Neuroevolution of a multi-agent system for the dynamic pickup and delivery problem

Jonathan Merlevede, Rinde R.S. van Lon, and Tom Holvoet

iMinds-DistriNet, KU Leuven, 3001 Leuven, Belgium.
{firstname.lastname}@cs.kuleuven.be

Abstract Automated transportation and logistics create particularly challenging problems for planners and schedulers. Besides their computational hardness, such systems need to cope with the dynamic and distributed nature of the problems. This article describes an agent-based approach to the dynamic pickup and delivery problem (PDP). We investigate the feasibility of using the neuroevolution of augmenting topologies (NEAT) algorithm to create and optimize a multi-agent system (MAS) for dynamic PDPs. A thorough feasibility study requires a significant effort since a platform is required that facilitates a comparison of MAS and centralized algorithms. We implemented an existing benchmark dataset for the dynamic pickup and delivery problem with time windows (PDPTW) in the RinSim multi-agent simulator. We supplied the NEAT algorithm with training data derived from this dataset and we deployed the resulting neural network in a homogenous MAS that uses a blackboard coordination model. Our preliminary results show that our approach is a double-edged sword, the resulting MAS responds in real-time (response time in ms) but the solution quality is worse compared to that of the benchmark dataset.

Keywords: pickup and delivery problems with time windows (PDPTW), artificial neural networks (ANN), neuroevolution of augmenting topologies (NEAT), multi-agent systems (MAS), distributed systems, decentralized, optimization, logistics, transportation, dynamic, real-time

1 Introduction

In 2012, total U.S. business logistics cost $1.28 trillion dollar, a 6.6 percent increase from the previous year and accounting for 8.5 percent of the U.S. gross domestic product [42]. Of these costs, 52% were due to transportation by truck or airplane. Clearly, the calculation of transportation routes is of tremendous economic value.

Constructing transportation routes is an intractable problem that has been the topic of intensive research for over half a century. In part due to computational limitations, many logistics companies impose constraints on their customers. For example, it is common practice to make customers order before a certain deadline and to process their requests only after this deadline expires.
In essence, this transforms an inherently dynamic problem into a static problem suitable for ordinary routing algorithms [25, 26]. Transforming a dynamic problem into a static one is not always appropriate or possible, e.g. in the case of taxi services, emergency services and specialized courier services. For these dynamic problems most existing techniques involve adapting an algorithm that creates routes for the static version of the problem [4, 9, 27]. Since solving a static problem each time new information is revealed is computationally intensive, the usual approach is to start by computing a good set of routes. As new information becomes available, these routes are updated using heuristic methods, for example by inserting, deleting or interchanging destinations in the existing routes.

In our research we investigate whether using a multi-agent system (MAS) poses a viable alternative to conventional centralized algorithms by embracing both the dynamism and the distributed nature of a problem. More specifically, in this paper we apply neuroevolution of augmenting topologies (NEAT) to create a MAS for the uncapacitated dynamic pickup and delivery problem with time windows (PDPTW). In order to evaluate the feasibility of this approach a thorough evaluation with state-of-the-art centralized algorithms is essential, as is argued in [18]. In the dynamic PDPTW there are two algorithmic properties of interest, (1) the solution quality according to the objective function and (2) the reaction time of the algorithm. Currently there are very few MAS papers that make an effort to compare with state-of-the-art centralized algorithms. An example of a paper with a good evaluation is the paper by Máhr et al. [21]. The reason for this lack of focus on good evaluation may well be due to its time consuming nature and an apparent reluctance from the MAS community in evaluation beyond a prototype.

This paper presents our first steps towards a NEAT and MAS based approach and a thorough evaluation. We start with an overview of related work (Sect. 2). We then give a precise definition of the dynamic PDPTW we are addressing (Sect. 3) and detail our NEAT and MAS approach (Sect. 4). We evaluate our approach by using the open-source RinSim simulator [17] and a dataset created by Gendreau et al. [9] (Sect. 5). Section 6 concludes and provides possible directions for future work.

2 Related work

We divide our discussion of related work into two groups: literature on MASs for dynamic pickup and delivery problems (PDPs) (Sect. 2.1) and literature on NEAT (Sect. 2.2). For a literature survey on the PDP we refer to [4, 25, 26].

