	-	-	-	12.63	12.63	0.06
<i>B</i>	13.32	13.32	4.05	13.32	13.32	0.16	-	-	-	13.32	13.32	0.10
<i>C</i>	17.46	17.62	9.83	17.46	17.46	1.45	-	-	-	17.46	17.46	0.91
<i>D</i>	13.40	13.68	27.66	13.38	13.39	5.22	-	-	-	13.38	13.38	4.28
<i>E</i>	9.64	9.70	6.54	9.64	9.64	0.40	-	-	-	9.64	9.64	0.43
<i>C</i> ₁	5.68	5.68	2.42	5.68	5.68	0.07	5.68	5.68	5.68	5.68	5.68	0.07
<i>C</i> ₂	13.00	13.07	11.58	13.00	13.01	0.85	13.00	13.03	13.03	13.00	13.00	0.78
<i>C</i> ₃	60.58	61.71	123.15	60.25	60.54	134.78	60.15	60.99	60.99	60.08	60.17	67.45
<i>C</i> ₄	135.03	137.21	541.10	133.70	134.19	901.81	133.63	136.28	136.28	132.78	132.99	472.55
<i>datA</i>	-	-	-	5.35	5.35	0.07	5.35	5.35	5.35	5.35	5.35	0.06
<i>datB</i>	-	-	-	12.78	12.78	1.08	12.78	12.86	12.86	12.78	12.78	0.74
<i>datC</i>	-	-	-	55.53	55.79	136.64	55.53	56.62	56.62	55.43	55.48	63.77
<i>datD</i>	-	-	-	134.00	134.38	945.89	133.98	136.19	136.19	132.85	133.11	472.33

Tables 4.7 to 4.13 now compare the results of HGS-SSP 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.

Table 4.7: Performance comparison on Group *A* instances

<i>n</i>	<i>m</i>	<i>C</i>	#	CS+BRKGA			ILS			HGS-SSP		
				Best	Avg	T	Best	Avg	T	Best	Avg	T
8	15	5	10	12.00	12.00	2.39	12.00	12.00	0.04	12.00	12.00	0.04
8	15	10	30	6.83	6.83	2.65	6.83	6.83	0.07	6.83	6.83	0.04
8	20	5	10	16.80	16.80	2.97	16.80	16.80	0.06	16.80	16.80	0.06
8	20	10	30	13.07	13.07	3.54	13.07	13.07	0.13	13.07	13.07	0.06
8	20	15	60	7.08	7.08	3.57	7.08	7.08	0.10	7.08	7.08	0.05
8	25	5	10	20.10	20.10	4.54	20.10	20.10	0.03	20.10	20.10	0.07
8	25	10	30	18.20	18.20	4.37	18.20	18.20	0.12	18.20	18.20	0.08
8	25	15	60	12.95	12.95	4.61	12.95	12.95	0.14	12.95	12.95	0.08
8	25	20	100	6.61	6.61	4.72	6.61	6.61	0.11	6.61	6.61	0.06

Table 4.8: Performance comparison on Group *B* instances

<i>n</i>	<i>m</i>	<i>C</i>	#	CS+BRKGA			ILS			HGS-SSP		
				Best	Avg	T	Best	Avg	T	Best	Avg	T
9	15	5	10	12.20	12.20	2.72	12.20	12.20	0.07	12.20	12.20	0.07
9	15	10	30	7.37	7.37	3.15	7.37	7.37	0.11	7.37	7.37	0.07
9	20	5	10	17.40	17.40	3.13	17.40	17.40	0.08	17.40	17.40	0.09
9	20	10	30	14.17	14.17	3.92	14.17	14.17	0.19	14.17	14.17	0.10
9	20	15	60	7.60	7.60	3.99	7.60	7.60	0.18	7.60	7.60	0.08
9	25	5	10	20.40	20.40	4.08	20.40	20.40	0.05	20.40	20.40	0.11
9	25	10	30	18.77	18.77	5.08	18.77	18.77	0.21	18.77	18.77	0.13
9	25	15	50	14.74	14.74	5.10	14.74	14.74	0.33	14.74	14.74	0.13
9	25	20	100	7.19	7.19	5.26	7.19	7.19	0.20	7.19	7.19	0.10

