Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 707056, 12 pages
http://dx.doi.org/10.1155/2015/707056

Research Article

A Hyperheuristic for the Dial-a-Ride Problem with Time Windows

Enrique Urra,[1] Claudio Cubillos,[1] and Daniel Cabrera-Paniagua[2]

[1]*Escuela de Ingeniería Informática, Pontificia Universidad Católica de Valparaíso, Avenida Brasil 2950, 2340025 Valparaíso, Chile*
[2]*Escuela de Ingeniería Comercial, Universidad de Valparaíso, Pasaje La Paz 1301, 2531075 Viña del Mar, Chile*

Correspondence should be addressed to Enrique Urra; enrique.urra@gmail.com

Received 25 September 2014; Accepted 15 December 2014

Academic Editor: Haipeng Peng

The dial-a-ride problem with time windows (DARPTW) is a combinatorial optimization problem related to transportation, in which a set of customers must be picked up from an origin location and they have to be delivered to a destination location. A transportation schedule must be constructed for a set of available vehicles, and several constraints have to be considered, particularly time windows, which define an upper and lower time bound for each customer request in which a vehicle must arrive to perform the service. Because of the complexity of DARPTW, a number of algorithms have been proposed for solving the problem, mainly based on metaheuristics such as Genetic Algorithms and Simulated Annealing. In this work, a different approach for solving DARPTW is proposed, designed, and evaluated: hyperheuristics, which are alternative heuristic methods that operate at a higher abstraction level than metaheuristics, because rather than searching in the problem space directly, they search in a space of low-level heuristics to find the best strategy through which good solutions can be found. Although the proposed hyperheuristic uses simple and easy-to-implement operators, the experimental results demonstrate efficient and competitive performance on DARPTW when compared to other metaheuristics from the literature.

1. Introduction

The dial-a-ride problem with time windows (DARPTW) [1] is known in the literature as a complex combinatorial optimization problem related to transportation, in which a set of customers must be picked up from an origin location and they must be delivered to a destination location. For achieving this, a set of vehicles are available, and a transportation schedule must be constructed for each one, which should be subject to several constraints. In the time-window-free version of the problem (DARP), the vehicles have freedom for defining the time at which customers are picked up/delivered, but under the time-window version (DARPTW, the one considered in this research) a vehicle schedule must assure that the customer is served in a restricted time range: the time windows (TW) itself. That constraint adds an important complexity degree to the problem, which can be proven to be *NP*-hard [2]. DARPTW comes from a family of pickup-and-delivery problems that originates from the travel salesman problem (TSP) [3]. While most of them must deal with objects, in DARPTW people must be transported; therefore, the problem evaluation is closely related with quality of service issues; for example, the total time a customer remains onboard a vehicle should not be excessive.

The solution space of the DARPTW problem is particularly challenging for any automated solving mechanism, because small changes in the solution structure could lead to completely infeasible solutions. For example, if a client is moved to a different vehicle schedule, a complete restructure of the latter is required, and it is highly probable that previous constraints that were fulfilled are now violated. Because of this and considering the high number of involved variables, common solutions in the literature for the DARPTW are based on heuristic methods. For example, Genetic Algorithms (GA) is a metaheuristic approach used in several works. In [4], a first bit-solution based variant was evaluated, and considering several feasibility problems involved in this implementation, an improved integer-based representation and more specialized operators were tested, which allowed converging towards feasible and better solutions.

In [2], a classical *cluster-first, route-second* approach was implemented, in which *clustering* is the process of assigning customers to vehicles and *routing* is the process of defining the order of the pickup and delivery of customers. The GA was used exclusively for the clustering process. In a previous work [5], a GA that uses preprocessing mechanisms for reducing the search space complexity was implemented, allowing executing more efficiently specialized solution-modification operators alongside the genetic operators. Another novel contribution of the latter research was implementing the GA for solving both scheduling components of the solutions: the routing and the clustering. A different metaheuristic used for solving DARPTW is Simulated Annealing (SA); for example, in [6] this technique was mixed with other specialized smaller heuristics, generating an efficient and stable approach that particularly improved quality of service issues. In [7], a multiobjective SA algorithm was implemented and embedded into a multiagent system to solve the dynamic version of the problem.

Among heuristic methods, metaheuristics algorithms are the most used techniques through which problems as DARPTW are solved. The prefix "meta" is because they define an abstract framework whose components must be adapted to the solved problem by including extensive problem knowledge within the operations of the algorithm [8]. This adaptation process allows a metaheuristic algorithm to perform efficiently for the target problem, but the main trade-off is a costly implementation process that can be eventually infeasible for real production environments. Under this scenario, a new type of heuristic method named *hyperheuristics* [9–11] appears in the optimization research field as a more balanced alternative in which, rather than adapting the main search mechanisms, these ones are encapsulated in a high-level layer that can be reused among different problems or families of problems. For achieving this idea at the design level, a hyperheuristic requires several low-level heuristics provided by the problem domain, that is, very simple and specialized operators. A hyperheuristic uses these low-level heuristics for searching an efficient search strategy by leveraging their combined behavior in different manners.

Many applications of hyperheuristic solvers were developed during the last decade [12]; however, no implementations for the DARPTW are currently provided to our knowledge. Only the work in [13] addresses DARP instances by using a hyperheuristic approach; however, the tackled problem does not consider time windows, and then it corresponds to a simplification of the problem instance considered in this work. Also in [14] a hyperheuristic was implemented for solving the VRP problem in its dynamic version.

In this work, a new hyperheuristic approach for solving the DARPTW is presented, which is based in an adapter layer that allows the interaction between both the hyperheuristic and the DARPTW domain. There are three main contributions related to the main research product of this work: (i) it represents the first approach for solving DARPTW under a hyperheuristic architecture, (ii) the hyperheuristic architecture implemented in this work may be reused for evaluating different transportation problems, which can be part of the same family of DARPTW, and (iii) although very basic hyperheuristic operators and DARPTW low-level heuristic were used, the obtained results are competitive and efficient regarding other heuristic approaches in the literature, which demonstrates the convenience of dynamically mixing low-level operators through high-level strategies even if all of them are very simple.

This paper is structured as follows. In Section 2, a general background on hyperheuristics is presented, as well as the main motivations behind the concept. In Section 3, the DARPTW problem is described in detail, which includes its mathematical definition. In Section 4, the overall design of proposed solution is presented, which includes the main algorithmic structures of the implemented solver. In Section 5, the experimental design and the obtained results are described and discussed. Finally, in Section 6, several conclusions regarding this work are outlined.