2.1 Multi-agent systems

Many MAS-based approaches to the dynamic PDP exist in the literature. The common approach is to model each vehicle as an agent. Although other kinds of agents representing e.g. distribution companies are often used, we use the terms ‘vehicle’ and ‘agent’ interchangeably.
Applicability The influential work of Fischer et al. [7, 8] states the applicability of MAS-based solutions to dynamic PDPs. They argue that many of the difficulties of the transportation domain (inherently distributed, highly dynamic, complex and the existence of cooperation) are naturally captured and alleviated by a MAS. More recently, Máhr et al. [21] have shown that even fully decentralized (‘flat’) MAS-based based approaches to the PDPTW can perform as well as state-of-the-art centralized algorithms based on mixed integer programming, especially in highly dynamic scenarios.

Coordination When a group of agents needs to make a collective decision such as who gets to serve a new request, a coordination mechanism such as bidding, auctioning or voting is typically used. This is the case in [8], which extends the contract-net protocol using a global auction mechanism that the authors call simulated trading. The authors claim that global information in the form of auctioning can significantly improve the allocation of goods to vehicles.

Evolution Beham et al. [3] were among the first to combine a MAS-based approach to the uncapacitated dynamic PDPTW with an evolutionary algorithm (EA) [6]. They define vehicle behavior using two heuristic functions or agent heuristics: one determines where agents travel to, the other what goods to pick up upon arrival at a pickup or delivery point. Their agent heuristics are weighted sums of hand-constructed heuristic functions, where an evolution strategy is used to evolve the weights. They use randomly generated scenarios to show that their approach produces valid routes and that evolved heuristics can, to a limited extent, be used to produce routes for other scenarios than the one used for training. They do not compare their results with those of other algorithms.

Van Lon et al. [19] use a similar approach, but use only a single agent heuristic instead of two, and genetic programming (GP) instead of an evolution strategy. Their agent heuristic assigns a priority to all unhandled requests, which is either a pickup or a delivery of a parcel that is already in the agent’s cargo. Agents continuously re-evaluate their agent heuristic on all parcels, allowing for diversion. Their results indicate that the evolved agent heuristic outperforms the hand constructed centralized hyper-heuristic by a large margin.

The paper of Vonolfen et al. [40] continues in this line of research. They start from [3] but use GP with a large number of variables, including variables that give information about other agent’s distances and destinations. They slightly outperform [3]. They also compare with an unspecified tabu-search algorithm which produces slightly better routes.

In [3, 19, 40], there is no explicit coordination. There is implicit coordination only through the hand-designed input heuristics and a common collection of open pickup requests. In [19], agents change this collection only after performing a pickup, which means that several agents may be driving to the same pickup or delivery point at the same time. Beham et al. [3] do not specify when agents are informed.
2.2 NEAT

NEAT is an evolutionary algorithm [6] technique that evolves both the weights and topology of artificial neural networks (ANNs) [34,38]. NEAT initializes a uniform population of ANNs without hidden neurons and a fully connected input and output layer. It gradually complexifies these ANNs by introducing additional neurons and connections when it finds that they increase performance.

NEAT has proven that it is capable of devising complex behavior on numerous occasions [14,29,37–39,43] and has become popular as a benchmark algorithm in the field of neuroevolution [15,23,32,33]. The large number of algorithms that have NEAT at their core [5,12,30,35,36] illustrate its flexibility.

Feature-selective NEAT (FS-NEAT) is a minor variation of NEAT introduced in [41]. FS-NEAT differs from NEAT only in the initialization of the population: instead of a homogeneous population of networks with fully connected input and output layers, FS-NEAT uses a population of ANNs that contain only a single connection between an arbitrary input and an arbitrary output. In this way, FS-NEAT implicitly performs feature selection. FS-NEAT outperforms NEAT in the presence of inputs that are irrelevant to the task, and its performance remains nearly constant with increasing numbers of irrelevant inputs. NEAT and FS-NEAT have similar performance in the absence of irrelevant inputs.

To the best of our knowledge, NEAT has never been applied to a transportation or logistics problem before.

3 Problem definition

This section gives a precise definition of the dynamic PDPTW that we investigate (Sect. 3.1), as well as a description of the scenarios that we use for training (Sect. 3.2).

3.1 Pickup and Delivery Problem

In the dynamic PDPTW a fleet of vehicles handles a set of transportation requests $R$ that arrive over time while vehicles are working. We use the definition from [9] which is a dynamic version of the uncapacitated static PDPTW from [31].