Table 4.9: Performance comparison on Group *C* instances

<i>n</i>	<i>m</i>	<i>C</i>	#	CS+BRKGA			ILS			HGS-SSP		
				Best	Avg	T	Best	Avg	T	Best	Avg	T
15	15	5	10	16.60	16.69	5.31	16.60	16.60	0.53	16.60	16.60	0.71
15	15	10	30	9.80	9.88	7.10	9.80	9.80	0.89	9.80	9.80	0.57
15	20	5	10	20.60	20.77	7.26	20.60	20.60	0.67	20.60	20.60	0.83
15	20	10	30	18.33	18.52	8.93	18.33	18.33	1.59	18.33	18.33	0.96
15	20	15	60	10.52	10.65	9.61	10.52	10.52	1.71	10.52	10.52	0.76
15	25	5	10	27.50	27.7	9.30	27.50	27.50	0.65	27.50	27.51	1.02
15	25	10	30	25.07	25.30	13.52	25.07	25.07	1.92	25.07	25.07	1.28
15	25	15	60	19.07	19.27	13.63	19.07	19.07	3.02	19.07	19.07	1.18
15	25	20	100	9.66	9.79	13.82	9.66	9.66	2.06	9.66	9.66	0.90

Table 4.10: Performance comparison on Group D instances

n	m	C	#	CS+BRKGA			ILS			HGS-SSP		
				Best	Avg	T	Best	Avg	T	Best	Avg	T
20	15	5	10	21.10	21.58	10.78	20.90	20.90	1.39	20.90	20.90	2.59
20	15	10	20	8.20	8.44	12.34	8.20	8.21	2.20	8.20	8.20	1.83
20	20	5	10	24.30	24.93	14.84	24.20	24.24	1.85	24.20	24.20	3.23
20	20	10	10	10.60	10.76	16.08	10.60	10.60	2.88	10.60	10.60	2.52
20	20	15	30	6.67	6.79	24.66	6.67	6.67	3.32	6.67	6.67	2.27
20	25	5	10	30.10	30.74	19.16	30.10	30.10	2.10	30.10	30.11	3.99
20	25	10	10	15.40	15.47	21.49	15.40	15.40	3.73	15.40	15.40	3.37
20	25	15	40	21.25	21.75	28.11	21.25	21.26	7.41	21.25	21.25	3.88
20	25	20	40	6.15	6.28	35.53	6.15	6.15	3.45	6.15	6.15	2.69
25	15	10	10	5.90	6.00	21.14	5.90	5.90	3.97	5.90	5.90	3.78
25	20	10	10	11.60	12.05	27.48	11.60	11.61	9.73	11.60	11.60	6.58
25	20	15	10	7.60	7.82	25.66	7.60	7.60	9.93	7.60	7.60	6.74
25	25	10	10	16.60	17.06	36.88	16.60	16.67	11.68	16.60	16.67	8.60
25	25	15	10	10.00	10.00	54.70	10.00	10.00	7.83	10.00	10.00	6.21
25	25	20	30	5.50	5.59	66.10	5.50	5.50	6.89	5.50	5.50	5.97

Table 4.11: Performance comparison on Group E instances

n	m	C	#	CS+BRKGA			ILS			HGS-SSP		
				Best	Avg	T	Best	Avg	T	Best	Avg	T
10	10	4	10	9.50	9.50	1.55	9.50	9.50	0.08	9.50	9.50	0.07
10	10	5	10	6.20	6.21	1.96	6.20	6.20	0.08	6.20	6.20	0.06
10	10	6	10	4.30	4.30	2.88	4.30	4.30	0.05	4.30	4.30	0.06
10	10	7	10	3.00	3.00	3.45	3.00	3.00	0.03	3.00	3.00	0.06
15	20	6	10	21.40	21.71	7.09	21.40	21.40	0.77	21.40	21.40	0.92
15	20	8	10	14.20	14.33	7.68	14.20	14.20	0.90	14.20	14.21	0.84
15	20	10	10	10.30	10.34	12.71	10.30	10.30	0.69	10.30	10.30	0.72
15	20	12	10	8.20	8.20	14.97	8.20	8.20	0.62	8.20	8.20	0.69

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 obtains 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 finds 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 4.12 and 4.13 now compare the performance of existing methods on the larger instances of Crama et al. (1994) and Catanzaro et al. (2015). On these instances, very significant differences of performance can be noticed.

