Transportation Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Methodology Based on Evolutionary Algorithms to Solve a Dynamic Pickup and Delivery Problem Under a Hybrid Predictive Control Approach

Diego Muñoz-Carpintero, Doris Sáez, Cristián E. Cortés, Alfredo Núñez

To cite this article:
Diego Muñoz-Carpintero, Doris Sáez, Cristián E. Cortés, Alfredo Núñez (2015) A Methodology Based on Evolutionary Algorithms to Solve a Dynamic Pickup and Delivery Problem Under a Hybrid Predictive Control Approach. Transportation Science 49(2):239-253. http://dx.doi.org/10.1287/trsc.2014.0569

TRANSPORTATION SCIENCE
Vol. 49, No. 2, May 2015, pp. 239–253
ISSN 0041-1655 (print) | ISSN 1526-5447 (online)

http://dx.doi.org/10.1287/trsc.2014.0569

A Methodology Based on Evolutionary Algorithms to Solve a Dynamic Pickup and Delivery Problem Under a Hybrid Predictive Control Approach

Diego Muñoz-Carpintero, Doris Sáez
Electrical Engineering Department, Universidad de Chile, 8370451 Santiago, Chile {dimunoz@ing.uchile.cl, dsaez@ing.uchile.cl}

Cristián E. Cortés
Civil Engineering Department, Universidad de Chile, 8370449 Santiago, Chile, ccortes@ing.uchile.cl

Alfredo Núñez
Section of Road and Railway Engineering, Delft University of Technology, 2628 CN Delft, The Netherlands, a.a.nunezvicencio@tudelft.nl

This paper presents a methodology based on generic evolutionary algorithms to solve a dynamic pickup and delivery problem formulated under a hybrid predictive control approach. The solution scheme is designed to support the dispatcher of a dial-a-ride service, where quick and efficient real-time solutions are needed. The scheme considers different configurations of particle swarm optimization and genetic algorithms within a proposed ad-hoc methodology to solve in real time the nonlinear mixed-integer optimization problem related with the hybrid predictive control approach. These consist of different techniques to handle the operational constraints (penalization, Baldwinian, and Lamarckian repair) and encodings (continuous and integer). For parameter tuning, a new approach based on multiobjective optimization is proposed and used to select and study some of the evolutionary algorithms. The multiobjective feature arises when deciding the parameters with the best trade-off between performance and computational effort. Simulation results are presented to compare the different schemes proposed and to advise conditions for the application of the method in real instances.

Keywords: predictive control; dynamic pickup and delivery problem; evolutionary algorithms
History: Received: May 2010; revision received: October 2013; accepted: June 2014. Published online in *Articles in Advance* March 10, 2015.

1. Introduction and Background

The dynamic pickup and delivery problem (DPDP) can be formulated as a set of transportation requests (identified by pickup and delivery locations) that are served by a fleet of vehicles initially located at several depots. The dynamic dimension appears when a subset of the requests is not known in advance; therefore, such requests have to be scheduled for service in real time, at the instant they call. The DPDP is now of great interest for practitioners and researchers because of the development and implementation of efficient online optimization tools; this fact is crucial for the emerging improvement in the quality of the formulation solutions. The DPDP has been intensely studied over the last 20 years (Psaraftis 1980, 1988; Bertsimas and Van Ryzin 1991; Bertsimas and Van Ryzin 1993a, b; Kleywegt and Papastavrou 1998; Gendreau et al. 1999; Swihart and Papastavrou 1999; Larsen 2000; Thomas and White 2004). The output of such a problem should be a set of routes for all of the vehicles, which change dynamically over time. The solution of a DPDP can be linked with the control of a typical dial-a-ride system (DRS), where the dispatcher has to make routing decisions for a fixed fleet of vehicles with limited capacity, operating in a real-time service where the demand (represented by passengers) is unknown in advance. Eksioglu, Volkan, and Reisman (2009) and Berbeglia, Cordeau, and Laporte (2010) presented comprehensive reviews of DPDP, dial-a-ride applications, and solution methods.

Xiang, Chu, and Chen (2008) studied a DRS by considering a complex set of constraints on a time-dependent network. With regard to real applications, Madsen, Raven, and Rygaard (1995) adapted the insertion heuristics by Jaw et al. (1986) to solve a real-life problem for moving elderly and handicapped people in Copenhagen, and Dial (1995) proposed an approach to the many-to-few dial-a-ride transit operation ADART (Autonomous Dial-a-Ride Transit), which is currently implemented in Corpus Christi, Texas. Gendreau et al. (1999) modified the tabu-search heuristics to solve the dynamic vehicle routing problem (DVRP) with soft time windows in an effort to find a solution method that will handle different DVRPs. More sophisticated tabu-search methods were recently developed, such as granular tabu search (Toth and

Vigo 2003) and adaptive memory based on tabu search (Tarantilis 2005). Xiang, Chu, and Chen (2008) developed a heuristic local search strategy that uses a secondary objective function to drive the search out of local optima. In the context of DPDP, Mitrovic-Minic, Krishnamurti, and Laporte (2004) introduced the concept of double-horizon based heuristics for solving the DPDP with time windows, showing that the method can yield gains in route costs when compared with classical (single) rolling horizon methods, but the improvement tends to decrease as instances become larger. Mitrovic-Minic and Laporte (2004) presented four waiting strategies for vehicles (drive-first, wait-first, dynamic waiting, and the advanced dynamic waiting). They concluded that in terms of total route length, the proposed strategies outperform the commonly used drive-first waiting strategy, making the advanced dynamic waiting strategy the most efficient.

The dial-a-ride system can be modeled for designing a hybrid predictive control (HPC) scheme, considering that potential rerouting of vehicles could affect the current decisions, by analyzing the extra cost of inserting real-time service requests into predefined vehicle routes while the vehicles are in service. In previous works of our group, a formulation of the DPDP in an HPC by specifying the state space variables and models was presented (Sáez, Cortés, and Núñez 2008; Cortés, Sáez, and Núñez 2008; Cortés et al. 2009). In those works, two solution algorithms using genetic algorithms (GA) and particle swarm optimization (PSO) were developed to solve real-time instances. To the best of our knowledge, no other hybrid predictive control approaches for solving DPDP have been proposed in the literature using PSO and GA that can perform real-time control on a dial-a-ride type of system.

In the literature, most of the applications using such methods (namely, PSO and GA) solve static cases, or vehicle routing problems (VRP) that neither include explicitly the dynamic behavior of the system, nor a reasonable set of future realizations of the stochastic demand. Specifically, GAs have been applied for various VRPs, considering different chromosome representations and genetic operators depending on the particular problem: Skrlec, Filipec, and Krajcar (1997), for the single vehicle capacitated VRP; Haghani and Jung (2005), for the multivehicle DVRP with time-dependent travel time and soft time windows. Zhu et al. (2006) proposed an adapted PSO algorithm to solve a static VRP with time windows. Jih and Hsu (1999) and Osman, Abo-Sinna, and Mousa (2005) presented a successful comparison of the GA against dynamic programming in terms of computation time; the former solved the DVRP with time windows and capacity constraints and the latter solved a multiobjective VRP. Ant colony methods, a metaheuristic inspired by the behavior of real ant colonies, have also been applied to solve DVRP (Montemanni et al. 2005; Dréo et al. 2006).

In the current work, the main objective is to present an efficient and systematic ad-hoc methodology to solve the HPC formulation of the DPDP based on generic evolutionary algorithms. The solution scheme will help dispatchers of the dial-a-ride service, where efficient real-time solutions are needed quickly to make the system work. Specific implementations of GA and PSO will result in several variants of the generic evolutionary algorithms. In addition, a multiobjective approach for tuning the parameters of the proposed evolutionary algorithms is presented. The multiobjective feature arises for deciding the parameters with the best trade-off between performance and computation time required for real applications. A detailed analysis of the accuracy and computational time is conducted, which can be extended to other complex nonlinear engineering problems containing mixed integers and continuous variables.

In §2, a summary of the problem statement proposed by Cortés et al. (2009) is presented, which is the starting point for the methodology based on generic evolutionary algorithms proposed in this work. In §3 we present the ad-hoc methodology, the PSO and GA configurations used within, and the parameter tuning based on multiobjective optimization. Next, in §4 a detailed computational analysis and comparison of the configurations of PSO and GA are carried out based on simulations. Finally, conclusions, remarks, and further research are presented in §5.