In this definition, nothing is known about the size and distribution of $R$. A request consists of pickup and delivery locations and corresponding time windows and an ‘announce time’, which is the point in time when the request is made known to the system. A request is considered satisfied or handled when a parcel is transported from its pickup to its delivery location. Time windows are half-open: vehicles cannot service a request before the beginning of the appropriate time window, but can still service a request after the end of that window. Other properties are as follows.

- Vehicles always travel at the same speed.
- Pickups and deliveries always require the same amount of service time.
– Vehicles can travel from point to point in a straight line.
– Vehicles start at the same depot at the beginning of the scenario.
– Vehicles are expected back at that depot at the end of the scenario.
– Vehicles are not allowed to divert from their destination once they have started driving.

Allowing diversion can increase the quality of routes [13], but is not allowed in the definition of [9]. The goal of the dynamic PDPTW is to minimize a sum of three criteria: total travel time, the sum of tardiness over all pickup and delivery locations and the sum of overtime over all vehicles. Equation 1 gives the resulting cost function.

\[
\sum_{k \in M} T_k + \sum_{v \in V} \max\{0, t_v - l_v\} + \sum_{k \in M} \max\{0, t_k - l_0\}
\]

In this equation, \(M\) is the set of vehicles, \(T_k\) is the total travel time by vehicle \(k\), \(V\) is the set of all pickup and delivery locations, \(t_v\) is the time at which the time window for service at pickup or delivery point \(v\) ends, \(l_v\) is the time at which pickup or delivery point \(v\) is serviced, \(t_k\) is the time at which vehicle \(k\) arrives back at the depot and \(l_0\) is the time at which the scenario ends.\(^1\) The problem ends only after all requests are handled and all vehicles are back at the central depot.

### 3.2 Scenario generation

A specific problem instance or scenario consists of a collection of requests. In order to be able to run an EA without overfitting we need a separate dataset for training. To create such a dataset, we implemented a scenario generator that follows the specification in [9]. The intention is for generated scenarios to be feasible and more or less realistic.\(^2\) The probability of a request appearing varies over both space and time as defined by inputs to the generator. We say that scenarios that are generated by the scenario generator using same parameters belong to the same scenario class. Since scenario generation is a stochastic process, individual scenarios belonging to the same class can be very different, and scenarios belonging to different classes can be very similar.

### 4 Approach

This section presents the MAS-based approach to the dynamic PDPTW that we use in our study of using NEAT in MASs. Like [3, 19, 40], we use an EA to combine the information from several hand-constructed heuristics into an agent heuristic that determines agent behavior. We first describe how the agent heuristic determines agent behavior (Sect. 4.1). We then present the set of hand-constructed heuristics that we use in our evaluation (Sect. 4.2).

\(^1\) Although [9] defines a more general cost function that allows the different components to have different weights, it only reports on experiments with all weights equal to one. We discarded the weights to allow a direct comparison of results.

\(^2\) The generator can create scenarios that are infeasible (impossible to complete before the scenario ends). We decided not to fix this, sticking to the description of [9].
4.1 Agent behavior

Whenever requests are announced, they are placed in a collection of requests $L$ stored on a central blackboard (i.e., a list that can be accessed and modified by all agents). Each agent $a$ also has a private collection of requests $R_a$ that $a$ has picked up but not yet delivered. Whenever $a$ is idle, it constantly checks to see if $L \cup R_a$ is empty. If it is not, it applies its agent heuristic to all requests in $L \cup R_a$ and selects request $r$ with the lowest value.\(^3\) If $a$ can reach $r$ early, i.e., before the beginning of the (pickup or delivery) time window associated with $r$, it waits until either a new request is added to $L$ or it can no longer reach $r$ early. It then again becomes idle. If $a$ cannot reach $r$ early, $a$ starts driving to $r$. Upon arrival, $a$ picks up or delivers $r$’s parcel and then again becomes idle. Additionally, before starting to drive, $a$ moves $r$ out of $L$ or $R_a$. If $r \in L$, then $r$ has not yet been picked up, and $a$ claims $r$ by moving $r$ from $L$ to $R_a$. This prevents other agents from also driving to $r$’s pickup destination. If $r \in R_a$, then the parcel associated with $r$ is already in $a$’s cargo. Since $a$ is in the process of delivering the parcel and cannot be interrupted, the request is handled and $a$ removes $r$ from $R_a$. Whenever $a$ is idle, $L \cup R_a$ is empty and $a$ can no longer reach the central depot before the end of the scenario, $a$ drives to the depot.