For the first two groups of each table (C_1 and C_2 from Table 4.12 and

Table 4.12: Performance comparison on Groups C_1 , C_2 , C_3 , and C_4

n	m	C	#	ILS			DQGA		HGS-SSP		
				Best	Avg	T	Best	Avg	Best	Avg	T
10	10	4	10	9.10	9.10	0.08	9.10	9.10	9.10	9.10	0.08
10	10	5	10	6.20	6.20	0.08	6.20	6.20	6.20	6.20	0.07
10	10	6	10	4.30	4.30	0.06	4.30	4.30	4.30	4.30	0.06
10	10	7	10	3.10	3.10	0.04	3.10	3.10	3.10	3.10	0.06
15	20	6	10	20.60	20.63	0.84	20.60	20.70	20.60	20.60	0.93
15	20	8	10	13.70	13.70	1.09	13.70	13.70	13.70	13.70	0.82
15	20	10	10	10.10	10.10	0.87	10.10	10.10	10.10	10.10	0.73
15	20	12	10	7.60	7.60	0.60	7.60	7.60	7.60	7.60	0.64
30	40	15	10	91.40	91.70	86.81	91.30	91.88	91.10	91.10	75.85
30	40	17	10	71.30	71.59	132.01	71.20	72.08	71.20	71.20	66.94
30	40	20	10	50.40	50.71	173.47	50.40	51.36	50.20	50.37	64.40
30	40	25	10	27.90	28.14	146.81	27.70	28.64	27.80	28.02	62.61
40	60	20	10	178.40	179.00	441.66	178.40	180.92	177.20	177.41	512.09
40	60	22	10	151.50	152.04	665.13	151.30	154.00	150.50	150.67	490.88
40	60	25	10	121.00	121.52	1016.37	120.90	123.90	120.20	120.44	478.33
40	60	30	10	83.90	84.18	1484.97	83.90	86.30	83.20	83.44	408.90

Table 4.13: Performance comparison on Groups $datA$, $datB$, $datC$ and $datD$

n	m	C	#	ILS			DQGA		HGS-SSP		
				Best	Avg	T	Best	Avg	Best	Avg	T
10	10	4	10	8.50	8.50	0.09	8.50	8.50	8.50	8.50	0.07
10	10	5	10	5.80	5.80	0.09	5.80	5.80	5.80	5.80	0.06
10	10	6	10	4.10	4.10	0.05	4.10	4.10	4.10	4.10	0.06
10	10	7	10	3.00	3.00	0.04	3.00	3.00	3.00	3.00	0.05
15	20	6	10	20.50	20.50	1.09	20.50	20.72	20.50	20.50	0.88
15	20	8	10	13.70	13.70	1.38	13.70	13.74	13.70	13.70	0.79
15	20	10	10	9.70	9.70	1.12	9.70	9.76	9.70	9.70	0.69
15	20	12	10	7.20	7.20	0.71	7.20	7.20	7.20	7.20	0.60
30	40	15	10	83.90	84.09	99.09	83.80	84.82	83.50	83.50	74.71
30	40	17	10	65.50	65.82	137.08	65.50	66.74	65.40	65.43	71.77
30	40	20	10	46.60	46.78	172.59	46.60	47.82	46.60	46.66	60.58
30	40	25	10	26.30	26.47	137.79	26.20	27.10	26.20	26.32	48.01
40	60	20	10	178.00	178.36	488.66	178.00	180.16	176.50	176.71	501.21
40	60	22	10	151.60	152.05	706.13	151.50	153.68	150.30	150.45	437.85
40	60	25	10	121.20	121.68	1069.98	121.20	123.68	120.30	120.61	466.12
40	60	30	10	85.20	85.42	1518.80	85.20	87.22	84.30	84.66	484.14