2. Problem Statement

2.1. General Description

In the context of control theory, the notion of hybrid systems arises when the problem conditions are characterized by both continuous and discrete/integer variables. In the last two decades, hybrid systems have been studied more intensely by researchers from several study areas, such as computer science and automatic control (see for example Bemporad and Morari 1999; Hegyi, De Schutter, and Hellendoorn 2005; Karer et al. 2007a, b; Núñez et al. 2009). Specifically, hybrid systems can be expressed as a nonlinear state space model given by

$$\begin{aligned} x(k+1) &= f(x(k), u(k)), \\ y(k) &= g(x(k), u(k)), \end{aligned} \tag{1}$$

where $x(k)$ are the continuous and/or discrete (integer) state space variables, $u(k)$ are the continuous and/or discrete input or manipulated variables, $y(k)$ define the continuous and/or discrete system outputs and $f(\cdot,\cdot)$, $g(\cdot,\cdot)$ are piecewise nonlinear functions. In

general, a hybrid predictive control controller minimizes the following generic objective function:

$$\min_{u(k),\ldots,u(k+N_u-1)} J(u(k),\ldots,u(k+N_u-1),\hat{x}(k+1),\ldots,\hat{x}(k+N),\hat{y}(k+1),\ldots,\hat{y}(k+N)), \quad (2)$$

where J is an objective function, k is the current time, N is the prediction horizon, N_u is the control horizon, $\hat{x}(k+t)$, $\hat{y}(k+t)$ are the expected state space vector and the expected system output at instant $k+t$, respectively, and $[u(k)^T,\ldots,u(k+N_u-1)^T]^T$ represents the control sequence, which corresponds to the set of optimization variables. Once expression (2) is optimized, only the first element of the control vector $u(k)$ is used to update the system conditions, based on the receding horizon methodology.

Conceptually, the HPC framework to model the DPDP incorporates stochasticity into the routing dispatch rules by considering the impact of future reassignments on the performance of already-scheduled customers (Cortés et al. 2009). The stochastic prediction allows the dispatcher to incorporate a more realistic and robust measure of effective travel (waiting) time experienced by the users into the decision objective function.

We consider a fleet of F vehicles, which are dynamically routed over an influence area A. The demand for service is unknown and is revealed in real time. Quick routing and scheduling decisions are required to handle the demand with the available vehicles. At any time k, each vehicle j is assigned to follow a sequence of pickups and deliveries (control action), and can be represented by the function $S_j(k)$, where the ith element of the sequence represents the ith stop of vehicle j along its route, and $w_j(k)$ is the total number of stops. A stop is defined by a user who requires the service (it could be its pickup or delivery). The initial condition $s_j^0(k)$ corresponds to the position of vehicle j at instant time k. The set of sequences $S(k)=[S_1(k)^T,\ldots,S_j(k)^T,\ldots,S_F(k)^T]^T$ associated with the fleet of vehicles correspond to the control (manipulated) variable $u(k)$. The sequence of stops assigned to vehicle j at instant k, $S_j(k)$, is given by

$$S_j(k)=\begin{bmatrix} s_j^0(k) \\ s_j^1(k) \\ \vdots \\ s_j^{w_j(k)}(k) \end{bmatrix} = \begin{bmatrix} r_j^1(k) & 1-r_j^1(k) & \Gamma_j^1(k) & label_j^1(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^i(k) & 1-r_j^i(k) & \Gamma_j^i(k) & label_j^i(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{w_j(k)}(k) & 1-r_j^{w_j(k)}(k) & \Gamma_j^{w_j(k)}(k) & label_j^{w_j(k)}(k) \end{bmatrix}, \quad (3)$$

where $r_j^i(k)$ is a binary variable defined as follows:

$$r_j^i(k)=\begin{cases} 1 & \text{if stop } i \text{ belonging to } S_j(k) \text{ is a pick-up} \\ 0 & \text{if stop } i \text{ belonging to } S_j(k) \text{ is a delivery.} \end{cases}$$

The first and second columns represent a pair identifying if stop i is either a pickup [1 0] or a delivery [0 1], respectively. The third column of the $S_j(k)$ matrix represents the external travel time function, where $\Gamma_j^i(k)$ is the expected total travel time between points $i-1$ and i plus the transfer operation delay at node i. For simulation purposes, we assume that the position of the vehicles can be measured or estimated at any moment. The last column $label_j^i$ keeps the passenger identifier, which is needed to check the feasibility of the sequence in terms of precedence (the pickup must occur before the delivery of the same client).

When the dispatcher makes a decision, first the passengers are assigned to a certain vehicle, and then they are inserted within the sequence of task that the vehicle follows. Figure 1 shows an example of a sequence for a vehicle j. Users labeled as "1," "2," and "3" are assigned to vehicle j. The sequence assigned considers to pickup user "$\text{label}_j^1(k)=1$" (coordinate 1^+), then to pickup user "$\text{label}_j^2(k)=3$" (coordinate 3^+), then to delivery user "$\text{label}_j^3(k)=1$" (coordinate 1^-) and so on.

Vehicles will travel according to the predefined sequence vector $S(k-1)$ while no new calls are received. When a new service request is received, the dispatcher calculates the control sequence in the next step $S(k)$ for the fleet of vehicles, adding the stops indicated by the new customer. Then, each sequence $S_j(k)$ remains fixed during the whole time interval $(k,k+1)$, unless a vehicle reaches a predefined pickup or delivery stop during such an interval, in which case its sequence will decrease in size showing that the scheduled task has been accomplished. Thus, in this scheme the problem is formulated in terms of a variable time step (triggered by events), which represents the time interval between two consecutive requests,

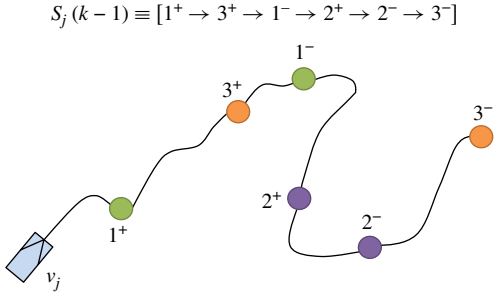


Figure 1 (Color online) Representation of a Sequence of Vehicle j and Its Stops

that is to say, the predictive controller takes a routing decision when a new call enters the system.

2.2. Predictive Dynamic Model and Objective Function

In the DPDP the state space variables include the clock time of departure $T_j^i(k)$ and the vehicle load $L_j^i(k)$, after vehicle j leaves stop i, both computed at instant k. At this point, let us define, for each vehicle $j \in V$, the load and departure time vectors as follows:

$$L_j(k) = \left[L_j^0(k) \;\; L_j^1(k) \;\; \ldots \;\; L_j^{w_j(k-1)}(k) \right]^T_{(w_j(k-1)+1)\times 1}, \quad (4)$$

$$T_j(k) = \left[T_j^0(k) \;\; T_j^1(k) \;\; \ldots \;\; T_j^{w_j(k-1)}(k) \right]^T_{(w_j(k-1)+1)\times 1}. \quad (5)$$

Thus, the set of state space variables for the entire system at instant k can be written as $x(k) = [L(k)^T, T(k)^T]^T$, where $L(k)$ and $T(k)$ represent the set of load and departure time vectors, respectively, $L(k) = [L_1(k)^T, \ldots, L_j(k)^T, \ldots, L_F(k)^T]^T$ and $T(k) = [T_1(k)^T, \ldots, T_j(k)^T, \ldots, T_F(k)^T]^T$. The output set $y(k)$ is represented by the vector of observed departure times of vehicles at stops, $T(k)$.

In the HPC approach, a dynamic model based on state space representation for both the vehicle load and the departure time at stops (as a function of segment travel times) is considered (Cortés et al. 2009). Both the clock time of departure $T_j^i(k)$ and the vehicle load $L_j^i(k)$ are stochastic variables, because they depend on the evolution of the system affected by uncertain demand. Therefore, and in order to work with deterministic values, reasonable estimations of the load and departure time vectors have to be obtained. The prediction when a new request occurred is given by the expected value of the state space vector for vehicle j, $\hat{x}_j(k+1)$. Analytically,

$$\hat{x}_j(k+1) = \begin{bmatrix} E\{L_j(k+1)/k\} \\ E\{T_j(k+1)/k\} \end{bmatrix} = \begin{bmatrix} \hat{L}_j(k+1) \\ \hat{T}_j(k+1) \end{bmatrix} = \begin{bmatrix} f_L(L_j(k), S_j(k)) \\ f_T(T_j(k), S_j(k)) \end{bmatrix} \quad \forall j = 1, \ldots, F, \quad (6)$$

where the functions f_L and f_T are the state space model defined in Appendix A.

Once the optimization is conducted, the proposed vehicle sequences and state space variables must satisfy a set of constraints given by the real conditions of the dial-a-ride system. Specifically, we must consider precedence and consistency constraints (pickup location goes before the delivery for the same client) in the solution of the HPC problem to generate only feasible sequences.