2. Hyperheuristics

The term heuristic has a broad significance in different areas of science. The concerns of this research reside on optimization, in which heuristics are associated with computational mechanisms that perform a search over the solution space of a problem to be solved, in the hope to find the best (if not the optimum) solution. The complexity of the strategy used in the search ranges from simpler algorithms to intelligent learning techniques. Different heuristic classifications have been emerged according to such complexity; for example, metaheuristics [15, 16] are algorithmic frameworks those search strategy is based on a set of subordinate and very specific operators, for example, simpler heuristics. Metaheuristics are *made-to-measure* techniques [11], because they include extensive problem domain knowledge in their design. This makes their applicability to resource-constrained scenarios harder, which are common in real industry settings. In response, the hyperheuristic concept has emerged as a relatively new heuristic technology. The search strategy of hyperheuristics can be considered as a process of *using heuristics to choose heuristics* to solve the problem in hand [17]; however, newer approaches even consider the *automated generation of heuristics* [9].

Hyperheuristics operate at a higher level than metaheuristics, because they often have no knowledge of the problem domain. To accomplish this, a hyperheuristic is commonly structured as Figure 1 shows. There is a high-level domain where the hyperheuristic itself resides. Also, there is a low-level domain, which can be considered the problem domain itself. In the latter, a collection of simple and highly specialized *low-level heuristics* reside, which are based on rich problem knowledge to operate. At different stages, the hyperheuristic selects and applies a low-level heuristic using some decision criteria to move from the current problem state to another. This process allows associating each heuristic, or even a combination of them, with the problem conditions and state. Because of this, in the literature it is often said that hyperheuristics *perform the search over a virtual solution space of low-level heuristics*, unlike simple heuristics or metaheuristics, which perform the search directly over the problem space. The *domain barrier* is

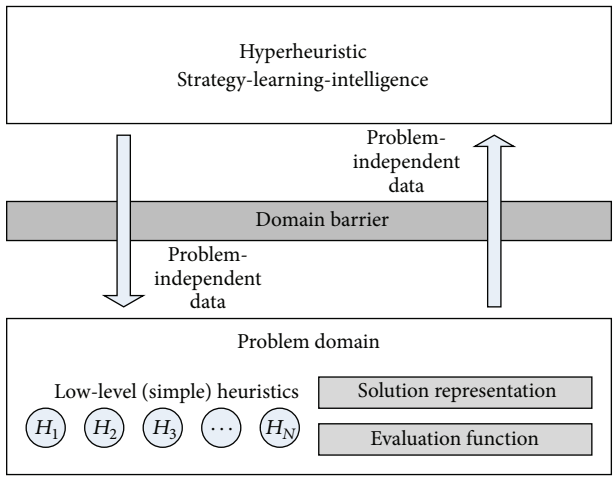


Figure 1: The generic hyperheuristic framework.

the key component of the framework, which prevents passing problem knowledge from the low-level to the high-level. Therefore the hyperheuristic only knows about the existence of the low-level heuristics, but only as if they were *black-boxes*. The logic which they use to perform some operation in the problem domain is completely encapsulated, and only *problem-independent data* is transferred between the layers, such as solution quality values and CPU times, which allows the hyperheuristic to make decisions and store useful search information.

As all the search complexity and intelligence are already implemented in the hyperheuristic, they can be directly reused and the problem domain expert does not even require knowing such mechanisms. This enables higher abstraction capabilities than metaheuristics, but it could also involve a trade-off between generality and efficiency. In general, it makes sense that highly specialized and intelligent search methods could perform better than intelligent but more generic ones. However, it is important to emphasize that the main hyperheuristic features are focused on their applicability in resource-limited scenarios, the ones in which *good-enough, soon-enough, cheap-enough* solutions are adequate [17].

It is said that hyperheuristics perform a search over a space of heuristics rather than a space of problem solutions [18]. For achieving this behavior, a hyperheuristic commonly uses two operator types: heuristic selection, which allows selecting and executing low-level heuristics, and *move acceptance*, which allows or denies movements through the search space by using some criteria. These operators are key components in the hyperheuristic search behavior. In the literature, a number of approaches can be found, from deterministic mechanisms to sophisticated learning approaches. In the following, a set of simple strategies are described, which appear in [11, 19–21], but they are also common in other publications.

(i) Basic heuristic selection operators are as follows.

(a) *Simple random*: select the next heuristic at random using a uniform probability distribution.

(b) *Random descent*; it works similar to *simple random* but applies the selected heuristic until no further improvement is possible.

(c) *Random permutation*: select a random sequence of the low-level heuristics and apply them in order.

(d) *Random permutation descent*: it works similar to *random permutation* but the sequence is kept until no further improvement is possible.

(e) *Greedy*: all heuristics are applied, but the best performing one is selected alongside their effects in the solution.

(ii) Basic move acceptance operators are as follows.

(a) *All moves*: *all moves* are accepted, regardless of their effect.

(b) *Only improving*: *only improving* moves regarding previous one are accepted.

(c) *Improving and equal*: *only improving* or equals moves regarding previous one are accepted.

3. Dial-a-Ride Problem with Time Windows

DARPTW comes from a family of transportation problems that originate from the TSP. It is known as a multiobjective transportation problem, in which two relevant elements must be minimized simultaneously: the customer inconvenience (individuals are transported) and the transportation costs. In the literature, there are many versions and specifications for the problem because of the variety of constraints. This work is based on the approach in [2], in which some complexities of the mathematical model are addressed by constraint relaxation.

In DARPTW, there is a set of n customer transportation requests, each one associated with a pickup location i and a drop-off one $n + i$. For each request, a time window for the pickup $[a_i, b_i]$ and one for the drop-off $[a_{n+i}, b_{n+i}]$ are generated, based on some customer preferences and the system configuration. There is a set of m vehicles available for transporting the customers, each one with a fixed capacity C. The goal is to generate a schedule in which each customer is transported by some vehicle while addressing several constraints, which include the arriving and departure at the specified time windows. In this work, a *single depot scenario* was considered, in which each vehicle k starts and ends its schedule at times T_s^k and T_e^k, respectively, both at a particular depot location d. The following sets are defined:

(i) $P = \{1, \ldots, n\}$: set of pickup locations,

(ii) $D = \{n + 1, \ldots, 2n\}$: set of delivery locations,

(iii) $N = P \cup D$: set of pickup and delivery locations,

(iv) K: set of vehicles,

(v) $V \subset K$: set of vehicles used in solution,

(vi) $A = N \cup \{d\}$: set of all possible stopping locations for all vehicles.