datA and *datB* from Table 4.13), 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. For the last two groups of each table (C_3 and C_4 from Table 4.12 and *datC* and *datD* from Table 4.13), 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 time which is two to three times smaller. These significant improvements, especially on larger instances, show that the proposed method performs a more sustained and diversified search in the solution space. We conducted pairwise Wilcoxon signed-rank tests, comparing the average solution value of HGS-SSP on each instance group with those of CS+BRKGA (Chaves et al., 2016), ILS (Paiva and Carvalho, 2017) and DQGA (Ahmadi et al., 2018), obtaining p-values of 3.78×10^{-5} , 6.31×10^{-4} and 8.84×10^{-4} , respectively. At a significance level of 0.05, this statistical analysis rejects the null hypothesis and confirms significant improvements. Finally, Catanzaro et al. (2015) published optimal solutions for several instances with 10 jobs and 10 tools from group *datA*. For all of these instances, the proposed algorithm retrieved the optimal solutions on every run.

4.5

Experiments on larger instances

The benchmark instances of Crama et al. (1994), Yanasse et al. (2009), and Catanzaro et al. (2015) 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 (2001), 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, 2001).

These new instances are divided into three groups (F_1 , F_2 and F_3). Within each group, the instances have the same number of jobs and tools but four possible capacity levels. For each group and capacity value we generated five instances. Overall, the number of jobs ranges from 50 to 70 and the number of tools ranges from 75 to 105. Since no previous results are available for these instances, we contacted previous authors to obtain the original source codes. Based on the available codes, Table 4.14 draws a comparison of the results of HGS-SSP, CS+BRKGA (Chaves et al., 2016) and ILS (Paiva and Carvalho, 2017) on the new instances. Each line corresponds to an instance group and capacity value. All the algorithms were implemented in C++ and run on the

same processor: an Intel Gold 6148 Skylake 2.4 GHz with 256 MB of dedicated RAM. Ten runs were done for each instance. As previously, we display the average and best solution quality over these runs, as well as the average CPU time of each method.

Table 4.14: Performance comparison on Groups F_1 , F_2 and F_3

n	m	C	#	CS+BRKGA			ILS			HGS-SSP		
				Best	Avg	T	Best	Avg	T	Best	Avg	T
50	75	25	5	279.00	282.82	1692.12	270.00	272.22	1760.02	268.40	268.70	3629.91
50	75	30	5	206.20	210.02	1486.26	197.00	199.72	3901.03	196.40	197.12	3065.40
50	75	35	5	155.60	159.12	1346.31	148.80	150.00	6345.77	147.20	148.12	2495.35
50	75	40	5	117.20	119.70	1186.40	110.60	111.86	7800.12	109.80	110.48	2465.92
60	90	35	5	434.80	439.76	4366.10	420.00	423.64	5888.10	414.60	415.40	12890.92
60	90	40	5	337.40	342.16	3859.72	324.40	327.12	11529.91	320.40	321.12	10520.43
60	90	45	5	263.80	267.92	3431.59	250.00	253.46	19783.86	246.80	248.52	8904.50
60	90	50	5	204.80	209.16	2996.04	193.60	195.84	28217.05	191.60	192.74	7888.26
70	105	40	5	606.60	611.98	5584.09	588.00	591.42	11290.74	577.00	577.56	21952.61
70	105	45	5	487.80	493.76	8280.22	468.40	472.20	20453.05	459.20	459.94	19704.58
70	105	50	5	394.80	399.42	7478.64	376.80	379.36	34754.23	369.80	371.14	17430.14
70	105	55	5	319.40	325.38	6684.31	303.40	305.76	53314.82	298.20	299.78	16582.72

As visible in Table 4.14, the use of larger instances emphasizes the differences of performance. On these instances, we observe that HGS-SSP significantly outperforms previous algorithms, obtaining better average and best solution quality for all groups and capacity values. Remarkably, the average solution quality of HGS-SSP is generally better than the best solution quality of the other algorithms. The statistical significance of this difference of solution quality is confirmed by pairwise Wilcoxon signed-rank tests, with p-values of 1.97×10^{-21} and 1.96×10^{-21} for ILS and CS+BRKGA. The proposed algorithm also proves to be more robust in the sense that the standard deviation of its solution values is smaller: 0.83 in average for HGS-SSP in comparison to 1.68 and 2.76 for ILS and CS+BRKGA, respectively. Finally, the CPU time of HGS-SSP on these instances is intermediate between that of CS+BRKGA and ILS.