The central dispatcher (controller) computes the control decisions, by means of the minimization of an objective function and predictions, for the entire control horizon $N_u = N$ (N is the prediction horizon), i.e., $S_k^{k+N} = [S(k)^T, \ldots, S(k+N-1)^T]^T$, and applies the updated sequence set $S(k)$ based on a receding horizon strategy. The optimization variables are the current sequence that incorporate the new request, and the future sequences that incorporate the prediction of future requests. Thus, the objective function comprises all of the scenarios h that consists of the sequential occurrence of $N-1$ estimated future request, with a probability p_h. The scenarios are obtained from historical data. Therefore, a reasonable prediction horizon N can be defined depending on the intensity of unknown events that enter the system in real time and on how good the prediction model is. If the prediction horizon is longer than one, the controller will add the future behavior of the system into the current decision. The performance of the vehicle routing scheme will depend on how well the objective function can predict the impact of possible rerouting due to insertions caused by unknown service requests. Analytically, a mono-objective function for a prediction horizon N, can be written as follows:

$$\min_{S_k^{k+N}} \sum_{t=1}^{N} \sum_{j=1}^{F} \sum_{h=1}^{H(k+t)} p_h(k+t)(C_j(k+t) - C_j(k+t-1)) \quad (7)$$

where

$$C_j(k+t) = \sum_{i=1}^{w_j(k+t)} \Big[\hat{L}_j^{i-1}(k+t)(\hat{T}_j^i(k+t) - \hat{T}_j^{i-1}(k+t)) + z_j^i(k+t)(\hat{T}_j^i(k+t) - T_j^0(k+t)) \Big]\Big|_h, \quad (8)$$

where $k+t$ is the instant at which the tth request enters the system, measured from instant k, $H(k+t)$ is the number of requests' patterns at instant $k+t$, $p_h(k+t)$ is the probability of occurrence of the hth request pattern, associated with a trip pattern related to a specific pair of zones. The patterns and $p_h(k+t)$ are calculated based on real-time or historical data, or a combination of both. This formulation is robust in the sense that different realizations of the stochastic demands are considered in the objective function. In Sáez, Cortés, and Núñez (2008) a zoning based on fuzzy clustering is designed, in which the estimation of trip patterns is systematized. The first term of (8) is related with the travel time of users, and the second term with the waiting time.

In Figure 2, the scheme for the HPC of the dial-a-ride system is presented. Note that the optimization problem associated with the HPC strategy will be solved using evolutionary algorithms as we explain in §3.

3. Solution Algorithms

For solving the optimization problem given by the objective function defined in (7) within the proposed HPC strategy applied to the dial-a-ride system

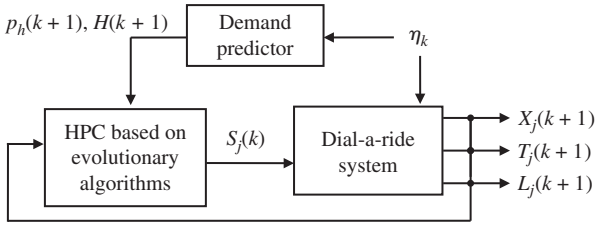


Figure 2 HPC Scheme for Controlling the Dial-a-Ride System

described in §2, we propose a new ad-hoc methodology based on generic evolutionary algorithms. The generic evolutionary algorithms to be used are described in Figure 3.

Evolutionary algorithms consist of a population where each member represents or encodes a solution of a given problem, and the quality of a solution (and indeed the member) is evaluated with a fitness function, which is directly related to the objective function. The members of the population evolve over the different iterations according to the operators of the algorithm (evolutionary stage). At the end of the algorithm, which, for instance, can be decided by reaching a maximum number of iterations, the solution to the problem is the one stored as the best member of the population (which may be the optimal or a suboptimal solution). In this work, PSO and GA evolutionary optimization algorithms are considered.

These evolutionary algorithms are used inside the ad-hoc methodology to solve the HPC of the dial-a-ride system (HPC-EA-DRS), and as such, they are used to find good insertion positions of the requests.

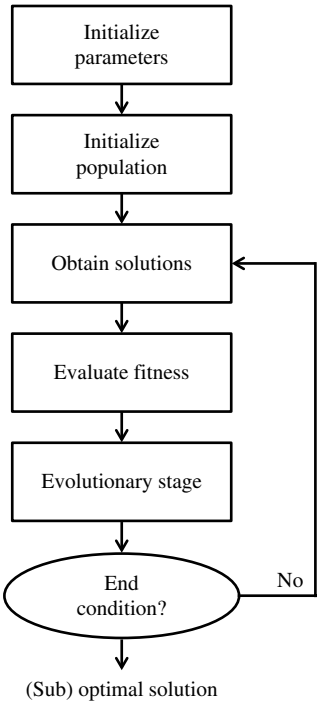


Figure 3 Generic Evolutionary Algorithm

The ad-hoc methodology for the HPC of the dial-a-ride system has been designed based on the particular structure of the problem, and is presented next.

3.1. Methodology Based on Evolutionary Algorithms (EA)

In this section, we present the developed methodology based on evolutionary algorithms to solve the hybrid predictive control for the dial-a-ride system for the two-steps ahead prediction problem within the HPC formulation described in §2.

As shown in expression (7), the objective function of the optimization problem depends not only on the current assignment for pickup and delivery but also on the expected future requests. Then, in order to find a good assignment for a given incoming request, it is necessary to compute optimal or near-optimal assignments for expected future requests in the next prediction step, assuming the sequence in the previous step is known. Thus, a two-level algorithm is considered; at each level, an evolutionary algorithm as that schematically presented in Figure 3 is executed, where the solutions represent the insertion positions (or the sequence) for the incoming request.

The first level solves the optimization for the insertion positions of the incoming call based on the proposed objective function shown in (7). The second level determines the positions associated with the expected future requests, using the sequences generated in the first level. The two-level procedure is conducted for every vehicle to determine the vehicle experiencing the lowest insertion cost (based on the objective function). The main reason for running the procedure separately for each vehicle is because different vehicles have different sequence lengths, which makes a consistent design of the solutions' encoding difficult when applied to different vehicles.

Each member of the population of the chosen evolutionary algorithm represents a candidate solution for the insertion positions of a request for a given vehicle j. The insertion positions are defined as $\vartheta = (pu, de)$, where pu represents the position of the pickup, and de is the position of the delivery. The decoded sequence based on (3) is

$$s_j^v(k) = \begin{bmatrix} s_j^0(k) \\ s_j^1(k) \\ \vdots \\ s_j^{pu}(k) \\ \vdots \\ s_j^{de}(k) \\ \vdots \\ s_j^{w_j(k)}(k) \end{bmatrix}$$

$$= \begin{bmatrix} r_j^0(k) & 1-r_j^0(k) & \Gamma_j^0(k) & label_j^0(k) \\ r_j^1(k) & 1-r_j^1(k) & \Gamma_j^1(k) & label_j^1(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{pu}(k) & 1-r_j^{pu}(k) & \Gamma_j^{pu}(k) & label_j^{pu}(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{de}(k) & 1-r_j^{de}(k) & \Gamma_j^{de}(k) & label_j^{de}(k) \\ \vdots & \vdots & \vdots & \vdots \\ r_j^{w_j(k)}(k) & 1-r_j^{w_j(k)}(k) & \Gamma_j^{w_j(k)}(k) & label_j^{w_j(k)}(k) \end{bmatrix} \quad (9)$$

The particular encodings of the insertion positions associated with each method, as well as the repair strategies designed to handle unfeasible solutions, are defined in §3.2.

For a specific vehicle, the first level considers the insertion positions of the incoming call; the fitness function for the first level $f_{1\text{st level}}(k)$ is given by the objective function associated with the whole optimization problem, as defined in (7). As this objective function considers the insertion costs of the expected future requests, the second level of the algorithm must be run inside the objective function computation step of the first level in order to find near-optimal insertion positions for every expected future request, assuming that the candidate solution for the incoming request being evaluated at the first level is assigned to such a vehicle. In addition, as the second level considers only the insertion positions of one of the expected requests (h) provided a candidate solution for the incoming request $S^{\vartheta_l}(k)$, the second level fitness, namely, $f_{2\text{nd level}}(k)$, considers the following objective function for every candidate solution:

$$f_{2\text{nd level}}(k) = \sum_{j=1}^{F} (C_j(k+2) - C_j(k+1))|_{S^{\vartheta_l}(k),\,h}. \quad (10)$$

Next, we describe the two-level HPC-EA-DRS algorithm that uses modules (see Figure 3) that in general will be different depending on the chosen evolutionary algorithm (these modules are described in Online Appendix B (available as supplemental material at http://dx.doi.org/10.1287/trsc.2014.0569).

The HPC-EA-DRS is as follows:

Step (0). Call function *InitializeParameters* of the selected evolutionary algorithm.

Step (1). Suppose that the predefined sequence set $S(k-1)$ is known. A new service request (call) enters the system. The first level of the algorithm then starts and the first level counter is set to $g_1 = 0$. The modules *InitPopulation* and *ObtainSequence* are used to generate a set of n potential sequences $S^{\vartheta_l}(k)$, with l: $1, 2, \ldots, n$. Note that $[n/F]$ candidate solutions are associated with each vehicle, which means that the insertion of the new call falls in the specific vehicle sequence (F is the fleet size).

Step (2). For each candidate sequence $S^{\vartheta_l}(k)$, $H(k+1)$ probable requests are considered. Then, the second level of the algorithm starts and the second level counter is set to $g_2 = 0$. *InitPopulation* and then *ObtainSequence* are applied to generate n potential sequences $S^{\vartheta_m}(k+1)|_h$, m: $1, 2, \ldots, n$, for each probable request pattern h: $1, 2, \ldots, H(k+1)$.