Several additional parameters are considered:

(i) s_i: the service time needed at location i,

(ii) $t_{i,j}$: the traveling time or distance from location i to j,

(iii) l_i: the change in vehicle load at location i,

(iv) MRD: the maximum route duration, for example, how long a vehicle schedule can be, from the starting time to the ending,

(v) MRT: the maximum ride time, for example, how long the difference between the pickup and the delivery times can be for a single client.

The following decision variables are used:

(i) $x_{i,j}^k$: a decision variable with value 1 if the vehicle k services a customer at location i and the next customer at location j, and 0 otherwise,

(ii) T_i^k: time at which vehicle k starts its service at location i,

(iii) L_i^k: load of vehicle k after servicing location i,

(iv) W_i^k: waiting time of vehicle k before servicing location i. This time is commonly generated when the vehicle arrives to i before the low-bound of the related time window.

The objective function will be weighted by using the following weights:

(i) w_1: weight on transport time, that is, the total time used by vehicles for moving between locations,

(ii) w_2: weight on excess ride time, that is, difference between the actual time at which the customer arrives to their destination location and the time at which the customer would have reached their destination location if the vehicle transported him directly to its delivery location,

(iii) w_3: weight on waiting time for customers, that is, total time spent by customers on board a stopped vehicle.

(iv) w_4: weight on route duration, that is, difference between the start and the end of vehicles' schedules,

(v) w_5: weight on time-window violation, that is, penalty applied when a vehicle arrives too early or too late at a location,

(vi) w_6: weight on excess of maximum ride time, that is, penalty applied when MRT is violated,

(vii) w_7: weight on excess of route duration, that is, penalty applied when MRD is violated.

Considering the above specifications, the mathematical model of DARPTW can be defined as follows:

$$\begin{aligned}\min \quad & w_1 \sum_{k\in V} \sum_{i,j\in A} t_{i,j} x_{i,j}^k \\ & + w_2 \sum_{k\in V} \sum_{i\in P} \left(T_{n+i}^k - s_i - T_i^k - t_{i,n+i}\right) \\ & + w_3 \sum_{k\in V} \sum_{i\in N} W_i^k \left(L_i^k - l_i\right) \\ & + w_4 \sum_{k\in V} \left(T_e^{\mathrm{k}} - T_s^k\right) \\ & + w_5 \sum_{k\in V} \sum_{i\in A} \max\left(0, a_i - T_i^k, T_i^k - b_i\right) \\ & + w_6 \sum_{k\in V} \sum_{i\in P} \max\left(0, \left(T_{n+i}^k - T_i^k\right) - \mathrm{MRT}\right) \\ & + w_7 \sum_{k\in V} \max\left(0, \left(T_e^k - T_s^k\right) - \mathrm{MRD}\right)\end{aligned} \tag{1}$$

$$\text{subject to} \quad \sum_{k\in V} \sum_{j\in P} x_{d,j}^k = m \tag{2}$$

$$\sum_{k\in V} \sum_{i\in D} x_{i,d}^k = m \tag{3}$$

$$\sum_{j\in A} x_{i,j}^k - \sum_{j\in A} x_{j,i}^k = 0 \quad \forall k \in V,\ i \in N \tag{4}$$

$$\sum_{k\in V} \sum_{j\in N} x_{i,j}^k = 1 \quad \forall i \in P \tag{5}$$

$$\sum_{j\in N} x_{i,j}^k - \sum_{j\in N} x_{j,n+i}^k = 0 \quad \forall k \in V,\ i \in P \tag{6}$$

$$x_{i,j}^k \left(T_i^k + s_i + t_{i,j} + W_j^k - T_j^k\right) \leqslant 0 \quad \forall k \in V,\ i, j \in A \tag{7}$$

$$T_i^k + s_i + t_{i,n+i} + W_j^k - T_{i+n}^k \leqslant 0 \quad \forall k \in V,\ i \in P \tag{8}$$

$$x_{i,j}^k \left(L_i^k + l_j - L_j^k\right) = 0 \quad \forall k \in V,\ i, j \in A \tag{9}$$

$$l_i \leqslant L_i^k \leqslant C \quad \forall k \in V,\ i \in P \tag{10}$$

$$L_d^k = 0 \quad \forall k \in \mathrm{V} \tag{11}$$

$$x_{i,j}^k \in \{0, 1\} \quad \forall k \in K,\ i, j \in A \tag{12}$$

$$T_i^k \geqslant 0 \quad \forall k \in K,\ i \in V \tag{13}$$

$$L_i^k \geqslant 0 \quad \forall k \in K,\ i \in V \tag{14}$$

$$W_i^k \geqslant 0 \quad \forall k \in K,\ i \in V. \tag{15}$$

The constraints defined above can be classified according to the following criteria.

```
(1)  function hyperheuristic( )
(2)      initialize problem domain and hyper-heuristic
(3)      run H_init and store its result in S_accepted
(4)      set S_best = S_accepted, S_candidate = null
(5)      while stop conditions have not been met
(6)              S_candidate = heuristic_selection (S_accepted)
(7)              if move_acceptance (S_candidate) = true
(8)                  set S_accepted = S_candidate
(9)              end if
(10)             if S_candidate > S_best
(11)                 set S_best = S_candidate
(12)             end if
(13)     end while
(14)     return S_best
(15) end function
```

Listing 1: The pseudocode of the hyperheuristic algorithm implemented.

(i) *Depot constraints*: equations (2) and (3) force a vehicle to start and finish its schedule in the depot location.

(ii) *Routing constraints*: equation (4) ensures that, for each location, there is an equal number of vehicles arriving and leaving. Equation (5) ensures that exactly one vehicle serves each pickup location, and (6) forces the same vehicle to serve both the pickup and delivery locations, for any customer.

(iii) *Precedence constraints*: equation (7) ensures that the arrival time at any location is greater than the leaving time of the previous location in the route. Equation (8) ensures that the pickup location of any customer is visited before its delivery location.

(iv) *Vehicle load constraints*: equation (9) forces that, for each vehicle, the same quantity of pickups and deliveries is performed within a route, (10) ensures that vehicle capacity is never exceeded, and (11) forces that all vehicles must be empty when starting and ending their schedules.