To eliminate the impact of CPU times in the analysis, we conducted a final experiment in which the three algorithms were run 10 times on each instance of Group F_1 (with 50 jobs and 75 tools) with a fixed termination criterion of $T = 5000$ seconds. During these runs, we measured the solution value (number of tool switches) of each algorithm at fixed instants: 50, 100, 250, 500, 750, 1000, 1500, 2500, 3750 and 5000 seconds. The average number of tool switches for each method is represented in Figure 4.2 as a function of CPU time. Moreover, the filled area represents the interval between the best and worst results over the 10 runs.

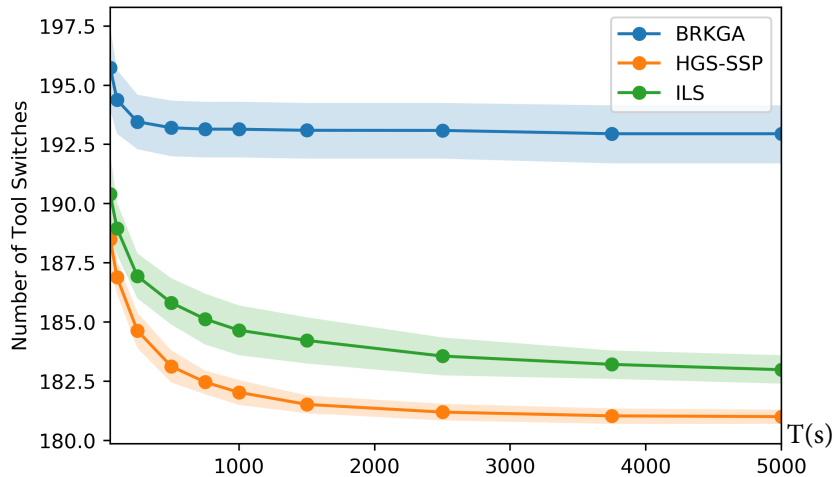


Figure 4.2: Average number of tool switches over time for the instances of Group F_1

As seen in Figure 4.2, HGS-SSP outperforms previous approaches in terms in performance and convergence behavior, since it finds solutions of better quality at every considered time instant. As such, it represents a promising approach for practical applications.

5 Conclusions

In this work, we introduced a simple and efficient metaheuristic for the well-known job sequencing and tool switching problem (SSP). The proposed HGS-SSP contrasts with previous algorithms, which had a tendency to be over-engineered and generally complex. Our algorithm finds a good balance between aggressive *intensification*, achieved by an efficient local search in the space of permutations, and *diversification* (Blum and Roli, 2003; Vidal et al., 2013), 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 algorithm could be extended and tested on other SSP variants, e.g., with multiple machines (Beezão et al., 2017) or with different objective functions (Calmels, 2019). 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 as 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.

Bibliography

- Adjiaşvili, D., Bosio, S., and Zemmer, K. (2015). Minimizing the number of switch instances on a flexible machine in polynomial time. *Operations Research Letters*, 43(3):317–322.
- Ahmadi, E., Goldengorin, B., Süer, G. A., and 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.
- Al-Fawzan, M. and Al-Sultan, K. (2003). A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & Industrial Engineering*, 44(1):35–47.
- Amaya, J. E., Cotta, C., and 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., and Sampels, M., editors, *Hybrid Metaheuristics*, pages 190–202, Berlin, Heidelberg. Springer.
- Amaya, J. E., Cotta, C., and 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.
- Amaya, J. E., Cotta, C., and Fernández-Leiva, A. J. (2011). Memetic cooperative models for the tool switching problem. *Memetic Computing*, 3(3):199–216.
- Amaya, J. E., Cotta, C., and Fernández-Leiva, A. J. (2012). Solving the tool switching problem with memetic algorithms. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 26(2):221–235.
- Amaya, J. E., Cotta, C., and Leiva, A. J. F. (2010). *Hybrid cooperation models for the tool switching problem*, pages 39–52. Springer, Berlin, Heidelberg.
- Avci, S. and Akturk, M. (1996). Tool magazine arrangement and operations sequencing on CNC machines. *Computers & Operations Research*, 23(11):1069–1081.