Step (3). Provided that $S^{\vartheta_l}(k)$ is known, evaluate the fitness function $f_{2\text{nd level}}(k)$, defined in (10), for all potential feasible sequences $S^{\vartheta_m}(k+1)|_h$. If $S^{\vartheta_m}(k+1)|_h$ or $S^{\vartheta_l}(k)$ are unfeasible in terms of the capacity of the vehicle or precedence, penalize its fitness (if a repair method is considered, the unfeasible solutions in terms of precedence constraints are repaired using *ObtainSequence*; thus, at this point the sequence is feasible, at least in precedence).

Step (4). If an ending criterion (maximum number of iterations) is satisfied, then proceed to Step 5, and return best solutions for every probable request. Otherwise, use the function *EvolutionaryStage* to update the particles or individuals. By using the function *ObtainSequence*, the sequences $S^{\vartheta_m}(k+1)|_h$, m: $1, 2, \ldots, n$ for h: $1, 2, \ldots, H(k+1)$ are generated $g_2 = g_2 + 1$. Go back to Step 3.

Step (5). At this step, the second level ends and we go back to the first level. Given that $S^{\vartheta_l}(k)$ is known and the solutions for $S^{\vartheta_m}(k+1)|_h$, h: $1, \ldots, H(k+1)$ are obtained in Step 3, the objective function $f_{1\text{st level}}(k)$, in this case considering two steps ahead, is evaluated and used as the fitness function. If $S^{\vartheta_l}(k)$ is unfeasible, penalize it.

Step (6). If a termination criterion (maximum number of iterations of Steps 2–6) is satisfied, then STOP, and return the best solution found. Otherwise, by using the function *EvolutionaryStage*, the positions and the velocities of all particles for l: $1, 2, \ldots, n$ are updated. Using *ObtainSequence*, $S^{\vartheta_l}(k)$, l: $1, 2, \ldots, n$, are generated $g_1 = g_1 + 1$. Go back to Step 2.

Next, we present the different evolutionary algorithm strategies, which are used as a part of this ad-hoc algorithm.

3.2. Evolutionary Algorithms for the HPC-EA-DRS

The evolutionary algorithms used in the HPC-EA-DRS are described in this section, including its basics and the actual operators according to the HPC formulation for the DPDP.

First, we highlight the PSO algorithm, which is based on a particle swarm that represents a population of candidate solutions (Kennedy and Eberhart 2001). The particles are initialized randomly, and then they move iteratively within the search space in order to find new solutions. The particles have a *fitness* associated with the solution quality, usually given by the objective function to be optimized. Each particle i is

characterized by a position, a velocity, its best previous position, and the best position among all of the particles belonging to the swarm. The particles are updated (they move) according to their cognitive and social behavior. The above description of PSO was originally conceived to solve continuous problems. In this work, we adapted the standard PSO configuration in order to add integer variables in the solution.

The second solver we explore is GA (Man, Tang, and Kwong 1998). It is based on biological evolution, and uses inherited operators such as mutation, selection, and crossover. In particular, the optimization variables in the HPC formulation of DPDP are discrete, and therefore the binary encoding is not necessary. In other words, the genes of the individuals (feasible solutions) are given directly by the integer optimization variables. In addition, gradient computations are not necessary as in conventional nonlinear optimization solvers, which saves significant computation time. Pseudocodes of both GA and PSO algorithms are presented in Online Appendix C.

Based on the standard PSO and GA methods, nine different cases are considered in this work, depending on the encoding and constraint handling methods. Two of these cases were previously proposed in Sáez, Cortés, and Núñez (2008) and Cortés et al. (2009), and the other seven are new ad-hoc implementations.

In these algorithms, each member of the population represents a candidate solution for the insertion positions of a request for a given vehicle. Each particle or individual (particle and individual are the typical nomenclature for candidate solutions in the PSO and GA literature, respectively) has two coordinates; the first represents the pickup position and the second represents the delivery position of insertion in the sequence of a single vehicle (no-swapping of stops is assumed as in Sáez, Cortés, and Núñez 2008; Cortés, Sáez, and Núñez 2008; Cortés et al. 2009). Particles or individuals are encoded by $x = (x_1, x_2)$, which represent a candidate solution. PSO algorithms can be used with continuous or discrete particles whose coordinates are of the form $x = (x_1, x_2) \in R$ or $x = (x_1, x_2) \in N$, respectively. In the case of GA, the coordinates of each individual are directly discrete, and therefore in such a case we considered $x = (x_1, x_2) \in N$. Hereafter in the paper, R and N represent the sets of real and integer numbers, respectively.

Recall that the insertion positions are defined as $\vartheta = (pu, de)$, where pu represents the position of the pickup, and de is the position of the delivery. A feasible insertion requires three conditions. First, the pickup and delivery positions have to be consistent with the length of the sequence; second, the pickup position in the sequence will always precede the delivery position; and third, the capacity constraint must be satisfied.

For instance, if we use a real-valued particle representation for vehicle j, a particle is encoded as $x = (x_1, x_2) = (0.7, 3.8)$. To decode such a particle, we simply approximate each coordinate to the upper integer to obtain the insertion positions $\vartheta = (pu, de) = (1, 4)$.

In this paper, three strategies are used for dealing with solutions that do not satisfy the precedence constraint (pickup before delivery) that appear by the generation of individuals/particles of GA and PSO:

—Penalty approach (called *P*). In this case, a penalization term is added to the objective function when some solutions (individuals/particles) generate unfeasible sequences. Specifically, a hard penalization is used by including a very high weighing factor in computing the fitness of the unfeasible solutions, regardless of the distance to the feasible region.

—Repair strategy (*R*1). The Baldwinian evolution (Hinton and Nowlan 1987) is considered to repair unfeasible individuals just for evaluation purposes, although they are not modified in the population. Then, the population is a mix of feasible and unfeasible individuals.

—Repair strategy (*R*2). Lamarckian evolution (Ackley and Littman 1994) is used to repair an unfeasible solution into a feasible one, which replaces the original unfeasible option in the population. Then, all members of the population are feasible.

The details of the entire repair procedure are presented in Online Appendix B, function *Obtain-Sequence*. Candidate solutions that do not satisfy the capacity constraint are always penalized.

According to the encoding and handling options of unfeasible solutions, six versions of PSO and three versions of GA are used: P-PSO-R, R1-PSO-R, R2-PSO-R, P-PSO-N, R1-PSO-N, R2-PSO-N, P-GA, R1-GA, and R2-GA. The prefixes P, R1, and R2 indicate the handling technique for unfeasible solutions based on penalization P, R1 for Baldwinian repair, and R2 Lamarckian repair, respectively. The suffixes R and N indicate that solutions are encoded in a continuous or integer domain, respectively. R2-GA was proposed in Sáez, Cortés, and Núñez (2008) and R1-PSO-R was proposed in Cortés et al. (2009).

3.3. Parameter Tuning for the HPC-EA-DRS

Several studies have reported the optimization of parameter tuning for PSO and GA. Most of these studies cover cognitive, social parameters, and inertia weights in case of PSO; mutation, crossover rates, and selective pressure parameters for GA. These studies offer recommended parameter sets, recognizing that these are problem dependent. Then, we perform a sensitivity analysis of these parameters; however, the obtained standard deviations turned out to be too large in order to draw any conclusions. Because of that, we decided to use a manually tuned set of

parameters for GA and a set of recommended parameters for PSO.

With regard to the remaining parameters, namely, population size and number of generations, in the literature we found studies that solve static problems (off-line mode, with no strict requirement on computation time), where the number of iterations is chosen large enough for the algorithm to converge. Besides, the population size is chosen large enough to perform a global search, without making the algorithm excessively slow. The problem faced here is dynamic, so the computational burden is very relevant. In general, because of the complexity of the system, there is no time to let the algorithms converge, and then the number of particles and maximum number of iterations are critical to the algorithm performance.

In this section, we propose a methodology to tune the remaining two parameters: the population size and the number of generations, by means of a multiobjective approach. The goal of this approach is to find the best set of combinations of these parameters in terms of quality of the solutions and computation time. Since accuracy and computation resources are clearly opposite, a multiobjective approach is proposed to set the value of these parameters. The multiobjective problem we solve is the following:

$$\begin{aligned} &\min_{n,g} \quad \{t_{n,g}, JP_{n,g}\} \\ &\text{s.t.} \quad (n,g) \in P, \end{aligned} \tag{11}$$

where n is the population size, g is the number of generations, P is the search domain of n and g. The mean computation time required for the optimization problem of each incoming request is $t_{n,g}$, using a population size n and number of generations g. The mean total cost of the system is $JP_{n,g}$ (which is a measure of the performance of the system), comprising the effective travel and waiting times for users and operational costs of the operator during the whole period of simulation, using a population size n and number of generations g. Given the stochastic nature of the algorithms, several replications are conducted considering every combination of parameters to find the mean values of the total cost $JP_{n,g}$ and computation time $t_{n,g}$.