4. Solution Design

Listing 1 shows the pseudocode of a hyperheuristic solver implemented in this work. In lines (2–4), the process is initialized, which includes the execution of an initializing low-level heuristic H_{init} to generate the first solution from which the search starts. S_{accepted} is the current accepted solution in which the search process is focused, and S_{best} is the best solution found during all the process. $S_{\text{candidate}}$ is a candidate solution for acceptance evaluation. In lines (5–13), the main iteration is performed, which stops when particular conditions are met; that is, a number of iterations and/or an execution time are reached. In line (6), a heuristic selection operator is executed, which allows selecting and running low-level heuristics from the set of all available low-level heuristics, and uses the S_{accepted} as input. Depending on the operator behavior, more than one low-level heuristic may be executed, that is, a greedy approach on which the low-level heuristic that provides the best output result is selected. The result of the operator, which is stored in $S_{\text{candidate}}$, is used as input in line (7) to execute the move acceptance operator. The latter allows deciding if the new solution will be used as the next move within the solution space. If the solution is accepted, the S_{accepted} one is updated as well, leaving the previous current solution otherwise. Regardless of this acceptance result, in lines (10–12) the S_{best} solution is updated if corresponding. Finally, in line (14), the best result found is returned.

From an architectural perspective, this work was focused on the generation and usage of components that can be further modified and replaced for other problems and scenarios, and thereby, hyperheuristic background concepts could be leveraged. The most basic component used is the *hMod* framework, a heuristic design library that was initially presented in a previous work [23]. One of the main features of *hMod* is its wide coverage for different heuristic types and abstraction levels, thus, algorithms of different complexity can be implemented through the framework: from simpler heuristics to complex meta- and hyperheuristic solvers. For supporting this, *hMod* provides a *Step* interface, which represents a particular stage within an algorithm that performs some operation related to the complete process. A complete algorithm is defined by the sequential execution of chained steps. In this way, any algorithm implemented through the framework must start with a particular step, and this is the case of the hyperheuristic itself, and the low-level heuristics at problem domain. In the former, the *Step* interface is used for both referencing the low-level heuristics and implementing the high-level solver itself.

It is important to remark that a selected low-level heuristic is not directly executed by the hyperheuristic main algorithm or its common operators. Instead, a particular *low-level heuristic call* operator, whose pseudocode is presented in Listing 2, is used each time a low-level heuristic must be executed. This is necessary because intermediary tasks must

```
(1) function low_level_heuristic_call(H_selected, HLS_input)
(2)     LLS_decoded = decode(HLS_input)
(3)     download (LLS_decoded)
(4)     execute H_selected
(5)     LLS_result = upload( )
(6)     HLS_encoded = encode(LLS_result)
(7)     return HLS_encoded
(8) end function
```

Listing 2: The pseudocode of the *low-level heuristiccall* operator.

```
(1)  function greedy_heuristic_selection(S_accepted)
(2)      set S_best = null, S_result = null
(3)      for each low-level heuristic H_current in H
(4)          S_result = low_level_heuristic_call (H_current, S_accepted)
(5)          if S_best = null or S_best < S_result
(6)              set S_best = S_result
(7)          end if
(8)      end for
(9)      return S_best
(10) end function
```

Listing 3: The pseudocode of the *greedyheuristic selection* operator.

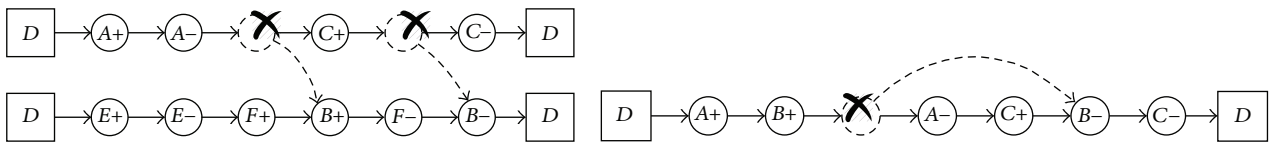
be executed alongside the low-level heuristic itself. The operator receives two arguments: the low-level heuristic to execute (H_{selected}) and the high-level solution to be used as input of the calling ($\text{HLS}_{\text{input}}$). To compatibilize the input solution with the low-level heuristic, a *decode* operation is called at line (2), which transforms $\text{HLS}_{\text{input}}$ to a low-level representation, storing it in $\text{LLS}_{\text{decoded}}$. To provide the decoded solution to the problem domain, a *download* operator is called at line (3), which takes such solution and puts it into the low-level data structures. H_{selected} is executed at line (4), assuming that the input low-level solution is available because of the previous procedures. After the low-level heuristic execution, an inverse process is performed to retrieve the result from the problem domain and to put the result into hyperheuristic structures. An *upload* operator is called at line (5), which retrieves the output low-level solution, storing it in $\text{LLS}_{\text{result}}$. An *encoder* operation is called at line (7) for converting $\text{LLS}_{\text{result}}$ into a high-level representation, which is stored in $\text{HLS}_{\text{encoded}}$. Finally, $\text{HLS}_{\text{encoded}}$ is returned as result of the low-level heuristic execution.

In this implementation, all the heuristic selection and move acceptance operators mentioned in Section 2 were implemented. In particular, all heuristic selection implementations involved at some point of their execution the call of a low-level heuristic through the mechanisms described above. In Listing 3, the pseudocode of the *greedy heuristic selection* operator is presented. All the heuristics in H, which correspond to the low-level heuristic set, are iterated and executed within the iteration at lines (3–8). At line (4), the *low-level heuristic call* operator is used as mentioned above. The greedy procedure always selects the best result obtained in each iteration, and the related evaluation is performed at lines (5–7).

Listing 4 presents the pseudocode of the *random permutation descent* operator. Originally, this operator is intended to execute a complete heuristic permutation $H(P) = H_1, H_2, H_3, \ldots, H_P$, with P the size of the permutation. When a H_i heuristic is executed in the permutation, it provides a result solution S_{result}, and exactly this solution is used as input of the next heuristic in permutation H_{i+1}. In preliminary evaluations of the operator, this behavior presented poor convergence due to the high possibility of disrupting a good-quality solution within the permutation. Because of this, in this work a modified version of the operator was implemented, in which the solution passed as input for H_{i+1} is S_{result}, only if the latter improves the previous solution used as input for H_i. This idea is reflected at lines (9–11) of the pseudocode, which compares the current input solution of the permutation S_{current} with the obtained solution after the low-level heuristic execution S_{check}. Only if S_{check} improves S_{current}, then S_{check} is assigned to S_{current}.

Another relevant element of the framework is the low-level heuristic set, which is more related to the problem domain. In this implementation, four different low-level heuristics were implemented.