This procedure implements the wait-earliest waiting strategy, which performs well without requiring large fleets of vehicles [22, 28]. As in [3, 19], there is only implicit coordination through the shared collection $L$.

4.2 Evolving heuristics

We evolve the agent heuristic using NEAT, which requires a fixed number of inputs and outputs. We define only a single output, which serves as the heuristic value and defines vehicle behavior as described in the previous subsection.

Because the number of inputs is fixed, we cannot use inputs like the locations of all open requests. Instead, we define a fixed number of simple input heuristics that summarize the information on a specific agent-request pair. These input heuristics should capture as much of the available information as possible. Any uncaptured information is not accessible to NEAT and therefore not used by any agent. We use the following values as inputs for request $r$ for agent $a$; some of these are taken from [3].

- **Waiters** The number of truck agents that are waiting on $r$.
- **CargoSize** The number of parcels in the cargo of $a$.
- **IsInCargo** Whether or not $r$ is in the cargo of $a$.
- **TimeUntilAvailable** Time remaining until $r$ can be served. This is equal to the time remaining until the beginning of the p/d (pickup or delivery) time window, minus the time required for $a$ to travel to $r$’s p/d point. We do not allow this value to become negative.

\(^3\) Ties are broken on a first-come, first-served basis where parcels in $L$ take priority over parcels in $R_a$.

\(^4\) The pickup location of $r$ if $r \in L$ and the delivery location of $r$ if $r \in R_a$. 
**Ado** The average distance between r’s p/d point and the delivery locations of the parcels that a is carrying.

**Mido** The minimum of the distance values between r’s p/d point and the delivery locations of the parcels that a is carrying.

**Mado** The minimum of the distances values between r’s p/d point and the delivery locations of the parcels that a is carrying.

**Dist** The distance from a to r’s p/d point.

**Urge** The urgency of a waiting order, defined as the difference between the end of r’s p/d time window and the current time. This value can become negative.

**Est** The difference between the start of the p/d time window and the current time. This value can become negative.

**Ttl** The time to live, which we define as the time that is left until the end of the scenario. This value can become negative.

**Adc** The average distance of r’s p/d location to all agents excluding a.

**Midc** The distance of r’s p/d location to the closest vehicle excluding a.

**Madc** The distance of r’s p/d location to the farthest vehicle excluding a.

We do not know whether all these heuristics are relevant for solving the problem, and therefore use FS-NEAT instead of regular NEAT.

Note that like in [3, 19, 40], the calculation of the input heuristics requires knowledge of the location of all agents. Because of this and because of the common collection of open requests, our MAS is not fully decentralized or ‘flat’.

### 4.3 Simulation-based fitness evaluation

NEAT requires a fitness or quality measure of the heuristics, that it evolves. We cannot analytically deduce this fitness, and therefore assign fitness values based on the performance of the heuristic in a number of simulated scenarios.

Simulations can end in one of two ways. Either the scenario has ended and all vehicles are back at the depot, or the simulation time has exceeded a predefined limit (which is far greater than the length of the scenario). In the latter case, we assign it a fitness of zero. In the first case, we use (1) as a measure of the quality of an individual. Because NEAT only supports maximizing positive fitness values, we subtract (1) from a large positive constant. Preliminary experiments indicate that this way of transforming the cost outperforms inversion and linear ranking.

We take three measures to make sure that fitness values are accurate measures of the relative quality of heuristics. Firstly, we compute fitness as the average performance over a number of simulations to prevent overfitted heuristics from having high fitness. Secondly, we fix the random seed of our simulator to ensure each heuristic is evaluated on a problem of the same difficulty. Thirdly and lastly, we forego a common optimization and re-evaluate all heuristics in the population each generation, including elite heuristics (i.e. heuristics that survived from the previous generation). This prevents elites from changing only in generations where evaluation is based on easy scenarios, and prevents lucky individuals from having a large influence on the evolutionary run.
5 Evaluation

This section evaluates the approach described above. It sketches the evaluation setup (Sect. 5.1), describes the performed evolutionary runs (Sect. 5.2) and presents our results together with the results reported in [9] (Sect. 5.3). It concludes with an analysis of these results (Sect. 5.4).