The solution of the multiobjective problem is a region P_S called Pareto optimal set. The set of all objective function values corresponding to solutions in the Pareto optimal set is known as Pareto optimal front $P_F = \{(t_{n^i,g^i}, JP_{n^i,g^i}): (n^i, g^i) \in P_S\}$. Figure 4 shows a graphical example of the multiobjective problem presented above, where the Pareto front is highlighted.

In Figure 4, any solution $X^i = \{n^i, g^i\}$ that belongs to the Pareto optimal set is not dominated by any other $X^j = \{n^j, g^j\}$. For example, $X^1 = \{n^1, g^1\}$ does not belong to the Pareto optimal set as it is dominated by $X^2 = \{n^2, g^2\}$. Notice that the number of function evaluations $(n \cdot g)$ is not the only determining factor for the quality of the solutions. It also depends on the design of the algorithms. In fact, larger population sizes are chosen to favor a global search, while more generations are used to get solutions that are more refined. Then, there might be combinations of population sizes and number of generations that evaluate few objective functions and reach better quality solutions than other combinations that evaluate more candidate solutions. The former are optimal configurations in the Pareto set, and the latter are dominated solutions. Then, with the multiobjective approach, we can find the optimal configurations that allow finding the best solutions in the least possible computational time.

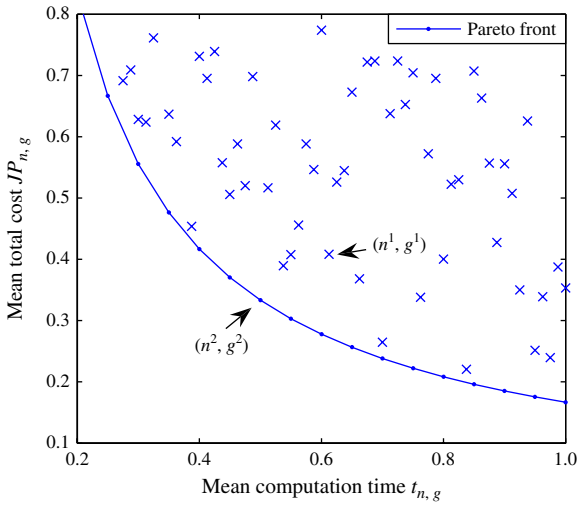


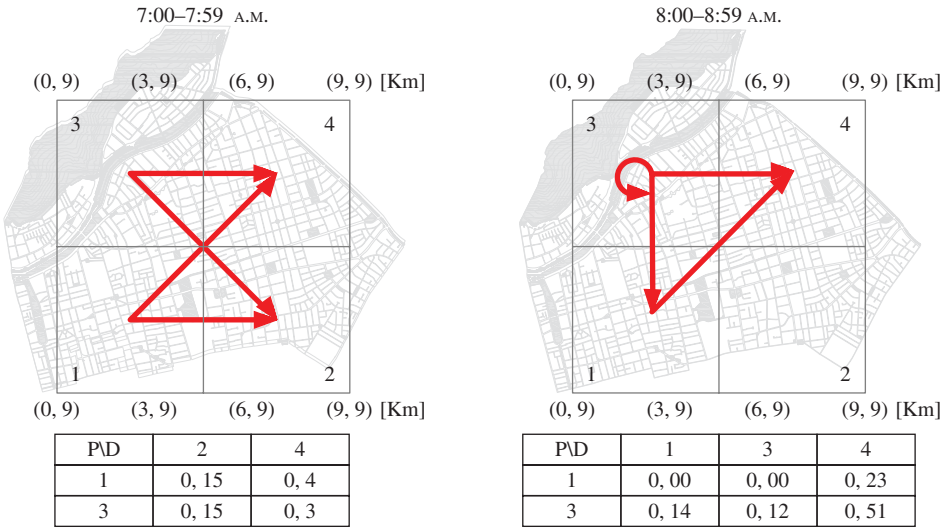
Figure 4 (Color online) Pareto Front Including Mean Total Cost and Computation Time

Once the Pareto set has been found, a single combination must be chosen according to some criterion. A natural one is to define a bound for the computation, and then the chosen combination of population size and number of generations is the one that finds the best solutions in the allowed time.

4. Simulation Experiments

4.1. Comparison of HPC-EA-DRS Methodologies

A discrete-event system simulation for a two-hour period is conducted in order to evaluate the performance of the proposed control methodology to dynamically decide the best route of vehicles according to the incoming demand, and in particular, to compare the different configurations of the considered evolutionary algorithms (GA and PSO). The scheme considers a fleet of nine small vehicles, each with space for four passengers. Dispatch and routing decisions are made by the controller in real time.



P\D	2	4
1	0, 15	0, 4
3	0, 15	0, 3

P\D	1	3	4
1	0, 00	0, 00	0, 23
3	0, 14	0, 12	0, 51

Figure 5 **(Color online) Origin-Destination Demand Patterns**

Although service requests are unknown, the average system pattern is assumed known from historical data, obtained from the average demand measured over the preceding week.

In the case study, four different future requests are considered between 7:00 and 7:59 A.M., and four more between 8:00 and 8:59 A.M. Each of these future requests triggers the discrete event model at an instant τ, and corresponds to the request that is optimized during the second level optimization (two-step ahead). Because the generated requests are just predictions, the calculated "near-optimal" sequences for such requests are not used for the dispatcher, meaning that only the sequence $S(k)$ for the effective current call is applied to the system.

The demand patterns and their probabilities are shown in Figure 5 and were determined by the zoning method used in Cortés et al. (2009).

We consider an urban service area of approximately 81 km^2. The vehicles are assumed to travel straight between stops at an average speed of 20 km/hr over the region. The experiment is repeated 30 times for each case of analysis (each configuration or PSO and GA), testing different demand requests that follow the pattern in Figure 5. The relevant indicators we measure are the average total cost in terms of waiting time, travel time, and operational cost for the entire two-hour period and the average computation time for solving the optimization problem associated with the insertion decision of every incoming request, as shown in Table 1. These tests were conducted using 10 generations and 10 individuals in the evolutionary algorithms used within the HPC-EA-DRS method. The considered configurations of evolutionary algorithms are (as introduced in §3.2): P-PSO-R, R1-PSO-R, R2-PSO-R, P-PSO-N, R1-PSO-N, R2-PSO-N, P-GA, R1-GA, and R2-GA.

Our routines for all proposed algorithms based on PSO and GA were coded in Matlab, including the different configurations explained above. The equipment utilized for the implementation of the routines is an iMac CPU Intel Core i3 (3.2 GHz, 4 GB).

Table 1 **Simulation Statistics for the Proposed Algorithms for Control the Dial-a-Ride System**

	Total cost statistics				Computation time statistics			
EA configuration	Mean [min]	Worst case [min]	Best case [min]	Standard deviation [min]	Mean [s]	Worst case [s]	Best case [s]	Standard deviation [s]
P-PSO-N	6,818.9	7,214.6	6,378.6	171.39	7.522	8.218	7.032	0.303
R1-PSO-N	6,809.0	7,136.1	6,487.8	132.79	12.509	14.291	11.686	0.515
R2-PSO-N	6,800.1	7,041.3	6,536.4	119.27	11.530	12.399	10.845	0.335
P-PSO-R	6,846.5	7,123.5	6,544.3	129.52	8.378	9.336	7.700	0.310
R1-PSO-R	6,823.6	7,168.8	6,510.5	156.27	13.468	14.868	12.888	0.455
R2-PSO-R	6,810.6	7,078.2	6,523.0	119.27	11.763	12.495	10.820	0.335
P-GA	6,787.3	7,022.4	6,582.0	90.81	27.010	28.551	24.897	0.956
R1-GA	6,803.0	7,039.4	6,581.0	107.63	24.764	26.064	23.413	0.643
R2-GA	6,796.8	6,898.5	6,689.8	43.38	31.751	33.353	30.330	0.548

Table 2 Comparison Between PSO-N and PSO-R Strategies

Performance change when shifting from a PSO-N to a PSO-R strategy		
Method	Extra computation time [%]	Total cost increase [%]
P-PSO	11.38	0.40
R1-PSO	7.67	0.21
R2-PSO	2.02	0.15

We observe that in terms of performance, the GA strategies provide better solutions than the PSO strategies. Nevertheless, GA takes twice as much time as PSO does, which supports our preference for PSO-based algorithms, mainly in case of projecting this methodology to potential real implementations. In fact, a real-time application might not justify the marginal increase in the quality of the solutions (Table 1) when considering the trade-off between solution quality and computational burden.

In Table 2 a comparison of the encoding method for PSO is presented. This table shows that the PSO-N configurations reach better solutions in shorter times than those obtained in the tests for PSO-R, under the same constraint handling technique. Thus, for the studied system, with 10 individuals and 10 generations, we conclude that PSO-N configurations perform better than PSO-R configurations. This result is reasonable considering the discrete nature of the optimization problem. In that sense, one might expect GA to perform well for this problem given its discrete encoding, although as mentioned above the computation time required for PSO is significantly less than that required for GA, which can be a determinant in the context of a real-time dispatch scheme where decisions need to be made fast enough in order to make the system work.