(1) *Move random customer from route* (Figure 2(a)): it picks the events (pickup and delivery) of a random customer and moves them to the schedule of another vehicle, if it is possible.



(a) Move random customer from route

(b) Move random customer event within a route

Figure 2: Two most basic low-level heuristics implemented for DARPTW. A single vehicle schedule, or route, starts/ends with a leave/arrival to depot event D. Within a single route, several clients are transported ($A, B, C, \ldots$), and different pickup events ($A+, B+, C+, \ldots$) and delivery events ($A-, B-, C-, \ldots$) are performed. The heuristic (a) performs modifications at the customer-to-vehicle assignation level, while the heuristic (b) performs modifications at the pickup/delivery events ordering level.

```
(1)  function rpd_heuristic_selection(S_accepted)
(2)      set S_result = null
(3)      if P[] is empty
(4)              store a random permutation of H in P[]
(5)      end if
(6)      set S_current = S_accepted, S_check = null
(7)      for each low-level heuristic H_current in P[]
(8)              S_check = low_level_heuristic_call(H_current, S_current)
(9)              if S_check > S_current
(10)                 S_current = S_check
(11)             end if
(12)     end for
(13)     if S_result <= S_accepted
(14)             clear P[]
(15)     end if
(16)     return S_result
(17) end function
```

Listing 4: The pseudocode of the *random permutation descent* heuristic selection operator.

(2) *Move random customer event within a route* (Figure 2(b)): it picks a random event and moves it within the feasible bounds of the schedule of the vehicle that is currently serving the event.

(3) *Move customer from all routes* (generalization of 1): for each vehicle in solution, the *move random customer from route* is performed.

(4) *Move random customer event in all routes* (generalization of 2): for each vehicle in solution, the *move random customer event within a route* is performed.

5. Experiments, Results, and Discussion

In this work, a part of a benchmark dataset provided by Cordeau and Laporte in [22] was used, which is the same used in a number of works, such as [2, 5–7]. A description of the used sets is provided in Table 1, which includes the number of customers and vehicles in each set. There are smaller sets ($pr01$, $pr02$, $pr11$, $pr12$, and $pr17$), medium sets ($pr03$, $pr05$, $pr15$, and $pr19$), and bigger sets ($pr16$). For the configuration of the weights in the objective function described in Section 3, the same values as [2] were used in this work, which are the following:

$$w_1 = 8, \quad w_2 = 3, \quad w_3 = 1, \quad w_4 = 1, \quad w_5 = n, \quad w_6 = n, \quad w_7 = n, \tag{16}$$

Table 1: Datasets used in this research, which are obtained from a previous work of Cordeau and Laporte [22]. Both the customer and vehicles count for each dataset are presented.

Dataset	Customers	Vehicles
pr01	24	3
pr02	48	5
pr03	72	7
pr05	120	11
pr11	24	3
pr12	48	5
pr15	120	11
pr16	144	13
pr17	36	4
pr19	108	8

Table 2: Comparison of different DARPTW approaches considered for evaluation in this work.

	Hyperheuristic (this work)	SA (Mauri and Lorena [6])	MOSA (Zidi et al. [7])	GA (Jorgensen et al. [2])	GA (Cubillos et al. [5])
Constraint managing	Soft	Soft	Hard	Soft	Hard
Depots	Single	Multiple	Multiple	Multiple	Single
Vehicles	Homogeneous	Heterogeneous	Homogeneous	Homogeneous	Homogeneous
Objective parameters	QoS oriented	QoS oriented	Balanced	QoS oriented	Balanced
CPU setup	*i5* 2.30 GHz	*Celeron* 2.0 GHz	*Core 2 Duo* 2.0 GHz	*Celeron* 2.0 GHz	*Pentium 4* 2.66 GHz

Table 3: Results for DARPTW obtained by using the hyperheuristic model with the *greedy heuristic selection* operator and the *random permutation descent* operator.

	HH with *greedy*					HH with *random permutation descent*				
	Route duration		Ride time		CPU time	Route duration		Ride time		CPU time
	Avg.	Best	Avg.	Best	min	Avg.	Best	Avg.	Best	min
pr01	1011.28	848.12	294.06	222.63	1.08	996.14	875.03	293.83	225.29	1.15
pr02	2061.82	2012.68	624.42	454.18	1.48	2044.30	1813.17	668.81	545.79	1.55
pr03	2779.83	2673.89	1112.08	967.50	1.93	2825.89	2669.76	1234.97	1044.24	2.00
pr05	4265.13	4007.28	1592.12	1396.41	3.00	4392.61	4212.04	2004.69	1566.09	3.09
pr11	917.10	794.14	221.50	174.97	1.07	921.48	787.68	226.73	151.52	1.14
pr12	1550.91	1475.81	368.66	294.49	1.49	1619.21	1475.81	399.42	268.65	1.56
pr15	4129.36	3937.46	1131.83	938.83	3.02	4338.53	3937.46	1219.13	1027.10	3.05
pr16	4979.85	4708.17	1618.35	1416.12	3.71	5108.65	4708.17	1892.12	1497.70	3.78
pr17	1237.40	1146.47	389.75	257.48	1.25	1263.79	1146.47	353.18	276.71	1.33
pr19	3626.02	3518.29	1804.54	1406.06	2.80	3677.65	3518.29	2266.98	1809.17	2.88

with n the number of transported customers for each dataset. This configuration prioritizes the quality of service, because it reflects common preferences of customers.

Regarding the move acceptance operators mentioned in Section 2, only *improve or equals* were used in this proposal, because the *all moves* alternative always showed poor convergence in preliminary tests, and the *only improving* did not provide relevant differences to the selected one. Regarding heuristic selection, the *greedy* and *random permutation descent* operators were used in this proposal, because they presented a more interesting behavior for evaluation rather than *simple random* versions. For all tests, about 200000 iterations were considered, without time limit, and 20 repetitions were performed on each test. The solver was implemented in Java, based on the current *hMod* framework [23], and it was executed in a *i5* 2.30 GHz processor.

Considering the numerous variants of DARPTW in the literature, it is important to remark several issues regarding how the problem is addressed by a heuristic method, before providing a comparison. Table 2 shows some of such issues for the methods to be compared with the hyperheuristic implementation. The feasibility issue is related to the manner how constraints are managed, particularly for time windows. In *Hard* cases, it is common to implement repair operators when infeasible solutions are obtained, and the *Soft* cases commonly use penalization in the objective function. The use of a single depot or multiple depots is not so relevant because the dataset used in all of these works only supports a single depot; therefore, the multiple depot heuristics only adapt to the case. The same occurs with the vehicle capacity issue, which is homogeneous by default in datasets. Finally, one may consider the objective function issue to explain possible tendencies of the results of each technique.