- Bard, J. F. (1988). A heuristic for minimizing the number of tool switches on a flexible machine. *IIE Transactions*, 20(4):382–391.
- Beezão, A. C., Cordeau, J.-F., Laporte, G., and Yanasse, H. H. (2017). Scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, 257(3):834–844.
- Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308.
- Bouzina, K. and Emmons, H. (1996). Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3-4):379–393.
- Burger, A. P., Jacobs, C. G., van Vuuren, J. H., and Visagie, S. E. (2015). Scheduling multi-colour print jobs with sequence-dependent setup times. *Journal of Scheduling*, 18(2):131–145.
- 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(15-16):5005–5025.
- Carlisle, M. and Lloyd, E. (1995). On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(93):225–235.
- Catanzaro, D., Gouveia, L., and Labbé, M. (2015). Improved integer linear programming formulations for the job sequencing and tool switching problem. *European Journal of Operational Research*, 244(3):766–777.
- Chaves, A., Lorena, L., Senne, E., and Resende, M. (2016). Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research*, 67:174–183.
- Crama, Y. (1997). Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, 99(1):136–153.
- Crama, Y., Kolen, A. W. J., Oerlemans, A. G., and Spieksma, F. C. R. (1994). Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, 6(1):33–54.
- Crama, Y. and van de Klundert, J. (1999). Worst-case performance of approximation algorithms for tool management problems. *Naval Research Logistics (NRL)*, 46(5):445–462.

- Djellab, H., Djellab, K., and Gourgand, M. (2000). A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, 64(1):165–176.
- Fathi, Y. and Barnette, K. W. (2002). Heuristic procedures for the parallel machine problem with tool switches. *International Journal of Production Research*, 40(1):151–164.
- Furrer, M. and Mütze, T. (2017). An algorithmic framework for tool switching problems with multiple objectives. *European Journal of Operational Research*, 259(3):1003–1016.
- Gendreau, M., Hertz, A., and Laporte, G. (1992). New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094.
- Ghiani, G., Grieco, A., and Guerriero, E. (2007). An exact solution to the TLP problem in an NC machine. *Robotics and Computer-Integrated Manufacturing*, 23(6):645–649.
- Ghiani, G., Grieco, A., and Guerriero, E. (2010). Solving the job sequencing and tool switching problem as a nonlinear least cost hamiltonian cycle problem. *Networks*, 55(4):379–385.
- Ghrayeb, O. A., Phojanamongkolkij, N., and Finch, P. R. (2003). A mathematical model and heuristic procedure to schedule printed circuit packs on sequencers. *International Journal of Production Research*, 41(16):3849–3860.
- Gray, A. E., Seidmann, A., and Stecke, K. E. (1993). A synthesis of decision models for tool management in automated manufacturing. *Management Science*, 39(5):549–567.
- Gribel, D. and Vidal, T. (2019). HG-means: A scalable hybrid metaheuristic for minimum sum-of-squares clustering. *Pattern Recognition*, 88:569–583.
- Hertz, A., Laporte, G., Mittaz, M., and Stecke, K. E. (1998). Heuristics for minimizing tool switches when scheduling part types on a flexible machine. *IIE Transactions*, 30(8):689–694.
- Hertz, A. and Widmer, M. (1996). An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete applied mathematics*, 65(1):319–345.