In Table 3, a comparison between the constraint handling techniques for PSO and GA is presented. We observe that for the PSO configurations, the schemes with repair result in better solutions than those with penalty, although they require longer computation times (Table 3). This happens because the stages involved in procedures with repair are capable of testing more candidate solutions than those with penalty. It is not clear then which strategy is better, considering the observed trade-off between computational time and objective function values. Note that this behavior does not apply to GA. The GA configuration with penalty is the best in terms of objective function. In this case, this approach seems to guide the population toward the optimum in a better fashion than the other constraint handling techniques, which result in better solutions.

For this combination of population size and generations, the previous results suggest that PSO strategies are more suitable for real-time application than GA mostly because they run faster. In addition, discrete encoding seems to be the best approach, and Lamarckian evolution shows better performance than Baldwinian evolution.

Table 3 Comparison Between Penalty and Repair Strategies

Performance change when shifting from a penalty to a repair configuration		
Method	Extra computation time [%]	Total cost decrease [%]
R1-PSO-N	66.30	0.15
R1-PSO-R	60.75	0.33
R2-PSO-N	53.28	0.28
R2-PSO-R	40.40	0.31
R1-GA	−8.32	−0.23
R2-GA	17.55	−0.14

In §4.2, the results of an optimal parameter tuning based on multiobjective optimization are presented for the evolutionary algorithms used within the HPC-EA-DRS methodology. For illustrative purposes, we chose the R2-PSO-N and R2-GA to show the tuning methodology. R2-PSO-N was used because it showed the best performance in quality solution among the PSO options for the original parameters, and R2-GA was chosen since it is the analog to R2-PSO-N among the GA approaches.

4.2. Parameter Tuning for PSO and GA in the HPC-EA-DRS Methodology

As stated before, R2-PSO-N and R2-GA have been chosen for parameter tuning. The parameters considered are the population size and number of generations. We consider 50 replications of each case, in order to find the mean computation time and mean total cost for each combination of the parameters. Both population size and number of generations vary in the range between 5 and 20. The Pareto front and other suboptimal combinations of population sizes and generations for both methods are presented graphically in Figure 6 and in detail in Table 4.

From Table 4 we appreciate that five out of the 11 points in the Pareto set associated with the R2-PSO-N configuration correspond to the smallest tested population size 5. The quality of the solutions for these points increases very fast with respect to the computation time, whereas in combinations with larger populations it increases very slowly. Moreover, the solution with 8 individuals and 11 iterations looks like a knee point. Note that the influence of the demand prediction, which is inherently stochastic, could cause some results not to follow the expected behavior. For example, the combination of 20 individuals and 20 generations should result in the best performance, which is not the case as it does not appear in the Pareto front (Table 4).

For the R2-GA strategy, we cannot observe a clear knee point as before. In addition, the points in the

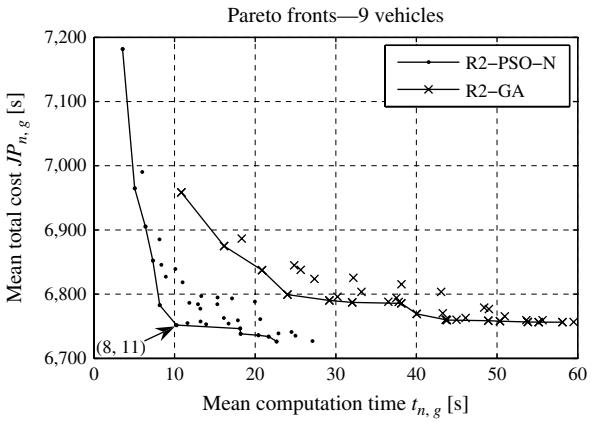


Figure 6 **Comparison of the Pareto Fronts and Dominated Solutions of Each Strategy**

Pareto optimal set are uniformly distributed for different population sizes (Table 5). This is very different from the case of R2-PSO-N, where almost half of the points correspond to the shortest population size available.

R2-PSO-N is faster than R2-GA for equal populations and generations. This is because R2-PSO-N evaluates fewer candidate solutions than R2-GA. To understand this, note that the fitness of tested solutions are stored so that it is not necessary to compute them again when a solution is repeated. Besides, it is known that PSO converges faster than GA (Nedjah and Macedo-Mourelle 2006), implying that some candidate solutions are more likely to be repeated (as the problem is discrete). Then, for a given population size and number of generations, PSO evaluates fewer candidate solutions, and therefore it runs faster.

From Figure 6, we can also notice that for a given computation time value, we always find a R2-PSO-N case that performs better than any R2-GA case. Conversely, for a given total cost value, we always find a R2-PSO-N case that is faster than any R2-GA implementation. Finally, for any R2-GA case, we can always find a R2-PSO-N case that runs faster and find a better performance as well. From these observations, we can say that the R2-PSO heuristic is more efficient than R2-GA-N for the described problem.

Table 4 **Optimal Parameters Found for R2-PSO-N**

Pareto optimal set for R2-PSO-N			
Population size	Number of generations	Mean computation time per request [s]	Mean total cost [min]
5	5	3.543	7,181.9
5	8	5.047	6,964.8
5	11	6.374	6,905.3
5	14	7.312	6,852.2
5	17	8.177	6,782.7
8	11	10.240	6,751.8
11	20	18.201	6,738.1
14	20	21.693	6,734.0
17	11	18.141	6,746.3
17	17	22.658	6,726.4
20	11	20.404	6,736.0

Table 5 **Optimal Parameters Found for R2-GA**

Pareto optimal set for R2-GA			
Population size	Number of generations	Mean computation time per request [s]	Mean total cost [min]
5	5	10.846	6,958.3
8	5	16.150	6,874.9
8	11	32.039	6,787.2
8	14	38.128	6,785.7
8	17	43.661	6,760.1
11	5	20.849	6,837.3
11	14	43.776	6,759.7
14	5	24.007	6,799.2
14	11	40.042	6,769.2
17	14	48.910	6,758.6
17	17	53.824	6,756.5
17	20	58.124	6,756.4
20	5	29.215	6,790.2
20	14	50.362	6,757.9
20	17	55.171	6,756.4

Finally, we have to say that in real implementations of a dial-a-ride system, one necessary condition is that the solution must be obtained quick enough to become acceptable for the service provider. In this case, we can easily choose the parameters that reach the best solutions in a reasonably short time. Thus, for example, in PSO, if we use a maximum decision time of 10 seconds for every request as a criterion, we can choose five individuals and 17 generations, which reaches the best mean total cost (6,782 min) among those reachable in a computation time that is less than 10 seconds (8.18 s).

4.3. HPC-EA-DRS with Different Fleet Size

Next, in order to see how extendable the results to other configurations of dial-a-ride systems are, we conduct some tests under different scenarios. The experiment considers different fleet sizes. A total of seven and 11 vehicles were tested to verify the properties of the algorithms R2-GA and R2-PSO-N. Figure 7 with the corresponding Tables 6 and 7 are the results for the case study with seven vehicles, and Figure 8 with Tables 8 and 9 comprises the results of 11 vehicles.

As can be seen in the results, in all cases the systematic better performances of R2-PSO-N in terms of Pareto optimal solutions with respect to R2-GA remains. The case study selected in the previous section represents a good example to show the characteristics of the algorithms. We can expect similar results under different fleet sizes for dial-a-ride set ups.

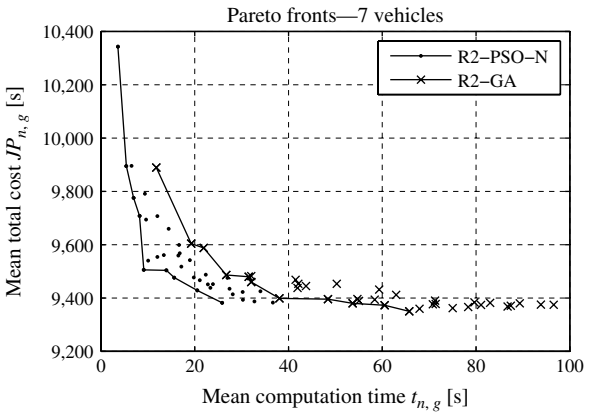


Figure 7 **Comparison of the Pareto Fronts and Dominated Solutions of Each Strategy**

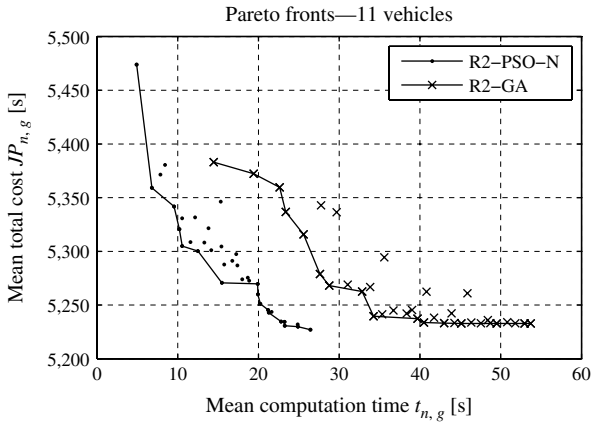


Figure 8 **Comparison of the Pareto Fronts and Dominated Solutions of Each Strategy**