The results obtained by the hyperheuristic implemented in this work, considering both the *greedy* operator and the *random permutation descent* operator, are shown in Table 3. The results of the SA-related metaheuristics are presented in Table 4, and the results of the GA-related ones are presented in Table 5. The leftmost column shows the benchmark instances used for evaluation. As each compared heuristic method may use different criteria for the evaluation of the problem rather than provide the result of the objective function, two different elements related to a single solution were presented: the route duration, that is, the total time spent from that vehicle, starting its schedule to the time at which it finishes, and the ride time, that is, the total time spent by customers aboard the vehicles. The former element reflects the performance of solutions for the transport system perspective, while the latter reflects the quality of service perspective. In all tables, two different values for both elements are presented: the average value (avg. column) of repetitions for each instance, when available, and the best result (best column) obtained for such instance. Also, the best CPU time in all repetitions for each instance (in minutes) is provided for

Table 4: Results for DARPTW obtained by using the Simulated Annealing (SA) solver described in [6] and a multiagent system with a SA embedded (MOSA) described in [7].

	SA (Mauri and Lorena [6])					MOSA (Zidi et al. [7])				
	Route duration		Ride time		CPU time	Route duration		Ride time		CPU time
	Avg.	Best	Avg.	Best	min	Avg.	Best	Avg.	Best	min
pr01	—	831.30	—	241.93	1.00	—	932.73	—	516.19	1.12
pr02	—	1992.34	—	310.17	1.20	—	1825.39	—	731.20	3.29
pr03	—	2404.67	—	894.08	1.46	—	2806.70	—	3546.64	12.30
pr05	—	3920.25	—	899.35	1.79	—	3809.00	—	2785.00	10.56
pr11	—	738.42	—	206.66	0.92	—	1003.77	—	598.13	1.53
pr12	—	1428.44	—	311.95	1.30	—	1366.43	—	811.50	5.62
pr15	—	3654.02	—	855.16	1.95	—	4051.60	—	3001.20	16.20
pr16	—	4318.33	—	1245.66	1.94	—	4512.30	—	2247.00	15.21
pr17	—	1095.67	—	345.10	1.05	—	1397.50	—	933.58	3.60
pr19	—	3315.28	—	1085.18	2.26	—	3312.70	—	2894.00	14.00

Table 5: Results for DARPTW obtained by using the *cluster-first*, *route-second* Genetic Algorithm (GA) described in [2] and a GA with preprocessing techniques described in [5].

	GA (Jorgensen et al. [2])					GA (Cubillos et al. [5])				
	Route duration		Ride time		CPU time	Route duration		Ride time		CPU time
	Avg.	Best	Avg.	Best	min	Avg.	Best	Avg.	Best	min
pr01	1041.00	1039.00	477.00	310.00	5.57	1012.32	955.25	546.56	524.59	1.36
pr02	1969.00	1994.00	1367.00	1330.00	11.43	1836.05	1839.06	889.05	838.41	4.08
pr03	2779.00	2781.00	3081.00	2894.00	21.58	2810.59	2787.18	1634.10	1597.95	7.96
pr05	4250.00	4274.00	5099.00	4837.00	58.23	4118.03	4068.05	3007.97	2935.48	18.43
pr11	907.00	928.00	630.00	549.00	5.46	908.09	902.18	454.40	449.91	1.58
pr12	1719.00	1710.00	1214.00	1300.00	11.72	1503.71	1503.34	766.21	744.93	4.49
pr15	4296.00	4336.00	4615.00	4720.00	58.93	4118.34	4057.08	3160.79	3152.67	22.09
pr16	5309.00	5227.00	6134.00	6397.00	81.23	4655.77	4658.35	2377.45	2348.48	17.48
pr17	1299.00	1316.00	990.00	784.00	8.29	1227.46	1223.68	612.40	612.40	3.13
pr19	3679.00	3676.00	5362.00	5358.00	44.66	3441.71	3427.06	2597.10	2515.53	25.42

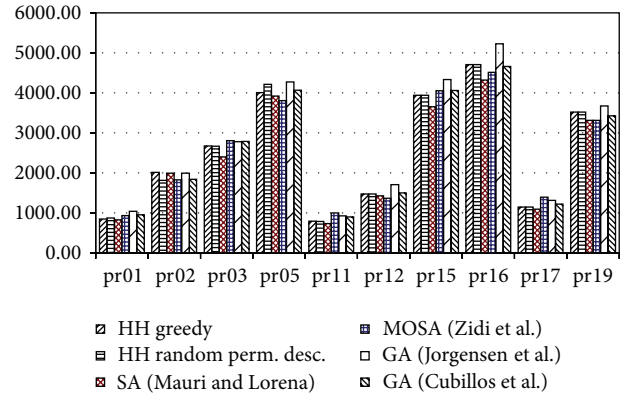
each heuristic method. For a better comparison of the values mentioned above, Figure 3 presents several charts in which the values in the tables are graphically contrasted.

Regarding the presented results, several observations can be made.

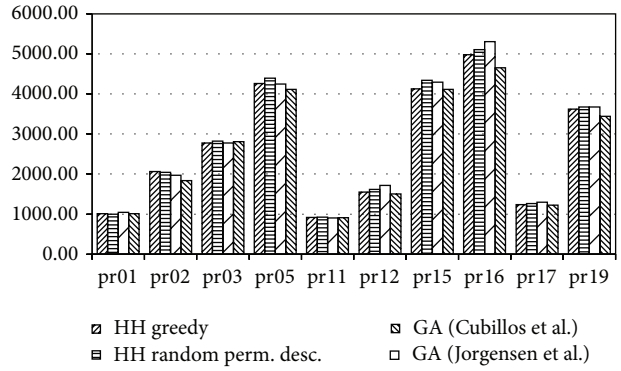
(i) In general terms, the usage of the *greedy heuristic selection* works slightly better than the *random permutation descent* version, not only for solution quality, but also for CPU time. Several exceptions were found for this pattern, for example, the *pr02* instance. The CPU time issue could be due to the generation of *random permutations* that may consume a few additional iterations than a direct iteration over the low-level heuristic set, which is the *greedy* operator behavior.

