Expert Systems With Applications 131 (2019) 148–171

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems

Maria Amélia Lopes Silva[a], Sérgio Ricardo de Souza[a,*], Marcone Jamilson Freitas Souza[b], Ana Lúcia C. Bazzan[c]

[a] Federal Center of Technological Education of Minas Gerais (CEFET-MG), Belo Horizonte ZIP Code: 30510-000, MG, Brazil
[b] Department of Computer Science, Federal University of Ouro Preto (UFOP), Ouro Preto ZIP Code: 35400-000, MG, Brazil
[c] Institute of Informatics, Federal University of Rio Grande of the Sul (UFRGS), Porto Alegre ZIP Code: 91501-970, RS, Brazil

ARTICLE INFO

Article history:
Received 6 February 2018
Revised 24 April 2019
Accepted 24 April 2019
Available online 24 April 2019

Keywords:
Multi-agent framework for optimization
Reinforcement learning
Metaheuristics
Multi-agent systems
Vehicle routing problem with time window
Unrelated parallel machine scheduling problem

ABSTRACT

This article presents a multi-agent framework for optimization using metaheuristics, called AMAM. In this proposal, each agent acts independently in the search space of a combinatorial optimization problem. Agents share information and collaborate with each other through the environment. The goal is to enable the agent to modify their actions based on experiences gained in interacting with the other agents and the environment using the concepts of Reinforcement Learning. For better introduction and validation of the AMAM framework, this article uses the instantiation of the Vehicle Routing Problem with Time Windows (VRPTW) and the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSP-ST), i.e., two classic combinatorial optimization problems. The main objective of the experiments is to evaluate the performance of the proposed adaptive agents. The experiments confirm that the ability to learn attributed to the agent directly influences the quality of solutions, both from the individual point of view and from the point of view of teamwork. In this way, the framework presented here is a step forward in relation to the other frameworks of the literature regarding to the adaptation to the particular aspects of the problems. Additionally, the cooperation between agents and their ability to influence the quality of the solutions of the agents involved in the search of the solution is confirmed. The results also strengthen the issue of the scalability of the framework, since, with the addition of new agents, there is an improvement of the solutions obtained.

1. Introduction

Metaheuristics have been consolidated as one of the main methodologies for optimization problem solving of diverse classes. This statement is justified in part by its versatility and its great adaptability to solve problems, but, on the other hand, mainly due to the possibility of obtaining, in limited computational time, good quality solutions for large and complex optimization problems. Problems having these characteristics in many and important cases do not have their solution known, in limited computational time, if only mathematical programming techniques are applied. This assertion is particularly true for the class of NP-Hard problems, which involves problems of great theoretical and practical relevance, such as the Vehicle Routing Problem or the Parallel Machine Scheduling Problem.

Blum and Roli (2003), Blum, Puchinger, Raidl, and Roli (2011), Talbi (2009) and Gendreau and Potvin (2010) present general and conceptual formulations about metaheuristics as well as important reviews regarding their use in solving optimization problems. In recent years, the combination of two or more metaheuristics for solving optimization problems has been growing (Blum et al., 2011; Cotta, Talbi, & Alba, 2005). The main objective of the hybridization of metaheuristics is to apply together the best features of each metaheuristic to solve a problem, allowing, besides getting the best solution quality in a shorter time, to increase the ability to tackle more complex problems. Cotta et al. (2005) and Blum et al. (2011) claim that hybridizations are responsible for many of the best results found in the literature for various classes of optimization problems, which justifies the rising interest for this approach.

The increase in the use of metaheuristics, whether or not by specialists in methods to solve optimization problems, has guided

* Corresponding author.
E-mail addresses: mamelia@ufv.br (M.A. Lopes Silva), sergio@dppg.cefetmg.br (S.R. de Souza), marcone@iceb.ufop.br (M.J. Freitas Souza), bazzan@inf.ufrgs.br (A.L.C. Bazzan).

https://doi.org/10.1016/j.eswa.2019.04.056
0957-4174/

researchers in the development of frameworks. These experts are looking for tools that can facilitate solving these problems, as well as offering features that enhance the performance of the solution process. At the same time, in the case of researchers, the search for new techniques to solve this class of problems is a fundamental challenge posed, but also is directed towards the development of these software tools.

Frameworks offer flexibility in incorporating new methods for the solution, without requiring efforts to remake the application, and, therefore, allowing the determination of better solutions with low development costs. Frameworks provide a structure of generic features for the solution of problems from a specific domain, making the development of new applications in this domain much simpler. A Metaheuristic Optimization Frameworks (MOF), as named in Parejo, Ruiz-Cortés, Lozano, and Fernandez (2012), is a software tool that provides implementations of a set of metaheuristics, via reusable codes, facilitating the development of applications for solving optimization problems.

Several frameworks for metaheuristics can be found in the literature (Alba, Luque, Garcia-Nieto, Ordonez, & Leguizamon, 2007; Cahon, Melab, & Talbi, 2004; Coelho et al., 2011; Durillo & Nebro, 2011; Fink & Voß, 2002; Gaspero & Schaerf, 2003; Talukdar, Baerentzen, Gove, & Souza, 1998), most of them with similar characteristics and proposals. A bibliographical review and a comparative study with major available frameworks can be found in Parejo et al. (2012) and Silva, de Souza, Souza, and de França Filho (2018). OptFrame is a framework proposed in Coelho et al. (2011). Its main characteristic is an interface for common elements of population-based metaheuristics and of trajectory-based metaheuristics. jMetal is a object-oriented framework based on the Java language (Durillo & Nebro, 2011). It includes a significant number of classic and modern methods for multi-objective optimization problems and a wide range of issues instances. ParadisEO is a global framework composed by 4 connected modules (Cahon et al., 2004; Liefooghe, Jourdan, & Talbi, 2011; Melab, Luong, Boufaras, & Talbi, 2013). These modules treat population and trajectory metaheuristics, multi-objective evolutionary techniques as well as parallel and distributed implementations. Likewise, the demand for increasingly adaptive and intelligent software has led to the incorporation of new technologies in the treatment of several problems, such as, for example, technology based on autonomous agents through multi-agent systems. The agent-based approach is distinguished by its power to model problems such as combinatorial optimization problems of a distributed nature and to express through the system entities the complexity of the relationships involved. As a consequence, the multi-agent approach has been applied in the solution via metaheuristics of various combinatorial optimization problems (Silva et al., 2018).

The current article presents a multi-agent framework called AMAM, i.e., "*Arquitetura Multiagente para Metaheurística*", in Portuguese, or Multi-agent Architecture for Metaheuristics, proposed in Silva (2007). This framework allows the easy hybridization of metaheuristics for solving combinatorial optimization problems, through a multi-agent structure. In this way, it is a generic and flexible structure, in which each metaheuristic is defined as an autonomous agent that interacts with its environment cooperatively.

Like other frameworks available in the literature, AMAM presents common features such as: (i) metaheuristics are pre-implemented to test and reuse; (ii) support the evaluation and comparison of different methods; (iii) ease in the development of a particular metaheuristic and their suitability to the treated problem. In addition to these features, the framework presented here has the strength of hybridization of metaheuristics through the parallel cooperative approach managed by Multi-Agent Systems (MAS). MAS are used here as a liaison between different metaheuristics for solving optimization problems. Each agent is responsible for performing its own task and, at same time, for using the solutions provided by other agents to improve their own solutions. In this approach, agents interact and work together to achieve a pre-defined objective. The interaction between the various agents occurs through a solution pool. The communication is governed by the rules of access to the pool, both for writing and for reading, aimed at ensuring diversity in the sharing of search information. Several previous works (Fernandes, de Souza, Silva, Borges, & Ribeiro, 2009; Silva, de Souza, de Oliveira, & Souza, 2014; Silva, de Souza, Souza, & de Oliveira, 2015) have presented different stages of the development of this framework.

This article also proposes the improvement of the self-adaptive capabilities of the framework agents, an issue discussed in Silva et al. (2015). The objective is to allow the agent to modify its actions based on the experiences acquired in the interaction with the other agents and with the environment. This experience is obtained using the concepts of Reinforcement Learning (RL) (Sutton & Barto, 1998), more specifically, through the Q-Learning algorithm (Watkins & Dayan, 1992). Therefore, here the concept of Reinforcement Learning is used to define the application order of neighborhood structures of local search. Since the neighborhood functions are specific parameters of the problem, the use of reinforcement learning in the selection of the most appropriate neighborhood to be used enables the framework to adapt better to the characteristics of the problem. The learning strategy is defined here for each agent individually.

The main motivation of this article is to present the AMAM framework as a consolidate software tool for solving combinatorial optimization problems using metaheuristics, including important characteristics as the autonomy of the agents, without any kind of explicit coordination between them. At the same time, this framework introduces, to the best of our knowledge, the first use of reinforcement learning for frameworks specialized in solving combinatorial optimization problems. This built-in adaptive capacity allows the agents to adjust to specific problems, providing the best performance of these in the framework.

In order to evaluate the learning performance, two computational experiments are conducted: (i) new tests using the learning proposal presented in Silva et al. (2015) and (ii) tests carried out with the learning structure proposed in this article. These two experiments are carried out under the same computational conditions and with the same cooperation structure among the agents, allowing a effective comparison between the found results. One of the main objectives of this article is to evaluate if the form of learning, embedded in the agent, has a direct influence on the performance of the framework with respect to the quality of the obtained results, both in terms of the individual point of view and in terms of the teamwork point of view.

For better introduction and validation of the AMAM framework, this article uses the instantiation of two classic and well-known combinatorial optimization problems. The first one is the Vehicle Routing Problem with Time Window – VRPTW (Toth & Vigo, 2002) and the second one is the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times – UPMSP-ST (Allahverdi, 2015; Allahverdi, Ng, Cheng, & Kovalyov, 2008). Both VRPTW and UPMSP-ST are NP-Hard problems and, in consequence, suitable to be used here for demonstrating the potentialities of the AMAM framework.

In summary, the main contributions of this proposal are:

(i) Improve the self-adaptive skills of the agent, using the concepts of Reinforcement Learning, specifically the Q-Learning algorithm, allowing the agent to better adapt to the specific parameters of the problem to be addressed by the framework;

(ii) Improve cooperation among agents, through the inclusion of criteria for insertion of new solutions into the cooperative structure, seeking greater diversity of solutions;
(iii) Demonstrate how the enhancement of individual agent skills directly influences the cooperative performance of agents.

The remainder of this paper is organized as follows. Section 2 presents a review about works on the use of learning in conjunction with metaheuristics within frameworks. Section 3 briefly reviews the Vehicle Routing Problem with Time-Windows (VRPTW) and the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSP-ST). Section 4 describes the AMAM framework and its main components. Section 5 shows the basic concepts of reinforcement learning and describes the proposed adaptive agent. Section 6 reports experiments carried out on the VRPTW instances and the UPMSP-ST instances using the AMAM framework. Finally, the last section present some conclusions and discusses future directions of research.

2. Bibliographical review

The use of Reinforcement Learning in conjunction with metaheuristics has been a subject of strong interest in the search for better and more efficient methods for solving optimization problems, as shown in, for example, Gambardella and Dorigo (1995), Meignan, Créput, and Koukam (2008), Barbucha, Czarnowski, Jędrzejowicz, Ratajczak-Ropel, and Wierzbowska (2010), Queiroz dos Santos, de Melo, Neto, and Aloise (2014), Lotfi and Acan (2015), Martin et al. (2016), Samma, Lim, and Saleh (2016) and Silva et al. (2015). Some of these articles are discussed in the following.

Gambardella and Dorigo (1995) introduce the Ant-Q algorithm, the first application of Q-learning algorithm for solving combinatorial optimization problems, according to the authors knowledge. The experiments performed use instances of the Traveling Salesman Problem (TSP). As posed in Dorigo, Caro, and Gambardella (1999), Ant-Q is an algorithm that tries to merge Ant System (AS) algorithm and Q-learning properties. In its turn, the AS algorithm is the initial proposal of the Ant Colony Optimization metaheuristic (Dorigo & Stützle, 2019). At Ant-Q algorithm, an Ant-Q-value table (*AQ (r, s)*, where *(r, s)* is a pair of cities of the TSP) is defined, associated with *Q*-values of Q-learning algorithm. It indicates how useful it will be to move in the edge *(r, s)*, this table being updated at run time. The objective is that the agents of the Ant-Q algorithms cooperate to learn AQ-values, thus seeking good solutions for the TSP. However, as indicated in Dorigo et al. (1999) and Dorigo and Stützle (2019), besides having good performance, due *to some aspects of Ant-Q, in particular the pheromone update rule, could be strongly simplified without affecting performance, Ant-Q was abandoned in favor of the simpler and equally good ACS* (Ant Colony System), introduced in Dorigo and Gambardella (1997).

Meignan et al. (2008) propose the Agent Metaheuristic Framework (AMF). AMF is based on an organizational model that describes a metaheuristic in terms of roles. Roles represent the main elements that compose metaheuristics, such as intensification, diversification, memory and adaptation or self-adaptation. According to the authors, "*to obtain a metaheuristic from the AMF organizational model, it is necessary to refine the different roles and determine the multiagent structure of the optimization system*". A metaheuristic called Coalition-Based Metaheuristic (CBM) and its application to the Vehicle Routing Problem are presented for illustrating the use of the AMF model. CBM is a metaheuristic based on the metaphor of coalitions. According to the authors, "*the coalition is composed of several agents which have the capacity to individually treat the optimization problem but cooperate to coordinate and improve the search*". An adaptive strategy is used by the CBM agent, through a learning mechanism, to select the most appropriate operator, based on the context. The reward is given to the agent only when it finds better solutions than those already found previously. According to the authors, the results show that the ability to learn improves the quality of solutions, especially when the number of agents involved increases.

Barbucha et al. (2010) presented JADE-based A-Team environment (JABAT), a tool for building A-Teams (Talukdar & Souza, 1990) architectures to solve different optimization problems. This framework produces solutions for combinatorial optimization problems using a set of optimization agents, each one implementing a solution algorithm. Barbucha et al. (2010) also proposed an extension of this framework, named Cooperative JADE-based ATeam (Cooperative JABAT), introducing a Reinforcement Learning mechanism. In this case, the reward is assigned to the optimization agents each time is found a better solution than the one previously. A negative reward can also be awarded if the agent fails to improve a solution. The weight of each agent in that context is considered when solutions are requested in the common memory. Thus, the required solutions are not sent immediately after the request of each agent, but only after all the agents perform these requests and, besides, considering the weights associated to each of them.

Queiroz dos Santos et al. (2014) show an implementation that proposes a hybridization of the reactive search with the Variable Neighborhood Search (VNS) metaheuristic (Mladenović & Hansen, 1997). The selection of the local search heuristic to be used at a certain point in the search is done in a self-adaptive learning through Reinforcement Learning. In this sense, Queiroz dos Santos et al. (2014) use the algorithm of Reinforcement Learning Q-Learning in two ways:

(i) Construction of the initial solution: instead of the construction usually used in the VNS metaheuristic, the initial solution construction uses the Q-learning algorithm, carried out from the knowledge of the environment (solutions of the problem search space);
(ii) Selection of Local Search to be applied: in order to efficiently explore the choice of which local search is best suited to a particular search point, the algorithm selects the next local search to be used based on the Q-learning algorithm.

The algorithm presented by the authors was applied to the Symmetric Traveling Salesman Problem (TSP), using the available instances in the TSPLIB repository (http://ftp.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html). Two comparisons are presented: (i) their algorithm compared with a version of VNS metaheuristics combined with the construction of initial solutions that use Q-learning, and (ii) their algorithm compared with the results already known in the literature. They conclude that their algorithm is competitive and that it performs better than the VNS version with Q-learning.