4.4. Comparison of HPC-EA-DRS with a Double-Horizon Heuristic

In this section, the proposed HPC-EA-DRS strategy is compared with a state-of-the-art heuristic method based on a double-horizon-based strategy (Mitrovic-Minic, Krishnamurti, and Laporte 2004) for solving the dynamic pickup and delivery problem. A double-horizon heuristic solves a dynamic problem assuming that the distribution of the probable future requests is unknown. The heuristic method considers the necessity of making good short-term decisions without experiencing adverse long-term effects. At the short-term stage, a static optimization is performed with the objective of minimizing the cost of serving the incoming request (coming up in real time). At the long-term stage, the aim is to provide enough room (slack) for making future insertions easier. The heuristic originally is designed for solving a DPDP version with time windows, and therefore, in the long-term the goal is to maximize the slack time to make insertion easier. In our formulation however, there are no explicit time windows; in this case the most important constraint is given by the capacity in the vehicles; therefore, our aim is in providing enough room to satisfy future requests at a good level of service. The

Table 6 **Optimal Parameters Found for R2-PSO-N**

Pareto optimal set for R2-PSO-N (7 vehicles)			
Population size	Number of generations	Mean computation time per request [s]	Mean total cost [min]
5	5	3.403	10,343.0
5	8	5.100	9,894.7
5	11	6.546	9,775.6
5	14	7.742	9,708.1
5	17	8.614	9,505.7
8	14	13.185	9,503.8
8	17	14.755	9,476.5
14	11	19.352	9,428.3
14	17	24.430	9,381.5

Table 7 **Optimal Parameters Found for R2-GA**

Pareto optimal set for R2-GA (7 vehicles)			
Population size	Number of generations	Mean computation time per request [s]	Mean total cost [min]
5	5	11.133	9,889.4
5	8	20.680	9,587.8
5	11	29.734	9,479.7
8	5	18.153	9,603.7
8	14	50.666	9,380.1
11	5	25.234	9,486.0
11	14	62.077	9,350.0
14	5	30.260	9,459.9
14	8	45.711	9,395.7
14	11	57.105	9,372.7
17	5	35.917	9,398.5

Table 8 **Optimal Parameters Found for R2-PSO-N**

Pareto optimal set for R2-PSO-N (11 vehicles)			
Population size	Number of generations	Mean computation time per request [s]	Mean total cost [min]
5	5	4.648	5,473.9
5	8	6.419	5,359.1
5	14	9.020	5,341.8
5	17	9.979	5,304.9
8	11	11.815	5,300.3
8	17	14.610	5,270.6
11	5	9.626	5,320.7
11	20	19.112	5,251.1
14	14	18.809	5,259.8
14	17	20.083	5,242.9
14	20	21.524	5,234.7
17	11	18.791	5,269.8
17	17	21.944	5,230.8
17	20	23.492	5,229.8
20	11	20.019	5,245.4
20	14	21.923	5,234.3
20	20	24.961	5,227.0

Table 9 Optimal Parameters Found for R2-GA

Pareto optimal set for R2-GA (11 vehicles)			
Population size	Number of generations	Mean computation time per request [s]	Mean total cost [min]
5	5	13.643	5,383.1
5	8	21.363	5,359.6
8	5	18.313	5,372.2
11	5	22.067	5,336.7
14	5	24.149	5,315.7
14	8	31.001	5,262.6
14	14	40.579	5,233.1
17	5	26.067	5,278.9
17	8	32.338	5,239.6
17	11	37.468	5,237.2
17	20	49.976	5,232.8
20	5	27.157	5,268.2
20	11	38.227	5,233.7
20	14	42.585	5,232.8
20	17	46.725	5,232.8
20	20	50.698	5,232.8

long-term optimization stage then will aim at penalizing sequences where the vehicle load is beyond a certain threshold.

Based on preliminary testing, we realized that the best results are obtained when the static problem is solved for the entire horizon, and the long-term goal acts as a penalization. In such a case, the objective function for the double-horizon heuristic is given by

$$\min_{S_k^{k+1}} \sum_{j=1}^{F} (\tilde{C}_j(k+1) - \tilde{C}_j(k)) \tag{12}$$

with

$$\tilde{C}_j(k+1) = C_j(k+1) + \sum_{i=w_{j,\,\text{long-term}}}^{w_j(k+1)} K \max(\hat{L}_j^{i-1}(k+1) - L_{\min} + 1, 0), \tag{13}$$

where $C_j(k+1)$ is the same specification as that defined in (8). The first term of (13) implies a static optimization of the incoming request and the second term penalizes the case of vehicle loads equal or greater than a parameter $L_{\min}$, proportionally to both K and $\hat{L}_j^{i-1}(k+1) - L_{\min} + 1$.

For comparing this heuristic method with our implementation, we run the case with nine vehicles based on the results of §4.2 corresponding to R2-PSO-N with eight individuals and 11 iterations (the knee point). For the double-horizon heuristic penalty, the parameter factor K and load threshold $L_{\min}$ are properly tuned; the long-term horizon starts at the 2nd half (rounded up) of the stops sequence. Table 10 shows the different parameters (K and $L_{\min}$) tuned with 2,000 randomly generated sequences. From the table, the best parameters for the heuristic are given by $K = 10$, $L_{\min} = 4$.

Table 10 Optimal Parameters Found for the Double-Horizon Heuristic

Parameter tuning for a double-horizon heuristic		
	Mean total cost [min]	
$K \backslash L_{\min}$	3	4
8	7,271.0	7,236.1
10	7,262.1	**7,226.1**
12	7,263.5	7,263.0
15	7,304.9	7,427.0
20	7,488.4	7,903.8

Table 11 Methods Comparison

Performances and computational times for the one-step-ahead method, R2-PSO-N and the double-horizon heuristic			
Method	Mean total cost [min]	Improvement [%]	Mean computation time per request [s]
One-step-ahead	7,274.8	0	0.034
R2-PSO-N	6,580.7	9.54	10.17
Double horizon	7,177.2	1.34	0.039

The mean performances and computational times for 2,000 randomly generated sequences are presented in Table 11, considering that the one-step-ahead method, R2-PSO-N and the double-horizon heuristic with their optimal settings (by tuning the different parameters of each specification) are used to control the system. We can observe that both R2-PSO-N and the double-horizon heuristic offer an improvement over the (myopic) one-step-ahead method; however, it is clear that the R2-PSO-N implementation gain is considerably larger than that of the double-horizon heuristic (almost 10% against 1.34%). The extra improvement requires of course more computation resources mainly involved in the prediction of future events phase, as shown in the table. However, the computation time spent by R2-PSO-N is still within reasonable ranges for a real-time implementation, which remains bounded and under control because of the nature of the EA algorithms we implemented.

5. Conclusions and Final Remarks

In this paper, we develop an ad-hoc methodology (HPC-EA-DRS) to solve the HPC formulation for the dial-a-ride system based on generic evolutionary algorithms. The formalization of the methodology is valid for a large variety of evolutionary algorithms as well as other heuristics of similar characteristics, and is based on generic operations performed by the evolutionary algorithms. In this work, we show nine specific implementations of the evolutionary algorithms (PSO and GA), including repair and penalty approaches that result in several variants of

the generic operations in the HPC-EA-DRS to handle the specific constraints of the problem behind the real-time operation of a dial-a-ride service provider. Using a first combination of population size and number of generations, we found that PSO with integer encoding and Lamarckian repair performs the best for the studied applications. The results are reasonable considering the embedded integral nature of most of the variables that describe a dial-a-ride system under an HPC scheme, such as that previously proposed by the authors.

With regard to the studied evolutionary algorithms (PSO and GA), we identified some configurations that run faster than others although they lose accuracy in performance. Thus, the strategy that is suitable for a specific application will strongly depend on the final objective pursued by the dispatcher. For instance, if decisions have to be made in a very short time (real time), we probably will choose the fastest available option regardless of the accuracy of the obtained solution (with different encoding and constraint handling methods).

With regard to the observed trade-off between accuracy and computation, we proposed a multiobjective approach for tuning the parameters of any strategy implementation, considering computation time and performance as the conflicting objectives. The output of the analysis is the set of combination of parameters that belong to the Pareto optimal set. This approach is applied to the R2-PSO-N and R2-GA-based solution methods. The analysis allows comparing both strategies, as well as defining the optimal parameter setting according to a predefined criterion, such as the presence of a knee point or a maximum allowed computation time. It is shown that PSO strategies are more efficient than those based on GA, because the former are faster and more accurate. We believe that PSO performs better than GA because the search heuristic of PSO is more suitable to the analyzed problem, which is clearly reflected by the fact that PSO finds better solutions, even evaluating fewer candidates. In addition, we tested our best implementation against a well-known heuristic method from the literature, obtaining much better performance from our method in case of adding prediction to the optimization scheme, at the expense of a higher (although controllable) computational cost.