(ii) When comparing the best results of each heuristic method, the hyperheuristic implementation provides a moderate improving in the solution quality. For the route duration factor, the results are competitive with the GA implementations, but the SA ones perform better in most cases. For the ride time factor, the SA of Mauri and Lorena performs better in general terms; however, the hyperheuristic implementation clearly outperforms the other metaheuristics. This behavior could be explained by the configuration of weights in the objective function. Both the Mauri and Lorena SA and the hyperheuristic are more focused on the quality of service optimization; therefore, it is not surprising to obtain a better performance in this perspective. Although Jorgensen et al. GA is also focused on the quality of service, the *cluster-first, route-second* approach may impact the results. A more detailed comparison with the SA approaches would be done if average values for both the route duration and ride time were available.

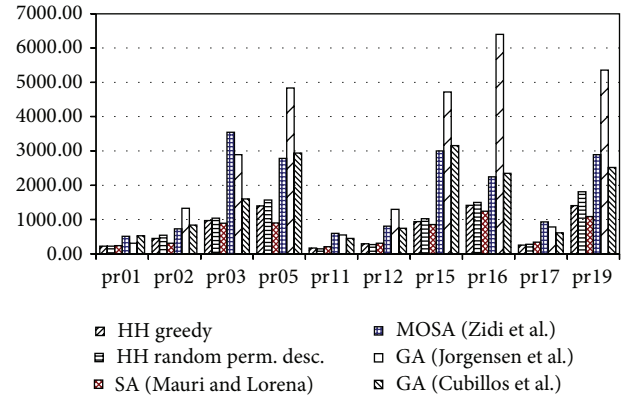
(iii) When comparing the average results with the hyperheuristic implementation and the GA metaheuristics, the behavior is similar to the best-values comparison. For the route duration factor, the results are similar between the hyperheuristic and the GAs, obtaining very moderate improvement in several cases for the former. However, in the ride time factor, the hyperheuristic clearly outperforms the GAs in all



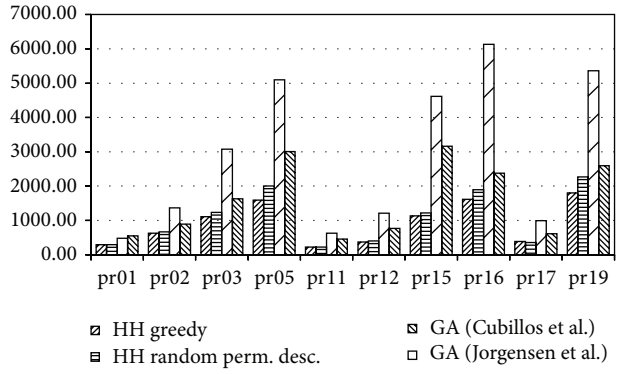
(a) Comparison of route duration (best result)



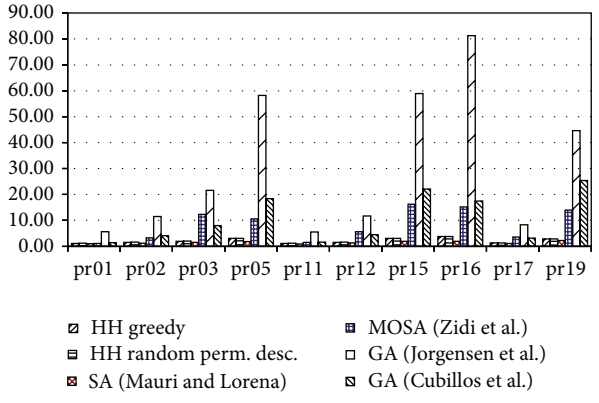
(b) Comparison of route duration (average result)



(c) Comparison of ride time (best result)



(d) Comparison of ride time (average result)



(e) Comparison of CPU time (best time)

Figure 3: Performance comparison of different heuristic methods for DARPTW, which includes the hyperheuristic implementation with *greedy* and *random permutation descent* heuristic selection, a Simulated Annealing (SA, Mauri and Lorena) in [6], a multiagent system with a SA embedded (MOSA, Zidi et al.) in [7], a Genetic Algorithm (GA, Jorgensen et al.) in [2], and another GA (Cubillos et al.) in [5]. The uppermost figures compare the *route duration* factor, which is focused on the transport system optimization. The middle figures compare the *ride time* factor, which is focused on the quality of service optimization. The leftmost figures compare the best value obtained, and the rightmost figures compare the average value obtained (not available in all cases). The bottom figure compares the CPU time.

instances. Again, this is mainly due to the weights configured for the objective function evaluation. The hyperheuristic appears to perform very consistently to this configuration.

(iv) Finally, for the CPU times, the hyperheuristic operates very efficiently regarding the other metaheuristics, and it is only outperformed by the SA of Mauri and Lorena by no more than a few minutes in

bigger instances. This is interesting considering that a hyperheuristic architecture is rather complex and the communication overhead between the hyperheuristic and the low-level could be high during the execution. However, it is important to consider also that the CPU setup in this work is better than others.

The results described above demonstrate that the hyperheuristic solver proposed in this work performs competitively regarding other metaheuristics in the literature, even with the usage of very simple operators at the high-level and simple but very specialized low-level heuristics at the DARPTW domain. It is important to remark that, because of the soft constraints used in the evaluation of the objective function, the results obtained by the hyperheuristic are not totally absent of infeasibility. However, the cases in which some constraint has been violated are limited, and for such cases, the violation magnitude is tolerable in practical settings (i.e., only a few time units of time windows violation).

6. Conclusions

In this work, a novel approach for solving the DARPTW problem was proposed: the usage of a generic-form hyperheuristic solver. Hyperheuristics are relatively new heuristic methods that were studied and developed during the last decade, and their conceptualization based in the reuse of solving intelligence has promoted evaluating them for a number of complex problems of the operations research field. To our knowledge, this work represents a first approach using hyperheuristics for solving the DARPTW problem, particularly with the complexity of including the time windows constraints. The experimental results demonstrated that this proposal can perform competitively regarding other metaheuristics for DARPTW, which include Genetic Algorithms and Simulated Annealing. Moreover, results showed that the hyperheuristic behaves consistently with the configured weights for the objective function calculation.

The hyperheuristic solver developed during this research is based on very basic and easy-to-implement operators, in both the high-level and the DARPTW domain. The competitive results obtained are regarded by the synergistic effect in the search process that is influenced by the usage of a simple high-level operator for finding efficient search strategies based on a set of simple but very specialized low-level heuristics of the problem domain. This is one of the most interesting aspects of the hyperheuristic concept, and this research makes a contribution in demonstrating such behavior in a complex problem such as DARPTW. Another clear advantage of the hyperheuristics usage for this domain is the possibility of easily reusing the implemented high-level solver in other different problems, maybe from the DARPTW family, such as TSP, VRP, and PDP. This can represent a contribution for extending the hyperheuristic applicability to more transportation-related or time windows-based problems.