Lotfi and Acan (2015) presented the Learning-Based Multi-Agent System (LBMAS) for solving combinatorial optimization problems. This system allows collaboration between metaheuristic agents in a population of common solutions and in a two-stage file, also common to the agents. In this way, different regions of the search space are exploited using the most effective agent at the moment. At each iteration of the search, the metaheuristic to be used in the next iteration is chosen. This selection is carried out using the concepts of roulette, in which the evaluation values of metaheuristics are obtained based on the improvement levels of the objective function of each of the metaheuristics. All metaheuristics initially have the same probability of being chosen and this probability changes (grows or decreases) according to the individual performance of the agents. The agents cooperate by sharing their

individual experiences through a two-stage external memory archive: the first stage stores promising solutions based on their evaluation value; and the second stage maintains the solutions according to their spatial distribution, based on a defined dissimilarity measure. In the proposal of Lotfi and Acan (2015), the multi-agent system is experimentally evaluated using instances of the Multiprocessor Scheduling Problem. The results showed that the LBMAS is competitive in relation to the other algorithms already proposed for the problem.

Martin et al. (2016) present a distributed agent-based framework called Multi-Agent Cooperative Search (MACS). This framework is implemented using the JADE platform (Bellifemine, Poggi, & Rimassa, 2007). In this proposal, each agent performs a different combination of local search metaheuristics/heuristics and adapts continuously throughout the search process using a cooperation protocol. This cooperation protocol allows communication between the agents involved in the search for a solution. An iteration of the communication protocol is called a conversation. Communication between agents takes place as follows: good patterns that improve solutions are identified, according to the frequency that they occur in the conversation, and are then shared among the agents. In order to do this, each metaheuristic agent breaks the current solution into edge objects and sends them to the launcher agent, which brings together all the edge objects and punctuates them according to their frequency. These elements are shared with the other agents and will be part of the next solutions. The implemented learning mechanism allows each agent to maintain a short-term memory of good edge objects, which are used at the beginning of each conversation to influence how new solutions are constructed. The edges identified by the learning are used to reorder the lists of elements to be inserted into the solutions. The constructive heuristics RandNEH and RandCWS, proposed in Juan, Faulin, Grasman, Rabe, and Figueira (2015), are used for the construction of new solutions. Martin et al. (2016) show the evaluation of MACS through two different optimization problems: (i) Permutation Flow-shop Scheduling and (ii) Capacitated Vehicle Routing. Tests were carried out using 5 scenarios for each of the two problems mentioned: (i) Single agent; (ii) 4 agents; (iii) 8 agents; (iv) 12 agents; and (v) 16 agents. The results show that, with a confidence level of 95%, scenarios with eight or more agents perform better than the scenario with an isolated agent. In addition, the results also show that in some cases the performance is better when the number of agents is doubled.

Samma et al. (2016) present a new algorithm based on Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 1995), called Reinforcement Learning-based Memetic Particle Swarm Optimization (RLMPSO). Motivated by the difficulty of integrating the PSO method and local search heuristics, the authors propose the use of Reinforcement Learning in the control of the operations applied to the swarm particles. In the definition of RLMPSO, Reinforcement Learning is implemented using the Q-learning algorithm. Each particle is subject to five operations: exploration, convergence, high-jump, low-jump, and fine-tuning. Three steps, described below, present the control of Q-learning under the possible operations:

(i) Obtain the best operation to be performed for each current particle;
(ii) Perform the selected operation and evaluate the value of the fitness function obtained. The immediate reward will be calculated according to the evaluation of the fitness function; if a better value is found, the reward is positive (1) and, otherwise, a negative reward is assigned (−1);
(iii) Update the Q-table for each current particle.

According to the authors, "*the effectiveness of RLMPSO has been evaluated using four unimodal and multi-modal benchmark problems, six composite benchmark problems, five shifted and rotated benchmark problems, as well as two real-world design problems*". The results obtained in the experiments performed with these problems show that the RLMPSO exceeds a significant number of PSO variants reported in the literature. Additionally, reinforcement learning involving agents has become a relevant topic nowadays, and a large number of publications address this issue from different points of view. It is worth highlighting (Noel & Pandian, 2014; Radac & Precup, 2018; Radac, Precup, & Roman, 2018), which deal with reinforcement learning involving artificial neural networks for continuous models; (Kazemitabar, Taghizadeh, & Beigy, 2018), which discuss about hierarchical reinforcement learning in large and complex systems; and (Salgado & Clempner, 2018), which study the interaction of agents through emotion and stimuli.

It should be emphasized that none of the proposals discussed above in this literature review uses reinforcement learning among its optimization problem solving structures.

3. Case study

This section addresses the instantiation of AMAM for two problems of great importance in Combinatorial Optimization. The purpose of this instantiation is to analyze this framework, evaluate their performance and show their potential. The problems considered here for case study purposes are the Vehicle Routing Problem with Time Windows (VRPTW) and the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSP-ST).

VRPTW is one of the most studied combinatorial optimization problems. It is a generalization of the classic Travelling Salesman Problem (TSP) (Applegate, Bixby, Chvátal, & Cook, 2007), in which, unlike this, there is a fleet of vehicles, a set of customers, geographically dispersed, to be satisfied, with their respective demands and the horizons of time for the attendance. The full description of this problem is at Section 3.1, under the terms of interest of this current article. For further details regarding VRPTW, excellent reviews of the research involving this problem can be found in Toth and Vigo (2002) and Toth and Vigo (2014).

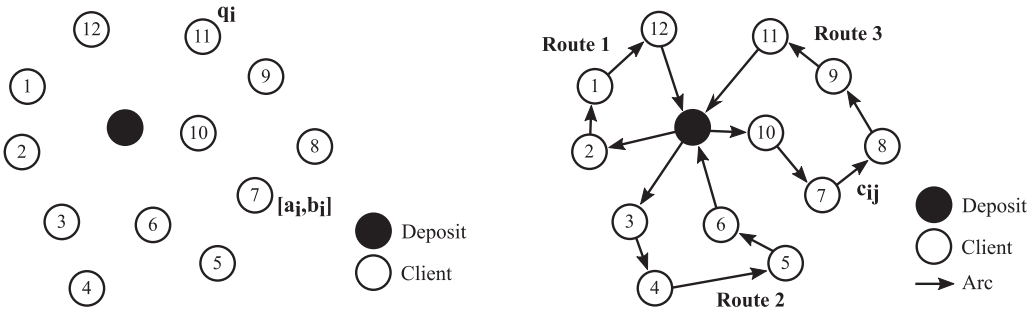
UPMSP-ST also has strong economic relevance in several types of industries. Like VRPTW, UPMSP-ST belongs to the class of NP-Hard problems, which justifies the use of metaheuristic methods in the solution of these problems. A detailed description of the UPMSP-ST can be found in Rabadi, Moraga, and Al-Salem (2006) and Vallada and Ruiz (2011), and good surveys concerning to this problem are (Allahverdi, 2015; Allahverdi et al., 2008). Section 3.2 presents this problem, in the terms that interest this current article.

3.1. Case 1: VRPTW

3.1.1. VRPTW basic definitions

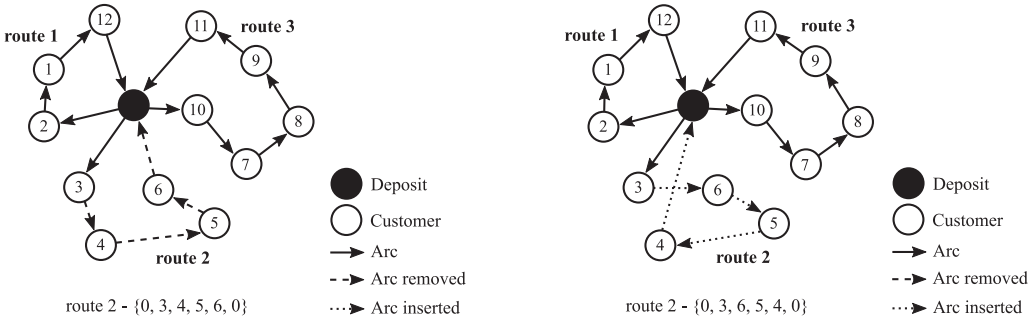
In this problem, a set $K = \{k : k = 1, 2, \ldots, |K|\}$ of vehicles is located at a single depot and must serve a set $C = \{i : i = 1, 2, \ldots, N\}$ of geographically spread customers. In the case considered here, the fleet of vehicles is homogeneous, i.e, all vehicles are equal and have the same capacity Q. Each customer i has a given demand q_i and must be served within a specified time window $[a_i, b_i]$ (see Fig. 1(a)). A solution for the VRPTW is a set of routes (see Fig. 1(b)), in which each route is represented by an ordered list of customers that determine the sequence in which they are to be served by one vehicle. The arcs show the connection between customers and have an associated value c_{ij}, which represents the travel cost between customer i and customer j. The solution x shown in Fig. 1(b) can be described as:

$$x = [0, 2, 1, 12, 0, 3, 4, 5, 6, 0, 10, 7, 8, 9, 11, 0]$$



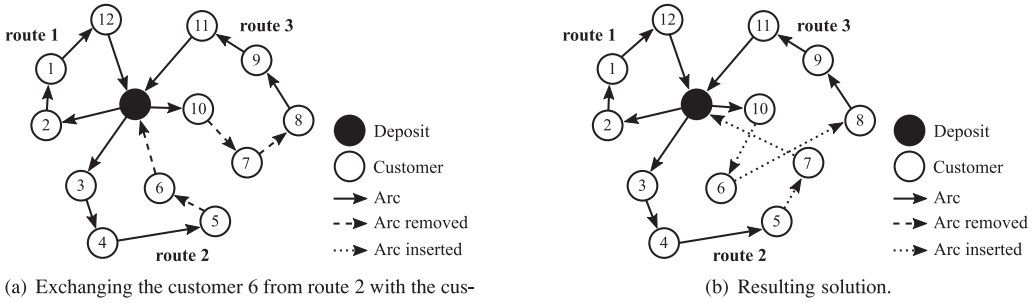
(a) Graph representing the elements of VRPTW. (b) Representation of a VRPTW solution.

Fig. 1. A VRPTW solution.



(a) Exchanging customers 4 and 6. (b) Resulting solution.

Fig. 2. An application of the Intra-Route Swap neighborhood function on 2 routes of a solution.



(a) Exchanging the customer 6 from route 2 with the customer 7 from route 3. (b) Resulting solution.

Fig. 3. An application of the Inter-Route Swap neighborhood function in a solution.

where the index 0 indicates the depot and the three routes of this solution are $route_1 = [0, 2, 1, 12, 0]$, $route_2 = [0, 3, 4, 5, 6, 0]$ and $route_3 = [0, 10, 7, 8, 9, 11, 0]$. Thus, the solution x is also described as $x = [route_1, route_2, route_3]$.

The objective of the VRPTW is to determine a set of routes in order to minimize the involved total cost with this operation. Each route is associated with a single vehicle. The routes must start and end in the depot. In our case, the cost of a solution x is calculated according to:

$$f(x) = \omega K(x) + \sum_{(i,j)\in E} c_{ij} \tag{1}$$

where:

- c_{ij} : cost between customers (i, j), which can be related to the distance between customers;
- E : set of arcs belonging to the solution x;
- $K(x)$: number of vehicles in the solution x;
- ω : an arbitrary large non-negative penalty factor.

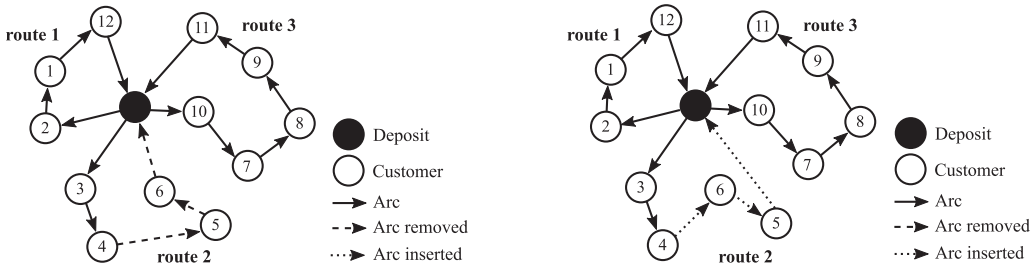
In this function, the first priority is to minimize the number of vehicles (or routes, in consequence). In case of a tie in the number of vehicles, the total distance traveled should be minimized.

3.1.2. VRPTW neighborhoods

A neighborhood is a function $\mathcal{N}(x)$ that describes a solution subset associated with the solution x belonging to the solution space of the problem. Each solution of this subset is called a neighbor. The function $\mathcal{N}(x)$ is defined as an operator that receives a solution x^1 and transforms it into another solution x^2, belonging to the neighborhood of x^1 (Milano & Roli, 2004).

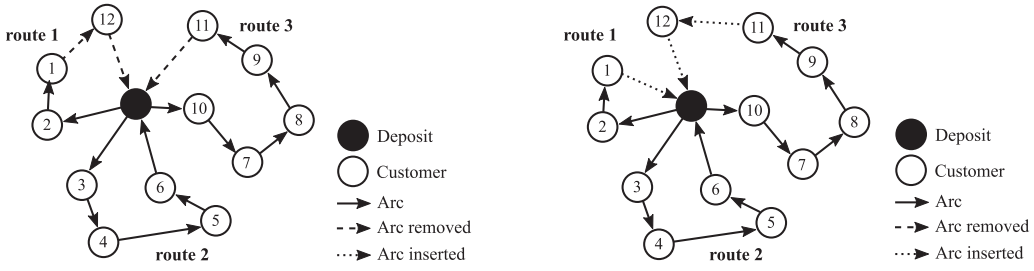
In order to explore the solution space, eight different neighborhood functions are used in the AMAM instantiation for solving VRPTW. The knowledge of these neighborhood structures becomes necessary because they make up the set of states defined in the learning model that will be described in Section 5. These structures are presented below:

(i) Intra-Route Swap: neighborhood function that performs the exchange move of a customer with another customer of the same



(a) Reallocating customer 6 in the same route. (b) Resulting solution.

Fig. 4. An application of the Intra-Route Shift neighborhood function in the route 2 of a solution.



(a) Customer 12 from route 1 is reallocated to route 3. (b) Resulting solution.

Fig. 5. An application of the Inter-Route Shift neighborhood function in a solution.

Table 1
Processing time.