- Jäger, G. and Molitor, P. (2008). Algorithms and experimental study for the traveling salesman problem of second order. In Yang, B., Du, D.-Z., and Wang, C. A., editors, *Combinatorial Optimization and Applications*, pages 211–224, Berlin, Heidelberg. Springer Berlin Heidelberg.
- Keung, K. W., Ip, W. H., and 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(11):771–777.
- Konak, A. and 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*, pages 43–48.
- Konak, A., Kulturel-Konak, S., and Azizoglu, M. (2008). Minimizing the number of tool switching instants in flexible manufacturing systems. *International Journal of Production Economics*, 116(2):298–307.
- Laporte, G., Salazar-González, J. J., and Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE Transactions*, 36(1):37–45.
- Matzliach, B. (1998). The online tool switching problem with non-uniform tool size. *International Journal of Production Research*, 36(12):3407–3420.
- Mütze, T. (2014). Scheduling with few changes. *European Journal of Operational Research*, 236(1):37–50.
- Oliver, I., Smith, D., and Holland, J. (1987). A study of permutation crossover operators on the traveling salesman problem. In Grefenstette, J., editor, *Genetic algorithms and their applications: Proceedings of the Second International Conference*, pages 224–230.
- Paiva, G. S. and Carvalho, M. A. M. (2017). Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers & Operations Research*, 88:208–219.
- Privault, C. and Finke, G. (1995). Modelling a tool switching problem on a single NC-machine. *Journal of Intelligent Manufacturing*, 6(2):87–94.
- Raduly-Baka, C. and Nevalainen, O. S. (2015). The modular tool switching problem. *European Journal of Operational Research*, 242(1):100–106.

- Salonen, K., Raduly-Baka, C., and Nevalainen, O. S. (2006a). A note on the tool switching problem of a flexible machine. *Computers & Industrial Engineering*, 50(4):458–465.
- Salonen, K., Smed, J., Johnsson, M., and Nevalainen, O. (2006b). Grouping and sequencing PCB assembly jobs with minimum feeder setups. *Robotics and Computer-Integrated Manufacturing*, 22(4):297–305.
- Senne, E. L. F. and 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*, pages 68–72, Stevens Point, USA. World Scientific and Engineering Academy and Society.
- Shirazi, R. and Frizelle, G. D. M. (2001). Minimizing the number of tool switches on a flexible machine: An empirical study. *International Journal of Production Research*, 39(15):3547–3560.
- Tang, C. S. and Denardo, E. V. (1988a). Models arising from a flexible manufacturing machine, Part I: Minimization of the number of tool switches. *Operations Research*, 36(5):767–777.
- Tang, C. S. and Denardo, E. V. (1988b). Models arising from a flexible manufacturing machine, Part II: Minimization of the number of switching instants. *Operations Research*, 36(5):778–784.
- Toffolo, T. A., Vidal, T., and Wauters, T. (2019). Heuristics for vehicle routing problems: Sequence or set optimization? *Computers & Operations Research*, 105:118–131.
- Tzur, M. and Altman, A. (2004). Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. *IIE Transactions*, 36(2):95–110.
- Vidal, T. (2017). Node, edge, arc routing and turn penalties: Multiple problems – One neighborhood extension. *Operations Research*, 65(4):992–1010.
- Vidal, T., Battarra, M., Subramanian, A., and Erdogan, G. (2015). Hybrid metaheuristics for the clustered vehicle routing problem. *Computers & Operations Research*, 58(1):87–99.
- Vidal, T., Crainic, T., Gendreau, M., and Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, 231(1):1–21.

- Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., and Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624.
- Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2014). A unified solution framework for multi-attribute vehicle routing problems. *European Journal of Operational Research*, 234(3):658–673.
- Whitley, D. (2019). *Next generation genetic algorithms: A user's guide and tutorial*, pages 245–274. Springer International Publishing.
- Yanasse, H. H., Rodrigues, R. d. C. M., and 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.
- Zeballos, L. (2010). A constraint programming approach to tool allocation and production scheduling in flexible manufacturing systems. *Robotics and Computer-Integrated Manufacturing*, 26(6):725–743.
- Zhou, B.-H., Xi, L.-F., and 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(9):876–882.