Because of the simplicity of the proposed implementation, many improvements could be considered for further versions in both the high-level and the low-level. Regarding the hyperheuristic operators, more learning-oriented heuristic selection operators could be developed for solving DARPTW more efficiently, for example, the well-known *choice-function* proposed in [24]. Also, different metaheuristic-inspired approaches for hyperheuristics may be evaluated for DARPTW, such as tabu search [25] or Simulated Annealing [26]. At the problem domain, new low-level heuristics may be included in the current set, particularly several approaches that could selectively repair infeasible solutions for a better convergence and higher-quality solutions. Also, it can be interesting to reevaluate the hyperheuristic model of this work for new objective function weights, for example, a different configuration more oriented to improve the transport system factors. This could provide more evidence of the consistent behavior of the hyperheuristic implemented in this work.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem (DARP): variants, modeling issues and algorithms," *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, no. 2, pp. 89–101, 2003.

[2] R. M. Jorgensen, J. Larsen, and K. B. Bergvinsdottir, "Solving the dial-a-ride problem using genetic algorithms," *Journal of the Operational Research Society*, vol. 58, no. 10, pp. 1321–1331, 2007.

[3] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, "Static pickup and delivery problems: a classification scheme and survey," *TOP*, vol. 15, no. 1, pp. 1–31, 2007.

[4] C. Cubillos, F. Guidi-Polanco, and C. Demartini, *Applying Genetic Algorithms to the Dial-A-Ride Problem*, WSEAS, Turin, Italy, 2004, http://www.worldses.org/online/.

[5] C. Cubillos, E. Urra, and N. Rodríguez, "Application of genetic algorithms for the DARPTW problem," *International Journal of Computers, Communications and Control*, vol. 4, no. 2, pp. 127–136, 2009.

[6] G. R. Mauri and L. A. N. Lorena, "A multiobjective model and simulated annealing approach for a dial-a-ride problem," in *Workshopdos Cursos de Computação*, 2006.

[7] I. Zidi, K. Zidi, K. Mesghouni, and K. Ghedira, "A multi-agent system based on the multi-objective simulated annealing algorithm for the static dial a ride problem," in *Proceedings of the 18th World Congress of the International Federation of Automatic Control (IFAC '11)*, Milan, Italy, 2011.

[8] K. Sörensen and F. W. Glover, "Metaheuristics," in *Encyclopedia of Operations Research and Management Science*, S. I. Gass and M. C. Fu, Eds., pp. 960–970, Springer, Boston, Mass, USA, 2013.

[9] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. John Woodward, "A classiffication of hyper-heuristics approaches," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., vol. 57 of *International Series in Operations Research & Management Science*, pp. 449–468, Springer, New York, NY, USA, 2nd edition.

[10] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments," in *Adaptive and Multilevel Metaheuristics*, C.

Cotta, M. Sevaux, and K. Sorensen, Eds., vol. 136 of *Studies in Computational Intelligence*, pp. 3–29, Springer, Berlin, Germany, 2008.

[11] P. Cowling, G. Kendall, and E. Soubeiga, "Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation," in *Applications of Evolutionary Computing*, S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, Eds., vol. 2279 of *Lecture Notes in Computer Science*, pp. 1–10, Springer, Berlin, Germany, 2002.

[12] E. K. Burke, M. Gendreau, M. Hyde et al., "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[13] R. R. S. van Lon, T. Holvoet, G. V. Berghe, T. Wenseleers, and J. Branke, "Evolutionary synthesis of multi-agent systems for dynamic Dial-A-Ride problems," in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '12)*, F. Stonedahl and R. Riolo, Eds., pp. 331–336, ACM, Philadelphia, Pa, USA, 2012.

[14] P. Garrido and M. C. Riff, "DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic," *Journal of Heuristics*, vol. 16, no. 6, pp. 795–834, 2010.

[15] S. Luke, *Essentials of Metaheuristics*, Lulu Press, 2009.

[16] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: over view and conceptual comparison," *ACM Computing Surveys*, vol. 35, no. 3, pp. 268–308, 2003.

[17] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: an emerging direction in modern search technology," in *Handbook of Metaheuristics*, vol. 57 of *International Series in Operations Research & Management Science*, chapter 16, pp. 457–474, Springer, 2003.

[18] J. Swan, J. Woodward, E. Özcan, G. Kendall, and E. Burke, "Searching the hyper-heuristic design space," *Cognitive Computation*, vol. 6, no. 1, pp. 66–73, 2014.

[19] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis*, vol. 12, no. 1, pp. 3–23, 2008.

[20] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *Practice and Theory of Automated Timetabling III*, vol. 2079 of *Lecture Notes in Computer Science*, pp. 176–190, Springer, Berlin, Germany, 2001.

[21] M. Ayob and G. Kendall, "A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine," in *Proceedings of the International Conference on Intelligent Technologies (INTECH '03)*, pp. 132–141, Chiang Mai, Thailand, 2003.

[22] J.-F. Cordeau and G. Laporte, "A tabu search heuristic for the static multi-vehicle dial-a-ride problem," *Transportation Research Part B: Methodological*, vol. 37, no. 6, pp. 579–594, 2003.

[23] E. Urra, D. Cabrera-Paniagua, and C. Cubillos, "Towards an object-oriented pattern proposal for heuristic structures of diverse abstraction levels," in *Proceedings of the Workshop on Agents and Collaborative Systems (WACS '11)*, School of Computer Engineering, Catholic University of Temuco, Temuco, Chile, 2013.

[24] G. Kendall, E. Soubeiga, and P. Cowling, Choice Function and Random Hyperheuristics, 2002, http://citeseer.ist.psu.edu/547597.html, http://www.cs.nott.ac.uk/~gxk/gxk1/../papers/seal01.pdf.

[25] E. Burke and E. Soubeiga, "Scheduling Nurses Using A Tabu - Search Hyperheuristic," 2003, http://citeseer.ist.psu.edu/713555.html, http://www.mistaconference.org/2003/papers/page197.pdf.

[26] B. Bai, J. Blazewicz, E. Burke, G. Kendall, and B. McCollum, "A simulated annealing hyper-heuristic methodology for flexible decision support," *4OR: A Quarterly Journal of Operations Research*, vol. 10, no. 1, pp. 43–66, 2012.