	1	2	3	4	5	6	7	8
m_1	67	3	29	85	11	36	25	12
m_2	43	40	29	26	46	49	7	5

route. Fig. 2 illustrates this neighborhood function. In this example, customers 4 and 6 of route 2 are swapped;

(ii) Inter-Route Swap: neighborhood function that performs the exchange move of a customer of a route with a customer of another route. Fig. 3 shows the Inter-Route Swap neighborhood function. In this figure, customer 6 is removed from route 2 and inserted into route 3 in place of the customer 7, which is consequently transferred to the route 2 in the place previously occupied by customer 6;

(iii) Intra-Route Shift: neighborhood function that performs the relocation move of one customer to another position on the same route. Fig. 4 shows the application of the Intra-Route Shift function, where the customer 6 of route 2 is removed from its position and inserted between customers 4 and 5;

(iv) Inter-Route Shift: neighborhood function that performs the relocation of a customer from one route to another one. The Inter-Route Shift neighborhood function is shown in Fig. 5, in which the customer 12 is taken from route 1 and then inserted into route 3;

(v) Two Intra-Route Swap: neighborhood function that consists of the exchange of customers on the same route, as well as the intra-route swap neighborhood function. However, in the Two Intra-Route Swap function, two consecutive customers are exchanged with two other consecutive customers of the same route;

(vi) Two Intra-Route Shift: neighborhood function that consists of the relocation of customers on the same route, as well as the intra-route shift neighborhood function. However, in the Two Intra-Route Shift function, two consecutive customers are removed from their positions and reinserted into another position of the same route;

(vii) Eliminates Smaller Route: neighborhood function that seeks to eliminate the smallest route of the solution. The smallest route is defined as the route that has the least number of customers. To this end, the customers of the smallest route of the solution are removed and inserted in others routes of the solution. The route and position for insertion of each removed customer are those that result in the best value of the objective function, respecting all constraints. A new solution is generated when all customers of the smallest route are reinserted in other routes. Fig. 6 shows an example of the Eliminates Smaller Route neighborhood function. In this example, the smallest route (route 1) is deleted and its customers are inserted into other routes (routes 2 and 3);

(viii) Eliminates Random Route: The Eliminates Random Route neighborhood function operates similar to the Eliminates Smaller Route function, but the route to be deleted is chosen randomly (see Fig. 7);

3.2. Case 2: UPMSP-ST

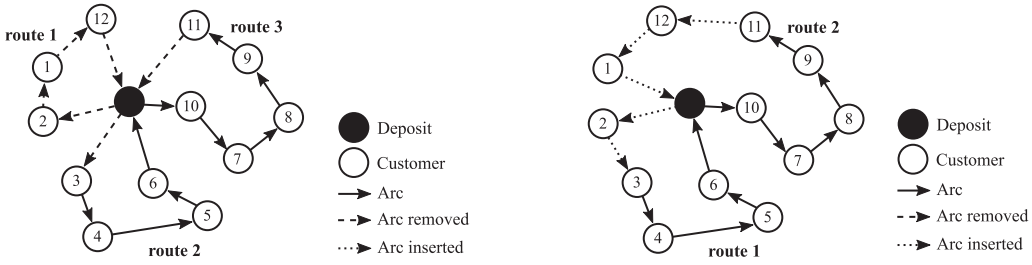
3.2.1. UPMSP-ST basic definitions

In this problem, a set $N = \{n : n = 1, 2, \ldots, |N|\}$ of jobs should be allocated to a set $M = \{m : m = 1, \ldots, |M|\}$ of machines, and each job $j \in N$ must be allocated to a single machine $i \in M$. In the version of UPMSP-ST considered here, we also define the processing time p_{ij}, which represents the time required to process the job $j \in N$ on a machine $i \in M$; and the setup time S_{ijk}, which represents the time required to set up the job $k \in N$ after the job $j \in N$ on the machine $i \in M$. Tables 1 and 2 present an example of this situation, with data referring to test instances for calibration experiments with 8 jobs and 2 machines, proposed by Vallada and Ruiz (2011). Table 1 displays the processing times of each of the 8 jobs on machines m_1 and m_2. Table 2 shows the setup times for machines m_1 and m_2. Since the problem is sequence-dependent, the matrix is used to indicate the setup time of job i after executing job j.

A solution to UPMSP-ST is a list of machines, where each machine is represented by an ordered list of jobs, which defines the sequence in which they will be performed. Fig. 8 illustrates a UPMSP-ST solution in which each job assigned to its machine has

Table 2
Setup time.

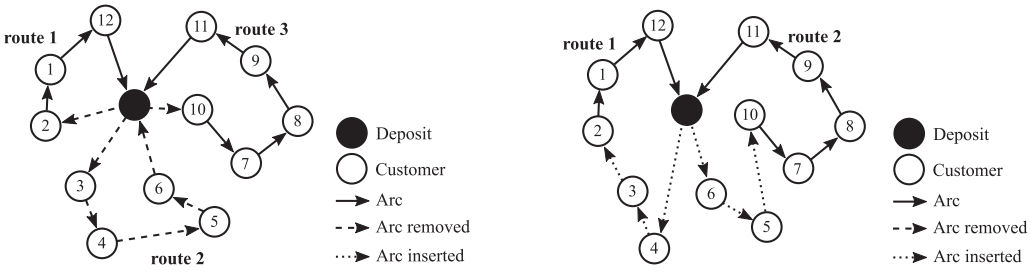
m_1	1	2	3	4	5	6	7	8	m_2	1	2	3	4	5	6	7	8
1	0	5	7	8	8	9	8	5	1	0	2	3	1	2	4	9	5
2	7	0	4	6	6	7	6	8	2	4	0	5	3	1	8	9	7
3	5	2	0	6	6	3	6	7	3	1	3	0	5	5	2	8	7
4	6	9	9	0	6	2	2	8	4	4	2	5	0	3	6	4	5
5	4	8	6	5	0	6	8	2	5	7	6	5	6	0	6	5	2
6	4	2	2	2	2	0	3	8	6	3	6	3	9	4	0	1	6
7	7	1	7	2	9	4	0	9	7	4	1	6	7	2	7	0	9
8	3	8	4	3	8	7	2	0	8	4	9	6	3	6	8	8	0



(a) Eliminating route 1 and inserting the customers in the other routes.

(b) Resulting solution.

Fig. 6. An application of the neighborhood function Eliminates Smaller Route in a solution.



(a) Eliminating route 2 and inserting the customers in the other routes.

(b) Resulting solution.

Fig. 7. An application of the neighborhood function Eliminates Random Route in a solution.

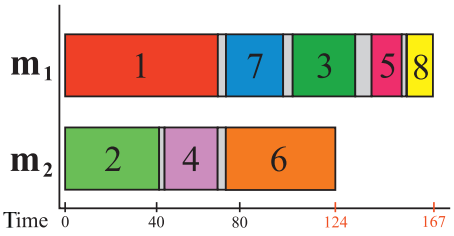


Fig. 8. A UPMSP-ST solution.

its processing time represented in the timeline and the intervals between the jobs represent the setup time. The values used in this example are based on the instance of the Tables 1 and 2.

The objective of UPMSP-ST is to allocate all n jobs on m machines to minimize the maximum completion time of the scheduling, value known as makespan. In Fig. 8, the makespan is defined by the completion time of machine m_1 and is given by the value 167.

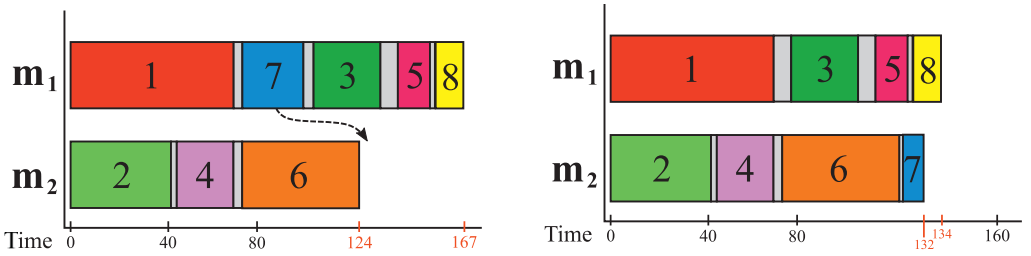
3.2.2. UPMSP-ST neighborhoods

Four neighborhood functions were used to explore the search space of the UPMSP-ST in the framework instantiation. As in VRPTW, these neighborhood structures make up the set of states defined in agent learning. The learning model used in this problem is presented in the Section 5. These structures are presented below:

(i) Multiple Insertion in Different Machines: neighborhood function that performs the relocation of a job from one machine to another one. Fig. 9 shows the application of this neighborhood function in which the job 7 is removed from the machine m_1 and inserted into the machine m_2. As can be seen, the makespan of the initial solution of Fig. 9(a) is obtained by the machine m_1, in the amount of 167; after application of the neighborhood function, the new makespan (solution of Fig. 9(b)) is again obtained by machine m_1, with a value of 134.

(ii) Multiple Insertion in the Same Machines: neighborhood function that performs the relocation move of one job to another position in the same machine. Fig. 10 shows the application of this neighborhood function in which the job 8 is removed from the last position of the machine m_1 and inserted in the second position of this same machine. This modification reduces the makespan. Indeed, before the move, the makespan was 167 (Fig. 10(a)), and, after the move, the makespan reduced to 164 (Fig. 10(b)).

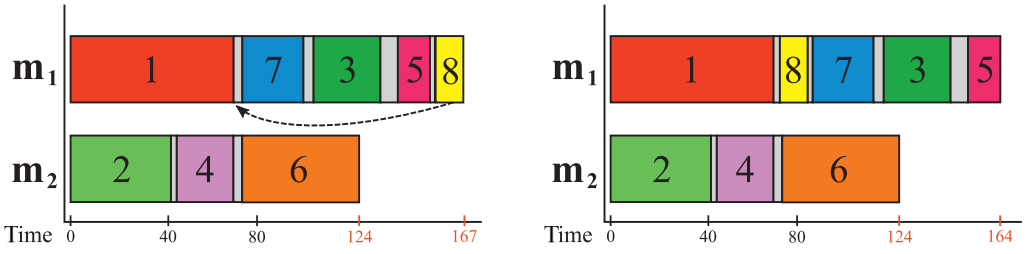
(iii) Swap between different machines: neighborhood function that performs the exchange move of a job of a machine with a job of another machine. These neighborhood functions are shown in Fig. 11, in which the job 1 of the machine m_1 is exchanged with the job 6 of the machine m_2. In this case, the appli-



(a) Initial solution.

(b) Solution after the application of the Multiple Insertion in Different Machines neighborhood function.

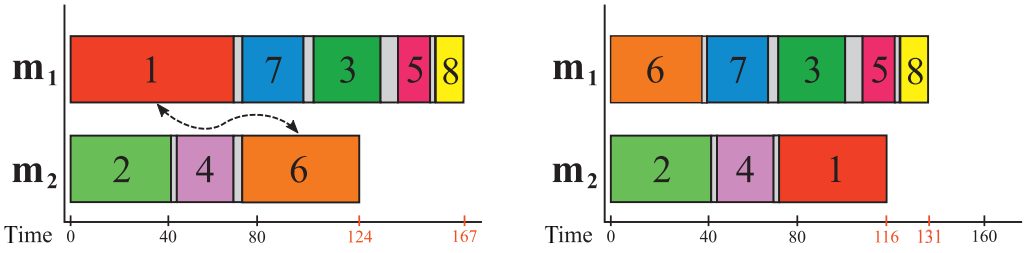
Fig. 9. Example: Multiple Insertion in Different Machines neighborhood.



(a) Initial solution.

(b) Solution after the application of the Multiple Insertion in the Same Machines neighborhood function.

Fig. 10. Example: Multiple Insertion in the Same Machines neighborhood.



(a) Initial solution.

(b) Solution after the application of the Swap between different machines neighborhood function.

Fig. 11. Example: Swap between different machines neighborhood.

cation of the move leads to a significant reduction of the makespan of the solution, since, in the initial solution shown in Fig. 11(a), the makespan value is 167, and, in the resulting solution (Fig. 11(b)), the value obtained is 131.

(iv) Swap between Same Machines: neighborhood function that performs the exchange move of a job with another job of the same machine. These neighborhood functions are shown in Fig. 12, in which the job 1 of the machine m_1 is exchanged with the job 8 of this same machine. The initial solution is shown in Fig. 12(a), with makespan 167. The solution obtained by applying this move is shown in Fig. 12(b), with resulting makespan valued at 163.

4. Multi-agent metaheuristic optimization framework

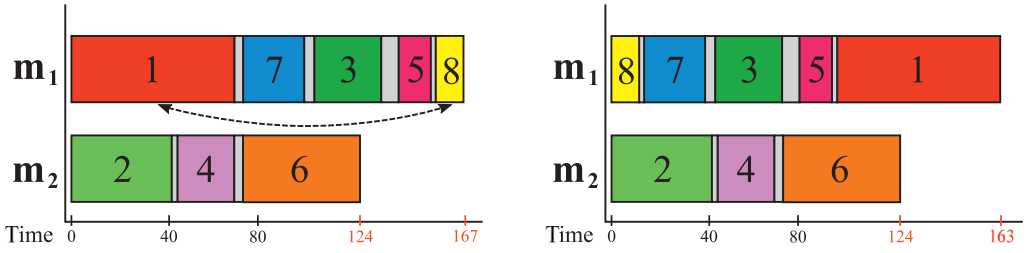
The adaptive agent proposed here is included as an integral part of the AMAM framework. It arises from initial formulations presented in Silva (2007), Fernandes et al. (2009) and Silva et al. (2014, 2015). In this framework, each agent encapsulates a heuristic/metaheuristic and has the function of seeking the solution for a given Combinatorial Optimization problem.

During the search process of the solution, the agents in the framework should go through the multi-agent system environment. In this case, the multi-agent environment is defined by the search space of the addressed problem. As shown in Fig. 13, the perception and action capabilities of the agent are defined in this environment as:

- Perception of the environment: ability of the agents to access information about the problem that are required to it;
- Positioning: ability of the agents to define their positions in the environment, either by the construction of a new solution or by the choice of solutions already available;
- Move: ability of the agent to move, from one solution to another in the environment. The move here comprises all kinds of solution modifications (neighborhood structures, operators) that allow the agent to move from one solution to another;
- Cooperation: ability of the agent to share and provide solutions for the other agents of the system.

The actions available to each agent define the vision that it will have of the environment. Therefore, its representation of the environment is partial. The goal is to apply, at the same time, the strengths of each metaheuristic through the cooperative work of the agents. The scalability of the AMAM architecture is guaranteed by the ease of adding new agents, with minimal impact on the rest of the architecture. These agents interact with the environment and with others agents cooperatively, exchanging and sharing information about their condition and about the environment.

The Object Oriented Programming paradigm is used to facilitate the development of the framework, allowing to reduce the effort



(a) Initial solution.

(b) Solution after the application of the Swap between Same Machines neighborhood function.

Fig. 12. Example: Swap between Same Machines neighborhood.

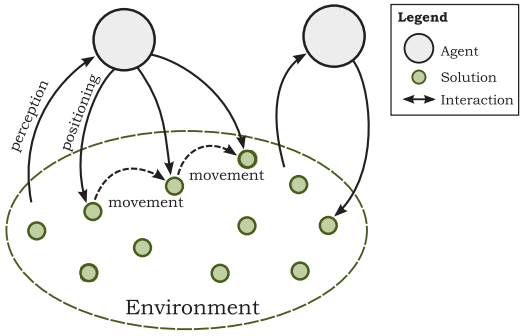


Fig. 13. Agent interaction.

used in the implementation of methods and in the adaptation of these to a specific problem. Therefore, a generic structure that enables the definition of the problem characteristics is used.

The conceptual model of the AMAM framework was originally proposed in Silva (2007). The development from the conceptual model is presented in Fernandes et al. (2009), and allows the creation of an instance of the environment and multiple agents to the search of the solution. Design patterns are used to ensure that the AMAM architecture be flexible and extensible. In this initial conceptual model, six major elements make up the proposed Multi-agent System: (i) Environment; (ii) Constructor Agent; (iii) Local Search Agent; (iv) Metaheuristic Agent; (v) Coordinator Agent; and (vi) Solution Analyzer Agent. The cooperative structure of AMAM architecture is reviewed and improved in Fernandes et al. (2009) and Silva et al. (2014). An adaptive memory strategy called Pool of Solutions is used for sharing information. The available solutions are stored in this Pool of Solutions, located in the Multi-Agent System environment. In the latest proposal, presented in Silva et al. (2014, 2015), the Coordinator and Solution Analyzer agents were removed from the architecture, in relation to the original proposal. The main objective of this change in the structure is to meet the need to increase the autonomy of the agent, preventing other agents interfere in its activities. The new structure for AMAM framework is composed of three main elements:

(i) Environment: defined mainly by the search space of the tackled problem. Therefore, it provides all information that is needed for solving the problem, i.e., in the case of VRPTW, the number of customers to be attended, the distance between customers, the number of vehicles, and so on;
(ii) Pool of Solutions: responsible for maintaining the shared solutions to all agents;
(iii) Metaheuristic Agents: responsible for guiding the search for the solution.

In Silva et al. (2015), self-adaptive skills based on learning are assigned to framework agents, using the principles defined in Learning Automata (Narendra & Thathachar, 1974). Finally, a new adaptive agent, incorporating the reinforcement approach by learning, is presented in the current proposal and is detailed in Section 5.

The strength of the proposed framework is the hybridization capacity of metaheuristics through a multi-agent approach, using concepts of cooperation and parallelism. Additionally, AMAM offers the possibility of parallel execution, in which each agent runs on a separate thread.