In further research, we expect to test the methodology and calibrate the parameters for real-size dial-a-ride configurations. Other methods and heuristics, such as differential evolution, will also be developed and contrasted with the best PSO implementations. Possible extensions in the way the continuous solutions are converted into an integer version are also topics to be investigated in the future.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/trsc.2014.0569.

Acknowledgments

The authors thank the financial support of CONICYT/FONDECYT/REGULAR/N°1141313, and the Millennium Institute "Complex Engineering Systems (ICM: P-05-004-F, CONICYT: FBO16)."

Appendix A

The vehicle load behavior is obtained using the following state space model:

$$E\{L_j(k+1)/k\} = \hat{L}_j(k+1) = A_L L_j(k) + B_L(S_j(k)),$$

where the corresponding matrices in (7) are

$$B_L(S_j(k))_{(w_j(k)+1)\times 1} = B_L^2 \cdot (S_j(k) \cdot B_L^1);$$

$$A_L = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix}_{(w_j(k)+1)\times(w_j(k-1)+1);}$$

$$B_L^1 = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}_{1\times 4}; \quad B_L^2 = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 1 & 1 & \cdots & 1 & 0 \end{bmatrix}_{(w_j(k)+1)\times(w_j(k)+1).}$$

Both the vehicle sequence matrix $S_j(k)$ and the expected load vector $\hat{L}_j(k+1)$, change their dimension dynamically by adding two rows when a new request occurs. Therefore, the matrix dimensions of A_L, B_L^1, B_L^2 are variable. The matrix B_L^1 is designed to remove the last two columns of the sequence vector, which are not necessary for representing load changes from step k to step $k+1$. On the other hand, when a request is satisfied, the first row of the sequence is eliminated. In fact, the adaptive behavior is captured by these techniques of expansion and reduction of matrix size.

The vehicle departure time behavior is obtained by using the same methodology. Analytically,

$$E\{T_j(k+1)/k\} = \hat{T}_j(k+1) = A_T \cdot T_j(k) + B_T(S_j(k)),$$

where

$$B_T(S_j(k))_{(w_j(k)+1)\times 1} = B_T^2 \cdot (S_j(k) \cdot B_T^1);$$

$$A_T = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & \cdots & 0 \end{bmatrix}_{(w_j(k)+1)\times(w_j(k-1)+1);}$$

$$B_T^1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}_{4\times 1}; \quad B_T^2 = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}_{(w_j(k)+1)\times(w_j(k)+1).}$$

As in the load state space model, the matrices A_T, B_T^1, B_T^2 change their dimensions dynamically.

References

Ackley DH, Littman ML (1994) A case for Lamarckian evolution. Langton CG, ed. *Artificial Life III* (Addison-Wesley, Reading, MA), 3–10.

Bemporad A, Morari M (1999) Control of systems integrating logic, dynamics and constraints. *Automatica* 35:407–427.

Berbeglia G, Cordeau JF, Laporte G (2010) Dynamic pickup and delivery problems. *Eur. J. Oper. Res.* 202(1):8–15.

Bertsimas D, Van Ryzin G (1991) A stochastic and dynamic vehicle routing problem in the Euclidean plane. *Oper. Res.* 39(4): 601–615.

Bertsimas D, Van Ryzin G (1993a) Stochastic and dynamic vehicle routing problem in the Euclidean plane with multiple capacitated vehicles. *Oper. Res.* 41(1):60–76.

Bertsimas D, Van Ryzin G (1993b) Stochastic and dynamic vehicle routing with general demand and interarrival time distributions. *Appl. Probability* 25:947–978.

Cortés CE, Sáez D, Núñez A (2008) Hybrid adaptive predictive control for a dynamic pickup and delivery problem including traffic congestion. *Internat. J. Adapt. Control Signal Processing* 22(2):103–123.

Cortés CE, Sáez D, Núñez A, Muñoz-Carpintero D (2009) Hybrid adaptive predictive control for a dynamic pickup and delivery problem. *Transportation Sci.* 43(1):27–42.

Dial R (1995) Autonomous dial-a-ride transit—Introductory overview. *Transportation Res. Part C* 3:261–275.

Dréo J, Pétrowski A, Siarry P, Taillard E (2006) *Metaheuristics for Hard Optimization Methods and Case Studies* (Springer-Verlag, Berlin).

Eksioglu B, Volkan A, Reisman A (2009) The vehicle routing problem: A taxonomic review. *Computers Indust. Engrg.* 57(4): 1472–1483.

Gendreau M, Guertin F, Potvin J, Taillard E (1999) Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Sci.* 33(4):381–390.

Haghani A, Jung S (2005) A dynamic vehicle routing problem with time-dependent travel times. *Comput. Oper. Res.* 32: 2959–2986.

Hegyi A, De Schutter B, Hellendoorn H (2005) Model predictive control for optimal coordination of ramp metering and variable speed limits. *Transportation Res. Part C* 13:185–209.

Hinton G, Nowlan S (1987) How learning can guide evolution. *Complex Systems* 1:495–502.

Jaw J, Odoni A, Psaraftis H, Wilson N (1986) A heuristic algorithm for the multivehicle many-to-many advance-request dial-a-ride problem. *Transportation Res. Part B* 20:243–257.

Jih W-R, Hsu JYJ (1999) Dynamic vehicle routing using hybrid genetic algorithms. *Proc. 1999 IEEE Internat. Conf. Robotics and Automation* (IEEE, Detroit), 453–458.

Karer G, Mušič G, Škrjanc I, Zupančič B (2007a) Hybrid fuzzy modeling for model predictive control. *J. Intelligent Robotic Systems* 50:297–319.

Karer G, Škrjanc I, Mušič G, Zupančič B (2007b) Hybrid fuzzy model-based predictive control of temperature in a batch reactor. *Comput. Chemical Engrg.* 31:1552–1564.

Kennedy J, Eberhart R (2001) *Swarm Intelligence* (Morgan Kaufmann Publishers, Burlington, MA).

Kleywegt A, Papastavrou J (1998) The dynamic and stochastic knapsack problem. *Oper. Res.* 46(1):17–35.

Larsen A (2000) The dynamic vehicle routing problem. Unpublished doctoral thesis, Technical University of Denmark, Lyngby, Denmark.

Madsen O, Raven H, Rygaard J (1995) A heuristics algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Ann. Oper. Res.* 60:193–208.

Man K, Tang K, Kwong S (1998) *Genetic Algorithms, Concepts and Designs* (Springer-Verlag, Berlin).

Mitrovic-Minic S, Laporte G (2004) Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Res. Part B* 38:635–655.

Mitrovic-Minic S, Krishnamurti R, Laporte G (2004) Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Res. Part B* 38:669–685.

Montemanni R, Gambardella L, Rizzoli A, Donati A (2005) Ant colony system for a dynamic vehicle routing problem. *J. Combinatorial Optim.* 10(4):327–343.

Nedjah N, Macedo-Mourelle L (2006) *Swarm Intelligent Systems* (Springer-Verlag, Berlin).

Núñez A, Sáez D, Oblak S, Škrjanc I (2009) Fuzzy-model-based hybrid predictive control. *ISA Transactions* 48(1):24–31.

Osman M, Abo-Sinna M, Mousa A (2005) An effective genetic algorithm approach to multiobjective routing problems (MORPs). *Appl. Math. Computation* 163:769–781.

Psaraftis H (1980) A dynamic programming solution to the single many-to-many immediate request dial-a-ride problem. *Transportation Sci.* 14(2):130–154.

Psaraftis H (1988) Dynamic vehicle routing problems. Golden BL, Assad AA, eds. *Vehicle Routing Methods and Studies* (North Holland, Amsterdam), 223–248.

Sáez D, Cortés CE, Núñez A (2008) Hybrid adaptive predictive control for the multi-vehicle dynamic pickup and delivery problem based on genetic algorithms and fuzzy clustering. *Comput. Oper. Res.* 35:3412–3438.

Skrlec D, Filipec M, Krajcar S (1997) A heuristic modification of genetic algorithm used for solving the single depot capacitated vehicle routing problem. *Proc. Intelligent Inform. Systems* (IEEE, Los Alamitos, CA), 184–188.

Swihart M, Papastavrou J (1999) A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *Eur. J. Oper. Res.* 114:447–464.

Tarantilis C (2005) Solving the vehicle routing problem with adaptive memory programming methodology. *Comput. Oper. Res.* 32:2309–2327.

Thomas B, White C III (2004) Anticipatory route selection. *Transportation Sci.* 38(4):473–487.

Toth P, Vigo D (2003) The granular tabu search and its application to the vehicle-routing problem. *INFORMS J. Comput.* 15(4): 333–346.

Xiang Z, Chu C, Chen H (2008) The study of a dynamic dial a ride problem under time-dependent and stochastic environments. *Eur. J. Oper. Res.* 185:534–551.

Zhu Q, Qian L, Li Y, Zhu S (2006) An improved particle swarm optimization algorithm for vehicle routing problem with time windows. *Proc. IEEE Congress on Evolutionary Comput. Internat.* (IEEE, Vancouver), 1386–1390.