The cooperation between agents occurs, in the current release, through the exchange of information in the search space of the problem. The available solutions are stored in a pool of solutions in the environment and shared by the agents at the end of each iteration. The purpose of this cooperative structure is to guide agents in the solutions space toward the most promising areas, and thus, improves the final result and reduces the time needed to solve the problem.

The maximum size of the pool of solutions is predefined and the insertion of new solutions is regulated by an evaluation function, as in Silva et al. (2015). This evaluation function is based on the niching techniques (Li, Epitropakis, Deb, & Engelbrecht, 2017) for coordinating solution files. When a solution needs to be inserted into the pool and no space in this pool is available, the existing solutions are evaluated according to the function:

$$g(\phi_i) = \sum_{j=1}^{P} \phi(\lambda_{ij}) \tag{2}$$

where P is the number of solutions in the pool and λ_{ij} is the distance between the solutions i and j. The evaluation function $g(\phi_i)$ is defined by the sum of the distances of a solution i to all the other pool solutions, where $\phi(\lambda_{ij})$ is defined as:

$$\phi(\lambda_{ij}) = \begin{cases} 1 - \dfrac{\lambda_{ij}}{pr}, & \text{if } \lambda_{ij} \leq pr \\ 0, & \text{if } \lambda_{ij} > pr \end{cases} \tag{3}$$

The factor pr is the pool radius, and controls the dispersion degree of the solutions, being a parameter of the problem. For example, for VRPTW, used as one of the case studies in the current article, it is the minimum number of arcs for one solution to be considered very close to the other. Therefore, its value depends directly on the size of the instance to be solved of the problem addressed.

The function (2) estimates the solution density in the neighborhood of the solution i by means of the distance between the solutions contained in the pool. The value λ_{ij} measures how much the solutions i and j are similar and depends fundamentally on the problem being treated. As an example, considering the case of VRPTW described in Section 3.1, the distance between two solutions is calculated in relation to the number of arcs that are not common to both solutions. Fig. 14 presents two examples of distance calculation between VRPTW solutions. For the first example, the distance between the solution i, shown in Fig. 14(a), and the

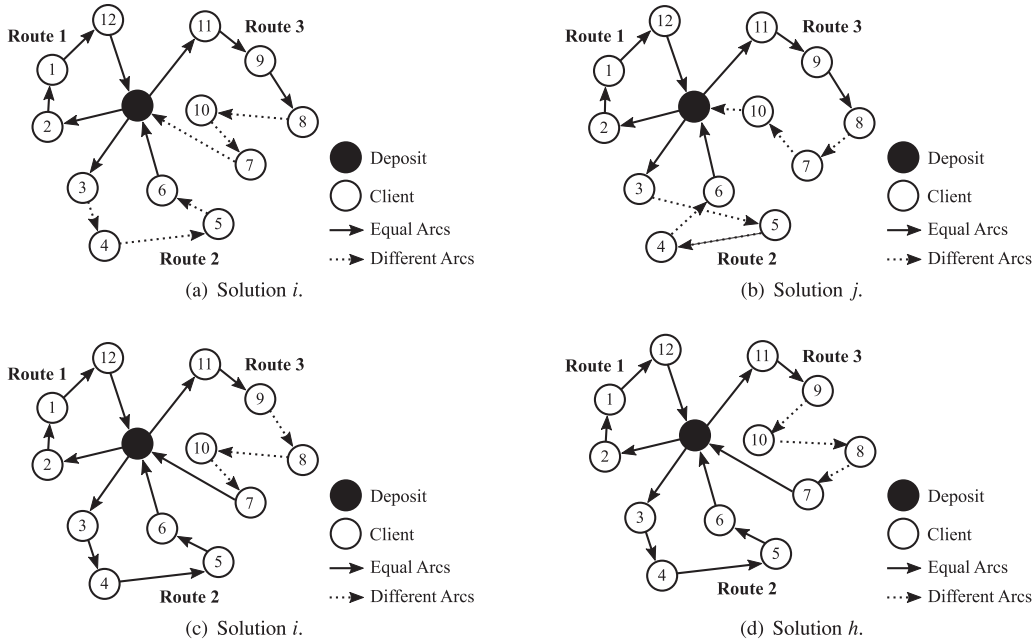


Fig. 14. Examples of calculating distance between solutions.

solution j, shown in Fig. 14(b), is equal to $\lambda_{ij} = 12$, i.e., there are 12 non-common arcs between these solutions. For the second example, the distance between the solution i, shown in Fig. 14(c), and the solution h, shown in Fig. 14(d), is equal to $\lambda_{ih} = 6$, i.e., there are 6 non-common arcs between these solutions.

As a consequence, after evaluating the value $g(\cdot)$ of each solution in the pool, the worst evaluated solution is excluded for the insertion of a new solution in the pool, if this new solution satisfies two criteria: (i) it is not in the pool yet; (ii) it has better objective function value than the worst pool solution. These criteria for the insertion of a new solution are proposed in the current article and is an original contribution of the current article.

The main objective of this evaluation function is to maintain the diversity of the pool, avoiding to keep very similar or even equal solutions. At the same time, the best existing solution in the pool is always stored in a specific attribute of the environment and updated at every insertion, thus preventing that this best solution found is eliminated.

The latest version of the AMAM framework, released in December 2017, is available at https://github.com/mamelials/AMAM-Multiagente-Architecture-for-Metaheuristics, under the GNU LGPLv3 license.

5. Adaptive agents

This section introduces the adaptive capabilities of the framework agents. It extends the learning characteristics initially addressed in Silva et al. (2015) for the same framework.

In Silva et al. (2015), the order of the neighborhoods in local search is chosen by applying an operator similar to the "roulette wheel" selection operator from Genetic Algorithms. For each possible pair of neighborhood structures ($m1$, $m2$), a probability of choice is assigned. Initially, all pairs of sequences have the same probability value. The probability of choice of the sequence ($m1$, $m2$) is updated by a reinforcement factor w if a move of the neighborhood structure m_2 applied after another move of the neighborhood structure m_1 improves the current solution. The concept that was used is very similar to that defined in Learning Automata (Narendra & Thathachar, 1974).

In the current article, the choice of application order of the neighborhood structures of the local search is improved using the Q-Learning algorithm. The details concerning this implementation are described in the following subsections. Initially, in Section 5.1, concepts related to Reinforcement Learning and the used algorithm are introduced. Then, in Section 5.3, the implementation details of this proposal are presented.

5.1. Reinforcement learning

Reinforcement Learning consists of learning what to do in a dynamic environment from trial-and-error-based interactions. These interactions are reinforced according to the effects they cause on the environment. In the model defined by reinforcement learning there are no input/output pairs and therefore the agent needs to gather experiences to improve their performance.

According to Narendra and Thathachar (1974), "*learning is defined as any relatively permanent change in behavior resulting from past experience, and a learning system is characterized by its ability to improve its behavior with time, in some sense tending towards an ultimate goal*". In the reinforcement learning, the behavior is improved from rewards obtained during the interactions of the agent with the environment. The learning takes place through the perception (i) of the state of the individual in the environment; (ii) of the actions performed in this environment; (iii) of the state changes resulting from these actions; and (iv) of the reward that the environment returns in response to the performed action. Through learning, the agent uses the reinforcement value in the subsequent decision-making.

A Reinforcement Learning (RL) system includes three basic aspects: (i) perception; (ii) action; and (iii) goal. In this system, as shown in Fig. 15, the agent perceives (partially) the state of the environment and, based on the knowledge obtained through this perception, selects an action to be performed. The action taken affects the environment, changing the state in which the agent is. Every agent within an RL system has a goal state that must be achieved. The main objective is to take the agent to select a sequence of actions up to the goal state, which maximizes the reinforcement accumulated over time. Thus, a control/decision policy is generated, characterized by the mapping of states and actions, representing

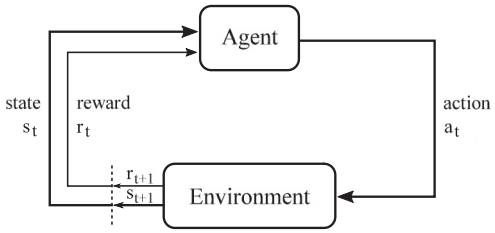


Fig. 15. Interaction agent-environment in the reinforcement learning (Sutton & Barto, 1998).

the behavior that the RL system follows until it reaches the objective.

The main elements that compose the formulation of the Reinforcement Learning Problem are:

(i) Set of States: set of all possible states describing the environment;
(ii) Set of Actions: set of all available actions;
(iii) Environment: the environment in the RL problem is dynamic and must be at least partially observable;
(iv) Control/Decision Policy: defines the behavior of the agent to achieve the goal at any given time. A control policy maps states s into actions a and is expressed by the function $\Pi(s, a)$. This function defines the probability that an action a will be chosen in a s state. These probabilities change as the agent accumulates experiences as a consequence of interactions with the environment. Thus, convergence to optimal policy Π^* expresses the learning process in the RL problem;
(v) Reinforcement/Reward: shows the feedback of the environment in relation to the behavior of the agent. The goal is to maximize this feedback received from the environment. To achieve this goal, the agent must consider the future behavior in the decision making made at the present moment. There are several models that define how the agent should accrue the rewards received. The most used is the finite-horizon discounted model (Kaelbling, Littman, & Moore, 1996). In this model, the RL system seeks to maximize the expected reward as a function of the sequence of received values until an instant of time T, in the form:

$$R_T = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T \tag{4}$$

Considering the rewards received in the long term, a discount factor γ is applied to the expression of reinforcement. As a consequence:

$$R_T = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots + r_T = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{5}$$

where $0 \leq \gamma \leq 1$. Thus, if $\gamma = 0$, immediate reinforcements are maximized; if $\gamma = 1$, the same importance is given to immediate and future earnings;
(vi) Reinforcement Function(Reward Function): reinforcement functions are not always simple to define and vary according to the problem addressed.
(vii) Value Function: value obtained with the mapping of the state or of the action-state pair, from the current and future rewards.
(a) Value-State Function: function that considers only the state and is denoted by $V^{\Pi}(s)$. The value-state function depends on the Π policy and is defined by:

$$V^{\Pi}(s) = E_{\Pi}\{R_t \mid s_t = s\} = E_{\Pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \tag{6}$$

(b) Value-Action Function: function which considers the state-action pair and is denoted by $Q^{\Pi}(s, a)$. As in the value-state function, the value-action function depends on the Π policy and is defined by:

$$Q^{\Pi}(s, a) = E_{\Pi}\{R_t \mid s_t = s, a_t = a\} = E_{\Pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \tag{7}$$

The value-state and value-action functions can be modeled by the Markov Decision Process (MDP) (Bellman, 1957; Bertsekas, 1987; Puterman, 1994). A good literature review of Reinforcement Learning is conducted by Kaelbling et al. (1996).

5.2. Q-learning algorithm

The Q-Learning algorithm, introduced by Watkins and Dayan (1992), stands out for being widely used, for being model-free and for dispensing knowledge of a policy. In this way, the agent, through the Q-learning algorithm, updates its function value, while following any policy. This algorithm allows to find an optimum policy of actions selection for any finite Markov Decision Process (MDP). The objective is, at each step of an episode, to maximize the value of the function $Q(s, a)$, defined as:

$$Q(s, a) = Q(s, a) + \alpha\left[r + \gamma \max_{\alpha'} Q(s', a') - Q(s, a)\right] \tag{8}$$

as shown in the pseudo-code presented in the Algorithm 1, where

Algorithm 1 Q-Learning algorithm.

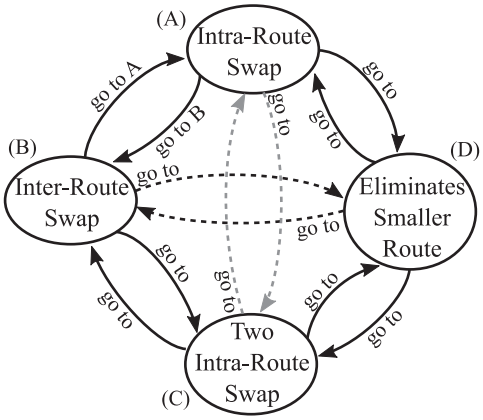
```
procedure QLearning(r, α, ϵ, γ)
    Initialize Q(s, a) arbitrarily;
    repeat                                          ▷ for each episode
        Initialize s;
        repeat                                      ▷ for each step of episode
            Choose a from s using policy derived from Q (e.g., ϵ-greedy);
            Take action a;
            Observe the next state s′ and the reward r;
            Q(s, a) ← Q(s, a) + α[r + γ max_α′ Q(s′, a′) − Q(s, a)];
            s ← s′;
        until s is terminal
    until Reaches the number of episodes
end procedure
```

α is the rate of learning; γ is the discount factor; s is the current state; a is the action taken; and s' is the resulting state. The learning rate and the discount factor are parameters that depend directly on the problem dealt with. The Q function estimates the expected utility of taking an action a in a given state s. An episode is defined here as a sequence of states ranging from an initial state to the final state.

5.3. RL-based adaptive agent

This section shows the RL-based adaptive agent. The adaptive capacity is assigned to the agent through an Adaptive Local Search (ALS-QLearning) based on the Variable Neighborhood Descent heuristic – VND (Mladenović & Hansen, 1997) and on the concepts of Reinforcement Learning.

VND is a heuristic of refinement that exploits the space of solutions by the systematic exchange of neighborhoods. Algorithm 2 presents the standard VND structure. As seen, for each neighborhood $\mathcal{N}(k)$ selected by the standard VND, a local search is performed on the current solution in order to find its best neighbor. This method employs a deterministic neighborhood ordering, being this neighborhood ordering scheme a parameter to be



(a) A graph with four states (neighborhood functions of the VRPTW) and the respective actions of MDP.

Q-table

state/action	A	B	C	D
A	0	0	0	0
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0

(b) Initial Q-Table for these four neighborhood functions.

Fig. 16. A graph representing the relationship between states (neighborhood functions) and possible actions.

Algorithm 2 Variable Neighborhood Descent (VND).

```
1: procedure VND(x, k_max)        ▷ k_max is the number of different neighborhood structures
2:     k ← 1;
3:     while k ≤ k_max do
4:         x ← bestNeighbor(x, N(k));
5:         if f(x') < f(x) then
6:             x ← x';
7:             k ← 1;
8:         else
9:             k ← k + 1;
10:        end if
11:    end while
12: end procedure
```

Algorithm 3 Next action choice function.

```
1: procedure CHOOSEANACTION(state, type_function)
2:     next_state ← 0;
3:     if type_function = 1 then
4:         next_state ← ε-greedy(state);
5:     else
6:         if type_function = 2 then
7:             next_state ← randomAction();
8:         end if
9:     end if
10: end procedure
```

determined. In general, this scheme is based on the complexity growth order of these structures. (i.e., the application order is predefined), but this order does not always produce the best solution because the best order may be highly dependent on the instance (Subramanian, Drummond, Bentes, Ochi, & Farias, 2010). If the solution found is better than the current solution, the first neighborhood function is used again; otherwise, the next neighborhood function is used until there are no more neighborhoods available. VND returns a local optimum in relation to the all explored neighborhoods.

In this proposal, the sequence in which neighborhoods are applied is defined through reinforcement learning, based on the Q-Learning algorithm. The main objective is to evaluate the gain obtained with the application of a sequence of two neighborhoods, and, from there, to reward the best sequences and maximize the accumulated reward. Each neighborhood to be used by the search method is considered, in this article, as a learning state.

The Markov Decision Process (MDP) for this proposal is defined as follows:

- Set of States S: states are the neighborhood functions, available for the problem to be handled by the framework. In the case of the test problems used here, we have:
  (a) For the VRPTW case study: the set of states is formed by the 8 neighborhood functions listed at Section 3.1.2:

$$S_{vrptw} = \{\text{Intra-Route Swap, Inter-Route Swap, Intra-Route Shift, Inter-Route Shift, Two Intra-Route Swap, Two Intra-Route Shift, Eliminates Smaller Route, Eliminates Random Route}\} \quad (9)$$

  (b) For the UPMSP-ST case study: the set of states is formed by the 4 neighborhood functions listed at Section 3.2.2:

$$S_{upmsp-st} = \{\text{Multiple Insertion in the Different Machines, Multiple Insertion in the Same Machines, Swap between Different Machines, Swap between Same Machines}\} \quad (10)$$

- Set of Actions $A(s)$: an action is defined as the change from one state to another ("*go to*"). In this way, the set of actions can be represented by a complete graph, in which each action is represented by an arc connecting two states (nodes of the graph). An example of a graph representing the relationship between states (neighborhood functions) and possible actions is shown in Fig. 16(a). In this example, only four neighborhood functions of the VRPTW are used to facilitate visualization. The Q table, referring to this example, is shown in Fig. 16(b). The Q table has dimensions given by $M \times M$, in which M is the number of states, that is, the number of neighborhood functions of the treated problem.
- Reward R: based on the value of the fitness function of the solution x obtained with the application of the current neighborhood function $\mathcal{N}(x)$. The way in which this transformation takes place is presented below.

Since the neighborhood functions are specific parameters of the problem being solved, the purpose of using reinforcement learning

is to allow the framework to better adapt to the own characteristics of this problem.

The adaptive local search algorithm proposed in this article is presented in Algorithm 4. In this algorithm, firstly, the Q table is initialized. This table is responsible for storing the values of the state-action pairs in the course of the learning process. In this proposal, each element of the Q table is initialized with a zero value, so as the learning process is carried out non-biased.

Algorithm 4 Adaptive Local Search based on Q-learning.

```
1: procedure ADAPTIVELOCALSEARCHQLEARNIG(x_0)
2:     initialize(Q(state, action));
3:     improved ← true;
4:     no_improvement ← 0;
5:     x* ← x_0;
6:     x ← x_0;
7:     repeat                                          ▷ for each episode
8:         reward ← 0;
9:         states_visited_count ← 0;
10:        next_state ← chooseAnAction(0, 2);          ▷ 2: initial state defined by random function
11:        x ← bestNeighbor(next_state, x);
12:        if (x is better than x*) then
13:            x* ← x;
14:            reward ← x.getFitnessLearning();
15:        else
16:            states_visited_count + +;
17:            while (not reached the state goal) do   ▷ state goal: improving the solution
18:                if (no_improvement = 0) then
19:                    state ← next_state;
20:                    next_state = chooseAnAction(state, 1);   ▷ 1: epsilon greedy function
21:                else
22:                    next_state = chooseAnAction(0, 2);   ▷ if not improved, the greedy function should not be used
23:                end if
24:                x = bestNeighbor(next_state, x);
25:                if (x is better than x*) then    ▷ reached the state goal
26:                    x* ← x;
27:                    improved ← true;
28:                    no_improvement ← 0;
29:                    reward ← reward + x.getFitnessLearning();
30:                    calculateQValue(state, next_state, reward);
31:                else
32:                    no_improvement + +;
33:                    if (the state has been visited) then
34:                        states_visited_count + +;
35:                    end if
36:                    if ((no_improvement > max_iterations_without_improvement) and (states_visited_count = q_size)) then
37:                        improved ← false;
38:                    end if
39:                end if
40:            end while
41:            ϵ ← ϵ * decay_rate;
42:        end if
43:    until (not(improved))
44:    return x;
45: end procedure
```

Then, for each episode related to the Q-Learning algorithm, the learning is associated with the search process of the solution. For this, in the first part, the initial state is randomly selected. The *chooseAnAction(state, type-function)* function, presented in Algorithm 3, is used to determine both the initial state (where *type-function* = 2 as shown at line 6 of Algorithm 3) and to determine next states with specific functions (where *type-function* = 1 as shown at line 3 of Algorithm 3). In this way, this function allows new selection functions to be easily inserted. The initial state is then applied to the solution through the best improvement strategy (*bestNeighbor(next-state, x)* – line 11 of Algorithm 4).

The reward used is based on the fitness function of the solution and its calculation depends on the type of problem being treated. Considering the treatment given to the Q matrix in the Q-Learning algorithm, the fitness value $reward(x)$ of the solution x is evaluated on the basis of the objective function $f(x)$ of the problem. Then, in the maximization problems, $reward(x) = f(x)$; in the minimization problems, $reward(x) = f(x_0) - f(x)$, where x_0 is the initial solution of the current local search.

As the objective is to define the gain obtained by applying a sequence of two neighborhoods, the reward is calculated from the sum of the values obtained by these neighborhoods, as shown at lines 14 and 29 of the Algorithm 4. In case of improvement of the solution with the neighborhood application, the value to be added to the reward is the fitness function of the current solution; if the new solution found is not better than the current one, the value to be added is zero.

The second part of the episode is executed until the goal is reached (lines 17–40 of the Algorithm 4). The objective is to find a better solution than the current one and, if achieved, ends the current episode. The other states to be visited in the episode are defined using the ϵ-greedy function (Watkins, 1989), presented in the Algorithm 5. The ϵ-greedy function selects a random action with a probability ϵ or an action that returns the highest reward with a probability $1 - \epsilon$. The next state is also applied to the solution through the best improvement strategy (*bestNeighbor(next-state, x)* – line 24 of the Algorithm 4). When the solution improves, the reward is assigned, its value in the Q matrix is calculated, the value of ϵ decreases with the decay rate, and the episode ends.

Algorithm 5 ϵ-greedy function.

```
1: function ϵ-GREEDY(current_state, ϵ, q_size)
2:     p ← random();
3:     if p ≤ ϵ then
4:         action ← random(q_size);
5:     else
6:         action ← maxAction(current_state);
7:     end if
8:     returnaction;
9: end function
```

Considering the specific characteristics of the optimization problems, a neighborhood needs to be applied to a solution, from a Descent/Ascent local search heuristic, only once. In consequence, the visited states (neighborhoods) are registered. When all states have been visited, the local search is finished. The same happens after a certain number of iterations without improvement. For the evaluation of the adaptive agent introduced here, the following section presents the experiments performed.

6. Computational experiments

This section presents the computational experiments performed in order to evaluate and test the AMAM framework. This framework is implemented in Java language with JDK 1.8, using the IDE Elipse neon 2. The experiments were executed using a computer with processor Intel i7 - 4500U, 1.8 GHz, 16 GB DDR3 RAM and

operational system Windows 7 Home Premium. It is important to highlight that the tests presented here were executed in a common notebook, with a single processor, showing the effectiveness of the exposed framework.

In order to allow its evaluation, the proposed framework was instantiated for two classical problems of Combinatorial Optimization. The first problem instantiated is the Vehicle Routing Problem with Time Window (VRPTW). The second problem instantiated is the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSP-ST). A description of these problems is presented in Section 4.

The 56 instances of VRPTW with 100 customers proposed by Solomon (1987) were used. These instances are formed by three different sets of customers (C-Cluster; R-Random; and RC-Random-Cluster) in accordance with the geographic distribution considered. Customer geographical positions are randomly generated in the R1 and R2 sets of problems. In C1 and C2 sets of problems, the geographical positions are clustered. In addition, a mixture of randomized and clustered structures are used in RC1 and RC2 problem sets. It is worthy mentioning that competing with the best literature results for these instances of VRPTW is out the scope of this experiment.

The set of instances used for computational tests associated with UPMSP-ST were proposed by Vallada and Ruiz (2011) and are available at http://soa.iti.es/problem-instances. For evaluating AMAM, 24 instances from this data set were chosen. These test problems used involve combinations of 50 and 100 jobs, with 10, 15, 20 and 25 machines. As for VRPTW, the main objective of this experiment is not to overcome the best results in the literature for UPMSP-ST.

The experiments were conducted in order to analyze the performance of the adaptive agent proposed here. The main objective is to evaluate if the form of learning, embedded in the agent, has a direct influence on the performance of the framework with respect to the quality of the obtained results, both in terms of the individual point of view and in terms of the teamwork point of view.

For this evaluation, the tests were made with the adaptive agent presented in the current proposal and with the adaptive agent proposed in Silva et al. (2015). The tests were carried out using the same test conditions for the two adaptive agent structures considered here and, as a consequence, the tests with the adaptive agent proposed in Silva et al. (2015) were executed again.

The agents used in this experiment implement, as performed in Silva et al. (2015), a variation of the Iterated Local Search (ILS) metaheuristic (Lourenço, Martin, & Stützle, 2003), a well known trajectory metaheuristic. This method is shown in Algorithm 6. In

Algorithm 6 Iterated Local Search (Lourenço et al., 2003).

```
1:  procedure ILS
2:      x_o ← pfihInitialSolution();
3:      x ← localSearch(x_o);
4:      while (not reached the stopping condition) do
5:          x' ← perturbation(x, level_perturbation);
6:          x'' ← adaptiveLocalSearch(x');
7:          if (acceptationCriteria(x, x'')) then
8:              x ← x'';
9:              level_perturbation ← 1;
10:         else
11:             level_perturbation++;
12:         end if
13:     end while
14: end procedure
```

this algorithm, the perturbation of the solution, performed from changes in the current solution, is implemented at levels, i.e., at each iteration the perturbation function is changed if there is no improvement in the solution (line 11), and returns to its first level (line 9), if a better solution is found. The *adaptiveLocalSearch*(s') function implements the specific adaptive local search of each proposal. The proposal described in Silva et al. (2015) is called here ALS-LA (Adaptive Local Search - Learning Automata); the proposal described in the current article is referred here as ALS-QLearning (Adaptive Local Search - QLearning). For the computational tests shown here, the values of the RL parameters (see Algorithm 1) used in the experiments are $\gamma = 0.9$, $\alpha = 0.1$ and $\epsilon = 0.05$, with decay rate given by 0.999.

The composition of the multi-agent system, used in the experiments for each of the proposals, involved identical ILS agents and the cooperation process described in Section 4. Four scenarios for each one of the two proposals are used to evaluate the framework:

(i) a single ILS agent;
(ii) two identical ILS agents;
(iii) four identical ILS agents;
(iv) eight identical ILS agents;

In this context, the proposals presented were analyzed in five ways:

(i) ALS-LA proposal performance: specific analysis of the results obtained by the ALS-LA proposal, comparing the performance of the scenario in which there is a single agent, which performs the search in an isolated manner, with the performance of the other scenarios (2 agents, 4 agents and 8 agents), in which a set of agents cooperates in the search for the solution. This item will allow us to evaluate the effectiveness of the cooperation and the scalability of the framework using the ALS-LA proposal;
(ii) ALS-QLearning proposal performance: specific analysis of the results obtained by the ALS-QLearning proposal, with the same purposes of the previous item;
(iii) Comparison between the two proposals: comparison of the means of solutions obtained by the two proposals (ALS-LA and ALS-QLearning);
(iv) Performance of individual learning: comparison of the performance of a single agent using each of the proposals (ALS-LA and ALS-QLearning); and,
(v) Team performance: the influence of individual learning on the cooperative process, through the use of two or more agents cooperating to solve the problem, also evaluated for each of the proposals (ALS-LA and ALS-QLearning).

Once the used algorithms are of stochastic nature, each one of the eight evaluated scenarios, being four scenarios about the ALS-LA proposal and four scenarios concerning the ALS-QLearning proposal, were executed 30 times for each instance. The results obtained in each one of these scenarios were compared using a parametric hypothesis test, with a confidence level of 95%. The parametric test used was the ANOVA variance analysis. This test verifies if there are differences between the averages of the populations of evaluated solutions. In this way, the hypotheses should be formulated as follows. Considering two scenarios SC1 and SC2, the following hypotheses are raised to compare the average solutions obtained in each one:

(i) Null hypothesis (H_0): the average of the solutions obtained in scenarios SC1 and SC2 is equal. If the null hypothesis is not rejected, then there is no significant statistical difference between the solutions obtained by the scenarios analyzed;
(ii) Alternative hypothesis (H_1): the average of the solutions obtained using SC1 is less than the average of the solutions obtained by SC2. If the null hypothesis is rejected, then there is statistical evidence (with 95% confidence level) that the solutions obtained by SC1 are better than those obtained by SC2.

Table 3
Number of times each scenario was better with ALS-LA VRPTW – values obtained from the parametric test.

Set of instances	Total instances per set	Scenarios			
		1 agent	2 agents	4 agents	8 agents
C1	**9**	1	2	2	5
C2	**8**	0	0	4	5
R1	**12**	0	4	8	8
R2	**11**	1	3	4	9
RC1	**8**	0	2	2	7
RC2	**8**	0	0	6	4
Total	**56**	**2**	**11**	**26**	**38**

Table 4
Number of times each scenario was better with ALS-LA UPMSP-ST – values obtained from the parametric test.

Set of instances	Total instances per set	Scenarios			
		1 agent	2 agents	4 agents	8 agents
50 jobs	**16**	0	1	3	16
100 jobs	**8**	0	0	1	8
Total	**24**	**0**	**1**	**4**	**24**

6.1. ALS-LA proposal

The analysis will begin by addressing the results obtained in the four scenarios of the ALS-LA proposal for the two problems instantiated. Tables 3 and 4 show the number of times that each scenario of the ALS-LA proposal was better, for the VRPTW instances and UPMSP-ST instances, respectively, based on the results of the parametric test used. The objective here is to evaluate the scalability of the proposals, that is, if the increase in the number of agents involved in the solution influences the performance of the multi-agent system.

For the VRPTW instances, the obtained results showed that there is statistical evidence that, at 89.28% of the instances, the scenarios with 2 or more agents were better than the scenario with 1 single agent. On the other hand, for the UPMSP-ST instances, the obtained results showed that there is statistical evidence that using 2 or more agents were better at 100.00% of the instances. Tables 3 and 4 also show that when the number of agents used to solve the two problems is increased, the evaluation function of the two problems decreases.

Figs. 17 and 18 use boxplot graphics to illustrate, through the results obtained in the 30 executions of the R201 and R203 instances of VRPTW, respectively, the improvement in the quality of the solution with the addition of more agents to the solution process. The same result can be observed in the boxplot graphic presented in Fig. 19, used to illustrate the results obtained for the 30 executions of the UPMSP-ST I_50_10_S_1-9_1 instance.

It is important to highlight that, in some cases, the scenarios tied with the best results, i.e., there was no statistical difference between the averages of the solutions. In these cases, all the scenarios that matched the best results were counted.

6.2. ALS-Q learning proposal

In this section, we focus on the results obtained by the ALS-QLearning proposal, introduced in the current article, for the two problems instantiated.

Regarding the VRPTW, from the 56 instances analyzed, in 12 of them (21.42%), the ALS-QLearning proposal obtained the best solution of the literature in all scenarios (including the scenario with a single agent). Therefore, in these instances, there was no statistical difference for comparison and, as a consequence, they were excluded from this analysis. For the remaining 44 instances, there is statistical evidence that, at 90.90% of them, the scenarios with 2 or more agents were better than the scenario with 1 single agent. Concerning the UPMSP-ST, there is statistical evidence that, at 95.83% from the 24 instances analyzed, the scenarios with 2 or more agents were better than the scenario with 1 single agent.

Tables 5 and 6 show the number of times that each scenario of the ALS-QLearning proposal was better, for the VRPTW and UPMSP-ST instances, respectively, based on the results of the parametric test used. As in Section 6.1, the objective is to evaluate the scalability of the proposals. In a similar way to the results shown in Tables 3 and 4, by the Tables 5 and 6, it is concluded that increases the number of times that each scenario is better in so far as the number of agents also grows.

Figs. 20 and 21 show examples of the effect of the addition of agents in the solution process. This figure exhibits the solution due to the instance R111, for VRPTW, and instance I_50_15_S_1-99_1, for UPMSP-ST problems, respectively, and reveals its improvement in so far as the number of agents grows.

As described in relation to the results of the ALS-LA proposal, for the VRPTW, the results obtained in some scenarios of the ALS-QLearning proposal also tied with the best results in the literature. In these cases, as in the analysis of the ALS-LA proposal, all the scenarios that matched the best results were counted.

6.3. ALS-LA × ALS-Q learning

This section presents the comparison between averages values of the solutions found for the proposals ALS-LA and ALS-QLearning, evaluated in this experiment. This comparison addressed the two problems instantiated. Sections 6.3.1 and 6.3.2 show the analysis of these results.

6.3.1. VRPTW

The cost of a solution for VRPTW is calculated according to Eq. (1). In this equation, the priority is to minimize the number of routes of the solution, i.e., the number of vehicles that are used. In some cases, the solutions generated in the evaluated scenarios may have different number of routes. If the number of routes is different, it is not possible to compare the cost of these solutions. In these cases, the comparison is made through the number of routes. The instances that obtained the same number of routes were compared in relation to the distance traveled. The same happens for the instances that obtained different number of routes, but did not

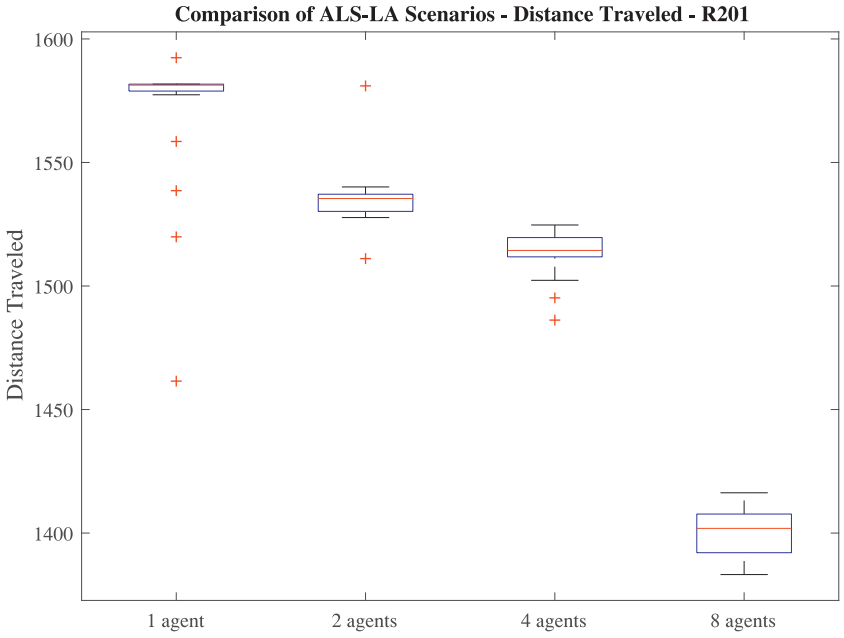


Fig. 17. Comparison of the ALS-LA scenarios in relation to the distance traveled - R201 instance.

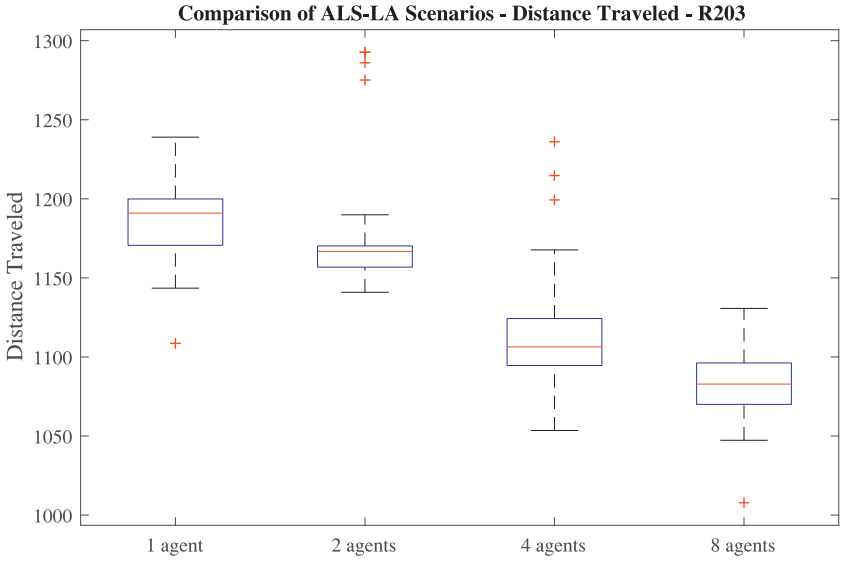


Fig. 18. Comparison of the ALS-LA scenarios in relation to the distance traveled - R203 instance.

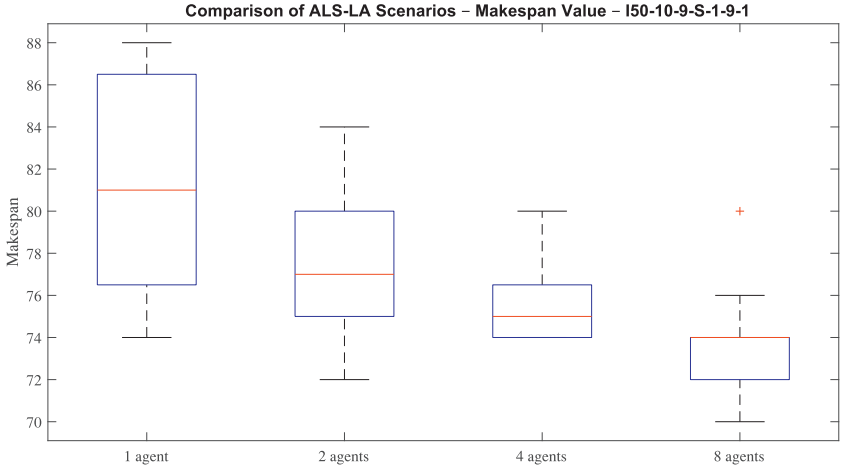


Fig. 19. Comparison of the ALS-LA scenarios in relation to the makespan - UPMSP-ST - I_50_10_S_1-9_1 instance.

Table 5
Number of times each scenario was better with ALS-QLearning – values obtained from the parametric test.

Set of instances	Total instances per set	Scenarios			
		1 agent	2 agents	4 agents	8 agents
C1	**3**	0	1	1	2
C2	**2**	0	1	1	2
R1	**12**	0	2	5	11
R2	**11**	0	4	4	9
RC1	**8**	1	3	6	8
RC2	**8**	0	2	4	6
Total	**44**	**1**	**13**	**21**	**38**

Table 6
Number of times each scenario was better with ALS-QLearning UPMSP-ST – values obtained from the parametric test.

Set of instances	Total instances per set	Scenarios			
		1 agent	2 agents	4 agents	8 agents
50 jobs	**16**	0	0	2	14
100 jobs	**8**	0	1	1	8
Total	**24**	**0**	**1**	**3**	**22**

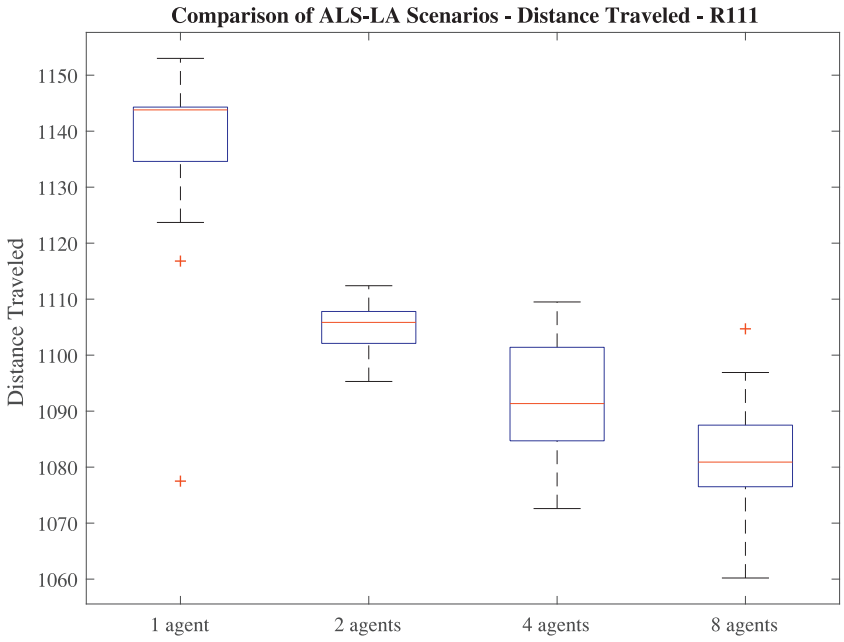


Fig. 20. Comparison of the ALS-QLearning scenarios in relation to the distance traveled - R111 instance.

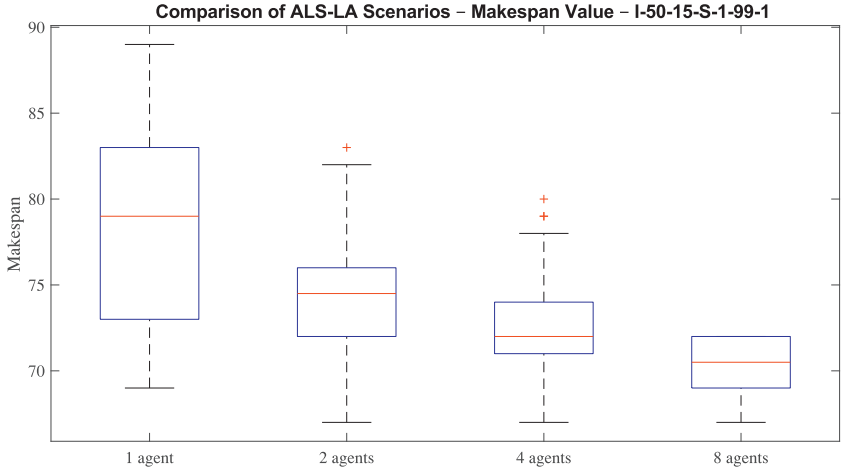


Fig. 21. Comparison of the ALS-QLearning scenarios in relation to the makespan - I_50_15_S_1-99_1 instance.

Table 7
Number of instances where the ALS-QLearning proposal is better than the ALS-LA proposal, per scenario, considering the Number of Routes.

Set of instances	Total instances per set	Total instances per set with different routes number	Scenarios			
			1 agent	2 agents	4 agents	8 agents
C1	**9**	**5**	5	4	2	4
C2	**8**	**1**	1	0	0	0
R1	**12**	**12**	8	10	9	11
R2	**11**	**2**	1	0	0	1
RC1	**8**	**8**	7	6	8	8
RC2	**8**	**3**	3	2	2	1
Total of instances	**56**	**31**	**25**	**22**	**21**	**25**

Table 8
Number of instances where the ALS-QLearning proposal is better than the ALS-LA proposal, per scenario, considering the total distance traveled.

Set of instances	Total instances per set	Total instances per set with equal routes number	Scenarios			
			1 agent	2 agents	4 agents	8 agents
C1	**9**	**4**	4	4	4	4
C2	**8**	**7**	6	6	4	6
R1	**12**	**0**	0	0	0	0
R2	**11**	**9**	9	9	9	9
RC1	**8**	**0**	0	0	0	0
RC2	**8**	**5**	5	5	5	5
Total of instances	**56**	**25**	**24**	**24**	**22**	**24**

Fig. 22. Comparison between ALS-LA and ALS-QLearning proposals in relation to the number of routes – RC106 instance.

present statistical difference in these values. In this case, the solutions with number of routes that do not influence the result were removed and, therefore, the statistical analysis was done with the other solutions in relation to the distance traveled.

In relation to the statistical analysis and considering the number of solutions routes, the ALS-QLearning proposal obtained the best results in the majority of instances and scenarios for VRPTW. This proposal found the best number of routes known in the literature in 60.71% of the evaluated instances, while the ALS-LA proposal achieved the best number of routes known in the literature at only 26.78% instances.

In this context, the number of times ALS-QLearning was better per scenario than ALS-LA was evaluated, considering the number of routes of the analyzed solutions. Table 7 shows the values concerning 31 instances. Regarding the 25 remain instances, ALS-QLearning and ALS-LA obtained the same number of routes and this situation is evaluated in Table 8 in the sequel. As can be seen, the ALS-QLearning proposal obtained better results in most of the analyzed instances. The results for groups of instances R1 and RC1 should be highlighted because the ALS-QLearning proposal was able to improve the results found in the ALS-LA proposal in all instances of these groups.

Fig. 22 illustrates the situation where the ALS-QLearning proposal achieves better results than the ALS-LA proposal. In this figure, the results for instance R106 show a reduction from 15 routes (obtained by ALS-LA) to 12 routes (obtained by ALS-QLearning).

The 25 instances not addressed in the situation dealt with in Table 7, that is, the ones that obtained the same number of routes for both the ALS-SA proposal and the ALS-QLearning proposal, were analyzed in Table 8 for the total distance traveled. This Table shows the number of times there is statistical evidence that ALS-QLearning was better than ALS-LA, regarding the distance traveled and per scenario. A direct observation of Table 7 concludes that the ALS-QLearning proposal obtained better results in most of the analyzed instances.

Fig. 23 shows a good example where the values of ALS-QLearning solutions are significantly better than ALS-LA solutions. In this specific case, there is a reduction in the distance traveled from 1609.4, obtained by ALS-LA, to the value of 1249.4, which is equivalent to the best known result of the literature, obtained by ALS-QLearning.

The performance of the individual learning and the team learning of the agents in the presented multi-agent environment were also evaluated. Table 9 presents this analysis. Regarding individual learning, the number of times that a single agent using the ALS-

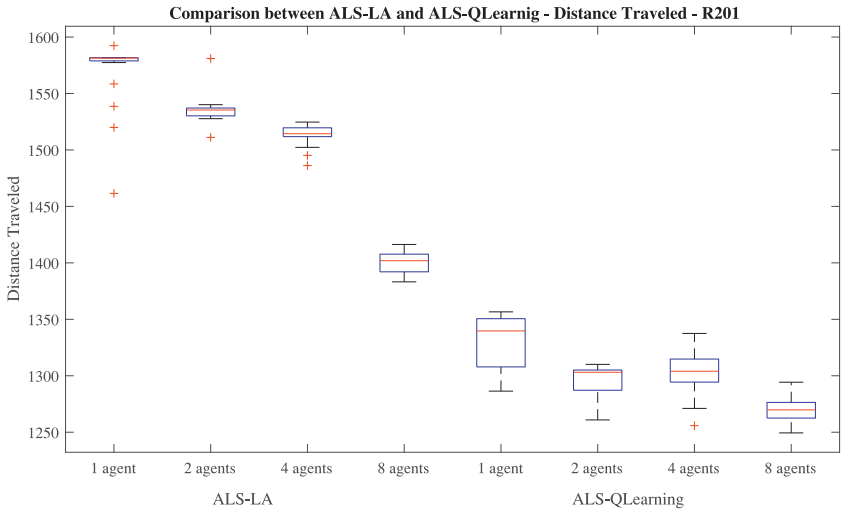


Fig. 23. Comparison between ALS-LA and ALS-QLearning proposals in relation to the distance traveled – R201 instance.

Table 9
Number of times ALS-QLearning is better than ALS-LA individually and as a team VRPTW.

Set of instances	Total of instances per set	Scenarios			
		Individually		In group	
		Value	%	Value	%
C1	**9**	9	100.0	9	100.0
C2	**8**	7	87.5	7	87.5
R1	**12**	11	91.7	12	100.0
R2	**11**	10	90.9	10	90.9
RC1	**8**	7	87.5	8	100.0
RC2	**8**	8	100.0	8	100.0
Total	**56**	**52.0**	**92.9**	**54**	**96.4**

QLearning proposal was better than a single agent using the ALS-LA proposal was assessed (value presented in the third column and the corresponding percentage value in the fourth column of the table). Regarding the evaluation of the influence of learning in the cooperative process, that is, in teamwork, the number of times the ALS-QLearning proposal was better, in two or more scenarios, than the ALS-LA proposal was also presented (value presented in the fifth column and the corresponding percentage value in the fifth column of the table).

At 92.9% of the instances, the parametric test used shows that there is statistical evidence that ALS-Q Learning is better than ALS-LA when using only one agent to solve the problem. Regarding the use of two or more agents to solve the problem, at 96.4% of the used instances the parametric test shows that there is statistical evidence that ALS-QLearning is better than ALS-LA.

Fig. 24 presents an example of the distances traveled obtained in the 30 runs of instance R203, for the four scenarios used in each of the tested proposals (ALS-LA and ALS-Q Learning). It is important to note how the values achieved by ALS-QLearning are better than those found by ALS-LA, whether considering a single agent or considering two or more agents. In addition, the values improve as the number of agents involved in the search for the solution increases.

6.3.2. UPMSP-ST

In relation to the statistical analysis and considering the makespan value, the ALS-QLearning proposal obtained the best results in the majority of instances and scenarios for UPMSP-ST.

In this context, the number of times ALS-QLearning was better per scenario than ALS-LA was evaluated, considering the makespan of the analyzed solutions. Table 10 shows the values concerning 24 instances. As can be seen, the ALS-QLearning proposal obtained better results in most of the analyzed instances. The results for groups of instances with 100 jobs should be highlighted because the ALS-QLearning proposal was able to improve the results found in the ALS-LA proposal in all instances of these groups.

Fig. 25 shows a good example where the values of ALS-QLearning solutions are significantly better than ALS-LA solutions. In this specific case, there is a reduction in the average makespan from 273.00, obtained by ALS-LA, to 221.00, which is equivalent to the best known result of the literature, obtained by ALS-QLearning.

The performance of the individual learning and the team learning of the agents in the presented multi-agent environment were also evaluated for the UPMSP-ST. Table 11 presents this analysis. Just like in VRPTW, regarding individual learning, the number of times that a single agent using the ALS-QLearning proposal was better than a single agent using the ALS-LA proposal was evaluated. This value is presented in the third column and the corresponding percentage value is showed in the fourth column of the table. Concerning the evaluation of the influence of learning in the cooperative process, that is, in teamwork, the number of times the ALS-QLearning proposal was better than the ALS-LA proposal in two or more scenarios was also presented. This value is presented in the fifth column and the corresponding percentage value is in the fifth column of the table.

For 83.33% of the instances, the parametric test used shows that there is statistical evidence that ALS-Q Learning is better than ALS-LA when using only one agent to solve the problem. Regarding the use of two or more agents to solve the problem, at 75.00% of the

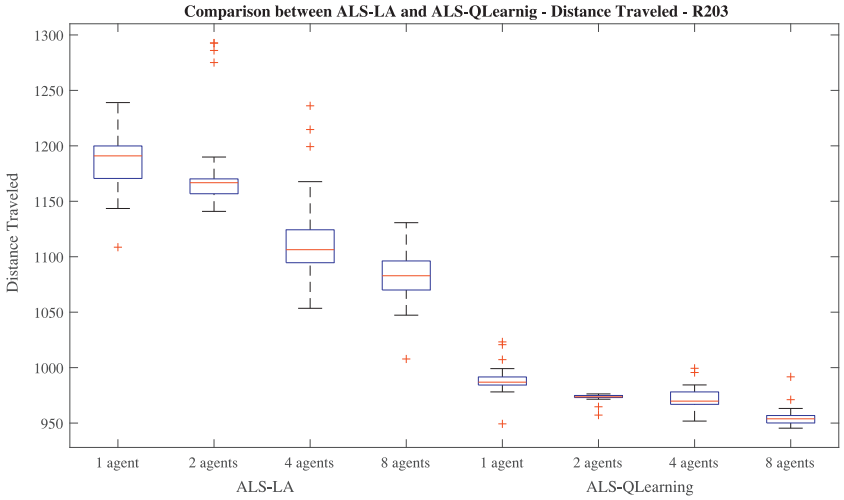


Fig. 24. Comparison between ALS-LA and ALS-QLearning proposals in relation to the distance traveled – R203 instance.

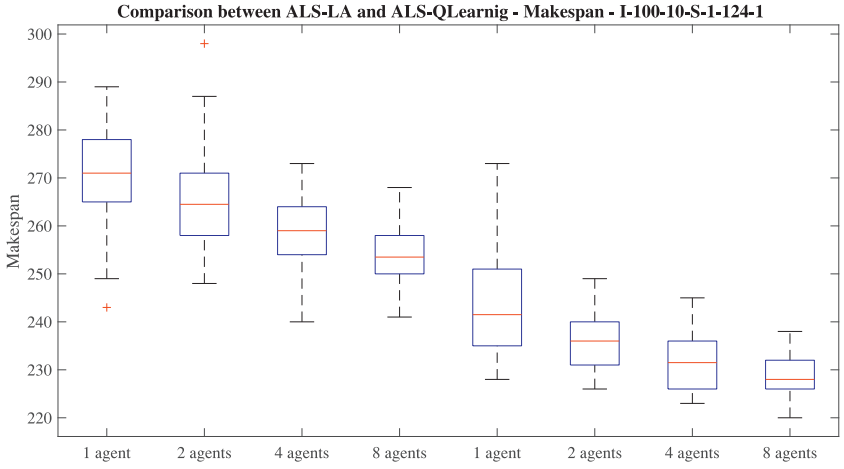


Fig. 25. Comparison between ALS-LA and ALS-QLearning proposals in relation to the makespan value – I_100_10_S_1-124_1 instance.

Table 10
Number of instances where the ALS-QLearning proposal is better than the ALS-LA proposal, per scenario, considering the total makespan UPMSP-ST.

Set of instances	Total instances per set	Scenarios			
		1 agent	2 agents	4 agents	8 agents
50 jobs	**16**	12	10	11	12
100 jobs	**8**	8	8	8	8
Total of instances	**24**	**20**	**18**	**19**	**20**

Table 11
Number of times ALS-QLearning is better than ALS-LA individually and as a team for UPMSP-ST.

Set of instances	Total of instances per set	Scenarios			
		Individually		In group	
		Value	%	Value	%
50 jobs	**16**	12	75.00	10	62.50
100 jobs	**8**	8	100.0	8	100.0
Total	**24**	**20**	**83.33**	**18**	**75.00**

used instances the parametric test shows that there is statistical evidence that ALS-QLearning is better than ALS-LA.

Fig. 26 presents an example of the makespan obtained in the 30 executions of the instance I_100_10_S_1-9_1, for the four scenarios used in each of the tested proposals (ALS-LA and ALS-QLearning). As in VRPTW, it is important to note how the values achieved by ALS-QLearning are better than those found by ALS-LA, whether considering a single agent or considering two or more agents. In addition, the values improve as the number of agents involved in the search for the solution increases. In this case, the ALS-

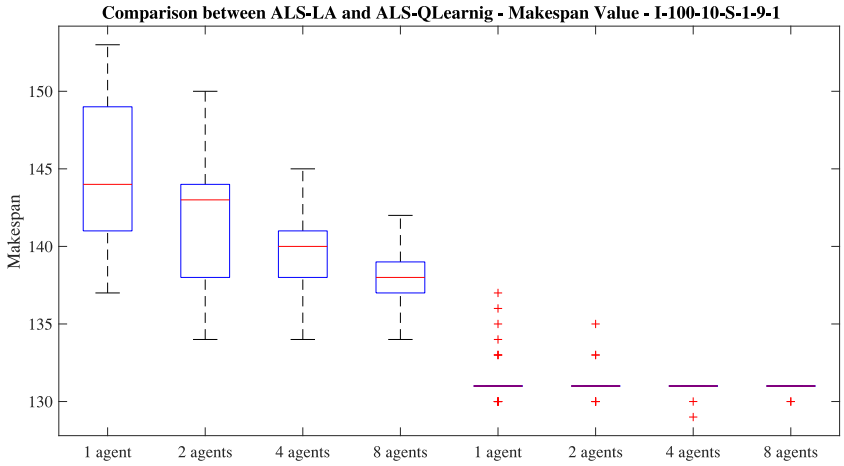


Fig. 26. Comparison between ALS-LA and ALS-QLearning proposals in relation to the makespan value – I_100_10_S_1-9_1 instance.

Table 12
Average costs of the ALS-AL proposal for VRPTW.

Class	BKS		ALS-AL							
			1 Agent		2 Agents		4 Agents		8 Agents	
	Cost	NR	Cost	NR	Cost	NR	Cost	NR	Cost	NR
C1	7455.42	90.00	11468.59	95,00	11106,00	93.73	10867.57	92.5	10722.16	93.57
C2	4718.87	24.00	5963.84	25,00	5760.4	24.57	5462.43	24,00	5336.65	24,00
R1	14524.02	143.00	16372.98	162.6	16303.79	162.67	16448.36	158.6	16051.72	158.93
R2	10461.33	30.00	12729.07	35.1	12400.8	34.17	12310.51	34.37	12000.78	34,00
RC1	11073.32	92.00	12565.04	107.47	12355.63	105.43	12325.24	106.33	12097.62	103.87
RC2	8953.92	26.00	11045.54	30.17	10931.74	29.47	10600.83	28.9	11220.26	28.47
TOTAL	**57186.88**	**405.00**	**70145.06**	**455.34**	**68858.36**	**450.04**	**68014.94**	**444.40**	**67429.19**	**442.84**

Table 13
Average costs of the ALS-QLearning proposal for VRPTW.

Class	BKS		ALS-QLearning							
			1 Agent		2 Agents		4 Agents		8 Agents	
	Cost	NR	Cost	NR	Cost	NR	Cost	NR	Cost	NR
C1	7455.42	90.00	7489.31	90.00	7459.83	90.00	7466.76	90.00	7446.39	90.00
C2	4718.87	24.00	4811.19	24.00	4720.42	24.00	4743,31	24.00	4716.08	24.00
R1	14524.02	143.00	14743.21	153.53	14566.80	152,63	14611.01	151.37	14567.00	150.70
R2	10461.33	30.00	10787.21	34.00	10567.68	33,57	10562.31	33.87	10522.76	33.03
RC1	11073.32	92.00	11179.67	102.80	11186.01	100,93	11153.62	97.87	11161.14	97.53
RC2	8953.92	26.00	9531.43	27.00	9353.25	27.00	9295.74	27.00	9316.87	27.00
TOTAL	**57186.88**	**405.00**	**58542.02**	**431.33**	**57853.99**	**428,13**	**57832.76**	**424.10**	**57730.25**	**422.27**

QLearning proposal found the best known result of the literature for all four scenarios.

6.4. Computational results for the average solution

For completeness, the computational results associated to the average solutions are shown.

Table 12 presents the average costs distance traveled (DT) and number of routes (NR) of the solutions obtained with the 30 executions of ALS-AL proposal. Table 13, on the other hand, shows the average costs of the solutions obtained with the 30 executions of ALS-QLearning proposal. In addition, Table 14 shows the average value of the makespan for the solutions obtained with the 30 executions with the ALS-AL proposal. In its turn, Table 15 includes the average value of the makespan for the solutions obtained with the 30 executions with the ALS-QLearning proposal.

From the analysis of these tables, some observations arose. The first observation, valid for both problems and proposals, is the effect of increasing the number of agents in use in the framework. There is a considerable improvement in the results obtained by increasing the number of agents in action in the framework. In the case of VRPTW, this fact is valid both with respect to the total number of vehicles and the total distance traveled, and regardless of the instance class evaluated. Clearly, therefore, there is an identification of a scalabilty effect on the number of agents. This effect, by the way, is also reported in applications of the A-Teams architecture (Barbucha et al., 2010).

The second observation is about the comparison between the two learning techniques and their results. Independently of the class of the problem evaluated, the great superiority of the results obtained with the ALS-QLearning proposal in relation to those obtained with the ALS-AL proposal, already previously proven in the statistical analysis, is also shown by the direct presentation of the results in these tables.

In addition, a noticeable issue is the results concerning the VRPTW instances from Table 13 are strongly competitive with respect to the total distance traveled, even though this is not a question put as a goal in the development of this framework at this moment. The total distance traveled found with the application

Table 14
Average costs of the ALS-AL proposal for UPMSP-ST.

Instance	BKS	Scenarios			
		1 agent	2 agents	4 agents	8 agents
I_50_10_S_1-9_1	**67.00**	81.15	77.70	75.65	73.55
I_50_10_S_1-49_1	**77.00**	92.80	88.25	87.55	85.70
I_50_10_S_1-99_1	**118.00**	131.85	128.10	123.05	123.25
I_50_10_S_1-124_1	**114.00**	132.00	130.34	124.40	121.50
I_50_15_S_1-9_1	**36.00**	46.50	43.85	43.35	42.35
I_50_15_S_1-49_1	**59.00**	70.30	68.40	67.00	66.70
I_50_15_S_1-99_1	**78.00**	87.05	85.40	82.45	80.45
I_50_15_S_1-124_1	**75.00**	92.85	90.30	84.85	81.25
I_50_20_S_1-9_1	**31.00**	40.15	37.90	36.45	35.25
I_50_20_S_1-49_1	**39.00**	63.65	60.65	60.80	57.15
I_50_20_S_1-99_1	**49.00**	68.05	65.35	65.55	65.60
I_50_20_S_1-124_1	**52.00**	71.70	68.05	66.15	66.50
I_50_25_S_1-9_1	**22.00**	26.60	24.90	23.55	22.60
I_50_25_S_1-49_1	**25.00**	49.70	50.20	46.30	44.95
I_50_25_S_1-99_1	**35.00**	60.30	59.50	56.40	51.05
I_50_25_S_1-124_1	**37.00**	62.30	59.40	59.60	57.20
I_100_10_S_1-9_1	**131.00**	144.73	141.80	139.90	137.87
I_100_10_S_1-49_1	**159.00**	201.07	196.77	191.40	188.33
I_100_10_S_1-99_1	**212.00**	266.53	256.83	253.30	246.33
I_100_10_S_1-124_1	**221.00**	270.87	265.30	258.83	254.23
I_100_15_S_1-9_1	**68.00**	81.77	81.20	79.07	77.80
I_100_15_S_1-49_1	**97.00**	137.37	132.40	128.80	123.47
I_100_15_S_1-99_1	**123.00**	167.50	159.27	153.70	151.70
I_100_15_S_1-124_1	**141.00**	189.07	179.20	174.13	169.40

Table 15
Average costs of the ALS-QLearning proposal for UPMSP-ST.

Instance	BKS	Scenarios			
		1 agent	2 agents	4 agents	8 agents
I_50_10_S_1-9_1	**67.00**	72.03	71.10	69.37	68.20
I_50_10_S_1-49_1	**77.00**	83.50	81.60	79.70	78.90
I_50_10_S_1-99_1	**118.00**	121.30	118.20	116.67	115.13
I_50_10_S_1-124_1	**114.00**	119.30	118.34	118.00	116.66
I_50_15_S_1-9_1	**36.00**	45.37	44.17	42.57	41.30
I_50_15_S_1-49_1	**59.00**	66.63	66.83	66.60	66.17
I_50_15_S_1-99_1	**78.00**	78.70	74.23	72.77	70.27
I_50_15_S_1-124_1	**75.00**	80.60	77.80	76.13	75.17
I_50_20_S_1-9_1	**31.00**	41.10	38.40	37.07	35.43
I_50_20_S_1-49_1	**39.00**	61.43	60.50	58.67	56.37
I_50_20_S_1-99_1	**49.00**	65.37	65.50	65.53	64.63
I_50_20_S_1-124_1	**52.00**	66.23	65.90	66.13	65.53
I_50_25_S_1-9_1	**22.00**	27.87	25.87	23.57	22.67
I_50_25_S_1-49_1	**25.00**	50.43	48.23	44.27	42.20
I_50_25_S_1-99_1	**35.00**	59.37	56.10	54.30	50.17
I_50_25_S_1-124_1	**37.00**	60.93	60.27	56.53	54.83
I_100_10_S_1-9_1	**131.00**	131.60	131.13	130.90	130.93
I_100_10_S_1-49_1	**159.00**	179.83	175.53	172.27	170.63
I_100_10_S_1-99_1	**212.00**	239.37	230.30	227.30	224.80
I_100_10_S_1-124_1	**221.00**	243.07	236.00	231.77	229.13
I_100_15_S_1-9_1	**68.00**	75.43	75.57	73.37	72.43
I_100_15_S_1-49_1	**97.00**	116.27	115.50	112.47	108.63
I_100_15_S_1-99_1	**123.00**	146.60	141.50	136.33	134.77
I_100_15_S_1-124_1	**141.00**	163.93	158.37	156.70	150.97

of the ALS-QLearning proposal is only 0.95% above the best value found in the literature for the VRPTW solution using metaheuristics. The total number of routes is 4.26% above (i.e., 17 routes) of the total number of routes associated with the best results in the literature.

7. Conclusions and future directions

This paper presented AMAM, a multi-agent framework for optimization using metaheuristics. Its main characteristic is to facilitate the hybridization of metaheuristics through a multi-agent structure. Each agent implements a heuristic / metaheuristic and the environment in which the agents act and dialogue is the search space of the combinatorial optimization being solved. Each agent act autonomously in this environment and interacts cooperatively with it and with the other agents. The interaction between the agents allows the metaheuristic hybridization. The latest version of the AMAM framework, released in January 2018, is available at https://github.com/mamelials/AMAM-Multiagente-Architecture-for-Metaheuristics, licensed under the GNU LGPLv3 license.

The main objective of this article was to propose new self-adaptive skills for the framework agents. Through these skills, the agents modify their actions based on the experience acquired in the interaction with the environment and with the other agents. The concepts of reinforcement learning, more specifically using the Q-Learning algorithm, are central in the definition of these new adaptive skills. The learning is used to select the application order of the neighborhood structures of the local search based on the VND heuristic.

In order to accomplish the validation of the AMAM framework with reinforcement learning, computational experiments were performed, using, as a case study for this purpose, the Vehicle Routing Problem with Time Window and Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times. The main objective of the experiments was to evaluate the performance of the adaptive agent. For this evaluation, the tests were made with the adaptive agent presented in the current proposal, called ALS-QLearning, and with the adaptive agent proposed in Silva et al. (2015), called ALS-LA.

The results obtained show that there are statistical evidences that the ALS-QLearning proposal obtained the best results in most instances and scenarios. In this way, the direct influence of the form of learning embedded in the agent is confirmed by the experiments, both from the individual point of view, and from the point of view of teamwork.

Additionally, for the two evaluated proposals (ALS-LA and ALS-QLearning), the scenarios with 2 or more agents were significantly higher in performance than the scenarios with 1 single agent. Thus, it is confirmed that the cooperation between the agents influences the quality of the solutions and the scalability of the framework, since, with the addition of new agents, there is an improvement in the results. The use of learning to assign adaptive capabilities to agents places the AMAM framework a step ahead of other frameworks in the literature as an alternative to the need to adapt the methods to specific aspects of the problem. In this way, it also makes possible future researches, such as the introduction of new forms of learning to improve adaptive capacities, as well as the study of the insertion of reinforcement learning methodologies among the agents.

Compliance and ethical standards

This study was funding by Brazilian agencies National Council of Technological and Scientific Development - CNPq (Grant 307915/2016-6) and Minas Gerais State Research Foundation - FAPEMIG (Grant PPM CEX 676/17). This study was financed in part by the Coordination for the Improvement of Higher Education Personnel (CAPES) - Brazil - Finance Code 001. Authors Maria Amélia Lopes Silva, Sérgio Ricardo de Souza Marcone Jamilson Freitas Souza and Ana Lúcia C. Bazzan declare that they have no conflict of interest. This article does not contain any studies with human participants or animals performed by any of the authors.

Credit authorship contribution statement

Maria Amélia Lopes Silva: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation,

Visualization, Writing - original draft, Writing - review & editing. **Sérgio Ricardo de Souza:** Conceptualization, Data curation, Methodology, Supervision, Writing - original draft, Writing - review & editing. **Marcone Jamilson Freitas Souza:** Conceptualization, Data curation, Funding acquisition, Methodology, Supervision, Writing - original draft, Writing - review & editing. **Ana Lúcia C. Bazzan:** Conceptualization, Methodology, Writing - original draft, Writing - review & editing.

Acknowledgments

The authors would also like to thank the Coordination for the Improvement of Higher Education Personnel (CAPES), the Minas Gerais State Research Foundation (FAPEMIG), the National Council of Technological and Scientific Development (CNPq), the Federal Center of Technological Education of Minas Gerais (CEFET-MG), the Federal University of Ouro Preto (UFOP), the Federal University of Viçosa (UFV) and the Federal University of Rio Grande do Sul (UFRGS) for supporting the development of the present study.

References

Alba, E., Luque, G., Garcia-Nieto, J., Ordonez, G., & Leguizamon, G. (2007). MALLBA: A software library to design efficient optimisation algorithms. *International Journal of Innovative Computing and Applications, 1*(1), 74–85.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research, 246*(2), 345–378. doi:10.1016/j.ejor.2015.04.004.

Allahverdi, A., Ng, C., Cheng, T., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research, 187*(3), 985–1032. doi:10.1016/j.ejor.2006.06.060.

Applegate, D., Bixby, R., Chvátal, V., & Cook, W. (2007). The traveling salesman problem: A Computational study. *Princeton series in applied mathematics* (2nd ed.). Princeton University Press.

Barbucha, D., Czarnowski, I., Jędrzejowicz, P., Ratajczak-Ropel, E., & Wierzbowska, I. (2010). JABAT middleware as a tool for solving optimization problems. In N. T. Nguyen, & R. Kowalczyk (Eds.), *Transactions on computational collective intelligence ii*. In *Lecture Notes in Computer Science: 6450* (pp. 181–195). Berlin, Heidelberg: Springer.

Bellifemine, F., Poggi, A., & Rimassa, G. (2007). *Developing multi-agent systems with JADE*. John Wiley & Sons.

Bellman, R. (1957). *Dynamic programming*. Princeton, NJ, USA: Princeton University Press.

Bertsekas, D. (1987). *Dynamic programming: Deterministic and stochastic models*. Englewood Cliffs, NJ: Prentice-Hall.

Blum, C., Puchinger, J., Raidl, G. R., & Roli, A. (2011). Hybrid metaheuristics in combinatorial optimization: a survey. *Applied Soft Computing, 11*(6), 4135–4151.

Blum, C., & Roli, A. (2003). Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys, 35*(3), 268–308.

Cahon, S., Melab, N., & Talbi, E.-G. (2004). Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics, 10*(3), 357–380.

Coelho, I. M., Munhoz, P. L. A., Haddad, M. N., Coelho, V. N., Silva, M. M., Souza, M. J. F., & Ochi, L. S. (2011). OptFrame: A computational framework for combinatorial optimization problems. In *Proc. of the VII alioeuro workshop on applied combinatorial optimization* (pp. 51–54). ALIO/EURO 2011.

Cotta, C., Talbi, E.-G., & Alba, E. (2005). Parallel hybrid metaheuristics. In E. Alba (Ed.), *Parallel metaheuristics: a new class of algorithms* (pp. 347–370). John Wiley & Sons.

Dorigo, M., Caro, G. D., & Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life, 5*(2), 137–172. doi:10.1162/106454699568728.

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66. doi:10.1109/4235.585892.

Dorigo, M., & Stützle, T. (2019). Ant colony optimization: Overview and recent advances. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of metaheuristics* (pp. 311–351)). Springer International Publishing. doi:10.1007/978-3-319-91086-4_10.

Durillo, J. J., & Nebro, A. J. (2011). Jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software, 42*(10), 760–771.

Fernandes, F. C., de Souza, S. R., Silva, M. A. L., Borges, H. E., & Ribeiro, F. F. (2009). A multiagent architecture for solving combinatorial optimization problems through metaheuristics. In *Proceedings of the 2009 IEEE international conference on systems, man and cybernetics (SMC 2009)* (pp. 3071–3076). doi:10.1109/ICSMC.2009.5345934.

Fink, A., & Voß, S. (2002). Hotframe: A heuristic optimization framework. In S. Voß, & D. L. Woodruff (Eds.), *Optimization software class libraries*. In *Operations Research/Computer Science Interfaces Series: 18* (pp. 81–154). Springer US.

Gambardella, L. M., & Dorigo, M. (1995). Ant-q: A reinforcement learning approach to the traveling salesman problem. In *Proceedings of the twelfth international conference on international conference on machine learning*. In *ICML'95* (pp. 252–260). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.. URL: http://dl.acm.org/citation.cfm?id=3091622.3091654.

Gaspero, L. D., & Schaerf, A. (2003). EASYLOCAL++: An object-oriented framework for the flexible design of local-search algorithms. *Software: Practice and Experience, 33*(8), 733–765.

(2010). Handbook of metaheuristics. In M. Gendreau, & J.-Y. Potvin (Eds.). *Internationalseries in operations research & management science*: 146 (2nd ed.). Springer.

Juan, A. A., Faulin, J., Grasman, S. E., Rabe, M., & Figueira, G. (2015). A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. *Operations Research Perspectives, 2*, 62–72.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research, 4*, 237–285.

Kazemitabar, S. J., Taghizadeh, N., & Beigy, H. (2018). A graph-theoretic approach toward autonomous skill acquisition in reinforcement learning. *Evolving Systems, 9*(3), 227–244.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 – international conference on neural networks: 4* (pp. 1942–1948). doi:10.1109/ICNN.1995.488968.

Li, X., Epitropakis, M. G., Deb, K., & Engelbrecht, A. (2017). Seeking multiple solutions: An updated survey on niching methods and their applications. *IEEE Transactions on Evolutionary Computation, 21*(4), 518–538.

Liefooghe, A., Jourdan, L., & Talbi, E. (2011). A software framework based on a conceptual unified model for evolutionary multiobjective optimization: paradisEO–MOEO. *European Journal of Operational Research, 209*(2), 104–112.

Lotfi, N., & Acan, A. (2015). Learning-based multi-agent system for solving combinatorial optimization problems: A new architecture. In E. Onieva, I. Santos, E. Osaba, H. Quintián, & E. Corchado (Eds.), *Hybrid artificial intelligent systems: Proceedings of the 10th international conference (HAIS 2015)*. In *Lecture notes in computer science: 9121* (pp. 319–332). Springer International Publishing. doi:10.1007/978-3-319-19644-2_27.

Lourenço, H. R., Martin, O. C., & Stützle, T. (2003). Iterated local search. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of metaheuristics*. In *International series in operations research & management science: 57* (pp. 321–353). Kluwer Academic Publishers.

Martin, S., Ouelhadj, D., Beullens, P., Ozcan, E., Juan, A. A., & Burke, E. K. (2016). A multi-agent based cooperative approach to scheduling and routing. *European Journal of Operational Research, 254*(1), 169–178.

Meignan, D., Créput, J.-C., & Koukam, A. (2008). An organizational view of metaheuristics. In N. Jennings, A. Rogers, A. Petcu, & S. D. Ramchurn (Eds.), *First international workshop on optimisation in multi-agent systems, AAMAS'08* (pp. 77–85).

Melab, N., Luong, T. V., Boufaras, K., & Talbi, E. (2013). ParadisEO-MO-GPU: A framework for parallel GPU-based local search metaheuristics. In *Proceedings of the 15th annual conference on genetic and evolutionary computation* (pp. 1189–1196). ACM.

Milano, M., & Roli, A. (2004). MAGMA: A multiagent architecture for metaheuristics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 34*(2), 925–941.

Mladenović, N., & Hansen, P. (1997). Variable neighbourhood search. *Computers & Operations Research, 24*(11), 1097–1100.

Narendra, K. S., & Thathachar, M. A. L. (1974). Learning automata – a survey. *IEEE Transactions on Systems, Man, and Cybernetics, SMC-4*(4), 323–334. doi:10.1109/TSMC.1974.5408453.

Noel, M. M., & Pandian, B. J. (2014). Control of a nonlinear liquid level system using a new artificial neural network based reinforcement learning approach. *Applied Soft Computing, 23*, 444–451.

Parejo, J. A., Ruiz-Cortés, A., Lozano, S., & Fernandez, P. (2012). Metaheuristic optimization frameworks: A survey and benchmarking. *Soft Computing, 16*(3), 527–561.

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York, NY: John Wiley & Sons.

Queiroz dos Santos, J. P., de Melo, J. D., Neto, A. D. D., & Aloise, D. (2014). Reactive search strategies using reinforcement learning, local search algorithms and variable neighbourhood search. *Expert Systems with Applications, 41*(10), 4939–4949.

Rabadi, G., Moraga, R. J., & Al-Salem, A. (2006). Heuristics for the unrelated parallel machine scheduling problem with setup times. *Journal of Intelligent Manufacturing, 17*(1), 85–97.

Radac, M.-B., & Precup, R.-E. (2018). Data-driven model-free slip control of anti-lock braking systems using reinforcement q-learning. *Neurocomputing, 275*, 317–329.

Radac, M.-B., Precup, R.-E., & Roman, R.-C. (2018). Data-driven model reference control of mimo vertical tank systems with model-free vrft and q-learning. *ISA Transactions, 73*, 227–238.

Salgado, M., & Clempner, J. B. (2018). Measuring the emotional state among interacting agents: A game theory approach using reinforcement learning. *Expert Systems with Applications, 97*, 266–275.

Samma, H., Lim, C. P., & Saleh, J. M. (2016). A new reinforcement learning-based memetic particle swarm optimizer. *Applied Soft Computing, 43*, 276–297.

Silva, M. A. L. (2007). *Modelagem de uma arquitetura multiagente para a solução, via metaheurísticas, de problemas de otimização combinatória (in portuguese)*. Belo Horizonte, Brazil: Federal Center of Technological Education of Minas Gerais (CEFET-MG) Master's thesis.

Silva, M. A. L., de Souza, S. R., de Oliveira, S. M., & Souza, M. J. F. (2014). An agent-based metaheuristic approach applied to the vehicle routing problem with time-windows. In *Proceedings of the 2014 brazilian conference on intelligent systems - enc. nac. de inteligência artificial e computacional (BRACIS-ENIAC 2014)*. São Carlos, SP, Brazil.

Silva, M. A. L., de Souza, S. R., Souza, M. J. F., & de Oliveira, S. M. (2015). A multi-agent metaheuristic optimization framework with cooperation. In *2015 brazilian conference on intelligent systems (BRACIS)* (pp. 104–109). doi:10.1109/BRACIS.2015.64. Natal, Brazil.

Silva, M. A. L., de Souza, S. R., Souza, M. J. F., & de França Filho, M. F. (2018). Hybrid metaheuristics and multi-agent systems for solving optimization problems: A review of frameworks and a comparative analysis. *Applied Soft Computing, 71*, 433–459. doi:10.1016/j.asoc.2018.06.050.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research, 2*(35), 254–264.

Subramanian, A., Drummond, L. M., Bentes, C., Ochi, L. S., & Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computer & Operations Research, 37*(11), 1899–1911.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (1st). Cambridge, MA, USA: MIT Press.

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation* (1st). John Wiley & Sons.

Talukdar, S., Baerentzen, L., Gove, A., & Souza, P. D. (1998). Asynchronous teams: Co-operation schemes for autonomous agents. *Journal of Heuristics, 4*(4), 295–321.

Talukdar, S., & Souza, P. S. (1990). Asynchronous teams. In *Proceedings of the second SIAM conference on linear algebra: Signals, system and control*. San Francisco, USA.

Toth, P., & Vigo, D. (2002). *The vehicle routing problem*. Philadelphia, USA: SIAM - Society for Industrial and Applied Mathematics.

Toth, P., & Vigo, D. (2014). *Vehicle routing: Problems, methods, and applications* (2nd ed.). Philadelphia, PA, USA: SIAM - Society for Industrial and Applied Mathematics.

Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research, 211*(3), 612–622. doi:10.1016/j.ejor.2011.01.011.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Cambridge, England: University of Cambridge Ph.D. thesis.

Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning, 8*(3), 279–292.