5.1 Evaluation setup

We perform simulations using RinSim, a high-quality, open source simulator written in Java that specifically targets the family of transportation and logistics problems [19]. We also use SharpNEATv2 [11], an implementation of NEAT in C#. SharpNEAT is a mature project and has been used in the literature [2,16,20,29], even though it differs from the original FS-NEAT in some ways such as speciation [10]. The Apache Thrift [1] framework organises communication between SharpNEAT and RinSim. All of our data and code is available on Figshare.\(^5\) Because of the need to evaluate millions of simulations that generally take a few seconds to evaluate, we use the grid computing framework JPPF [24] to distribute simulation tasks over 75 computers with a combined number of over 250 cores.

5.2 Evolutionary runs

We examined the performance of our approach for three scenario classes (Tab. 1). The characteristics of these classes are taken from [9], who report results for five scenarios per class.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Scenario class & Duration & Average request intensity (on average) & Fleet size \\
\hline
4h-24 & 4 hours & 24 requests per hour & 10 vehicles \\
4h-33 & 4 hours & 33 requests per hour & 10 vehicles \\
7.5h-24 & 7.5 hours & 24 requests per hour & 20 vehicles \\
\hline
\end{tabular}
\caption{Characteristics of the three investigated scenario classes}
\end{table}

We generated a training set of 1000 scenarios and a test set of 500 scenarios for each class. We then performed four evolutionary runs, each time using a population size of 600. Finally, we simulated each input- and champion heuristics on all scenarios in each of the test sets and on the 15 scenarios of [9].

In run one we always perform fitness evaluation using the same single scenario, also used by [9].\(^6\) This scenario is four hours long and has an average request intensity of 21 requests per hour (i.e. precisely 84 requests). In runs two

\(^5\) http://dx.doi.org/10.6084/m9.figshare.956301

\(^6\) This corresponds to the scenario with instance number one in Tab. 2 of [9].
to four we perform fitness evaluation using the training scenarios of class 4h-24, 4h-33 and 7.5h-24. We refer to these runs as run 4h-24, run 4h-33 and run 7.5h-24. We always perform three simulations (using three different scenarios) for each fitness evaluation.

We monitor the evolutionary process while it is running and stop evolution when there has been no improvement for a large number of generations. Run one to four took 2503, 600, 632 and 159 generations. Therefore, for example evolving the champion of class 4h-33 took 600 \times 632 \times 3 = 1137600 simulations. On our grid this took two hours and 55 minutes, giving an average simulation time of approximately 2.3 seconds per core. Scenarios in 7.5h-24 take 4.4 seconds on average: they are longer and require more work per time unit because of the larger fleet size. Scenarios in 4h-24 take only 1.3 seconds to simulate.

5.3 Results

We evaluated every primitive and champion heuristic on all test sets (Tab. 2). Perhaps surprisingly, Est always constructs better routes than all other primitive input heuristics including Dist.8 The evolved heuristics easily outperform the input heuristics. As expected, the best routes are created by the heuristic that was evolved using training scenarios of the same class (e.g. the champion of run 4h-24 is best for creating routes for scenarios in 4h-24). Looking at performance on test set 7.5h-24, the routes of Est are 47% more expensive than the routes constructed by the champion of run 7.5h-24. For run 4h-24, this increases to 89% and for run 4h-33 to 196%.

The overfitted champion heuristic constructs a route with a cost of 756, which is 63% more than the cost of 465 reported in [9] and 70 better than the routes created by the non-overfitted champion heuristic.

The behavior of NEAT in terms of fitness progression, genome growth and species sizes is very similar when overfitting and not overfitting. This shows that NEAT has no problem with nondeterministic fitness functions. The neural

---

7 In generation \(g\), we use scenarios numbered \(3(g-1) + 1 \mod 1000\) to \(3(g-1) + 3 \mod 1000\).
8 This cannot be deduced from Tab. 2, where we left out the results of all primitive heuristics except Dist and Est.
Table 3. Costs of routes by champion heuristics of run two to four compared to the costs reported in [9].

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>336&lt;sup&gt;a&lt;/sup&gt; 553 (159%)</td>
<td>473 736 (156%)</td>
<td>539 1021 (189%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>65&lt;sup&gt;b&lt;/sup&gt; 215 (330%)</td>
<td>4392 4596 (104%)</td>
<td>1 108 (108%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>55&lt;sup&gt;c&lt;/sup&gt; 78 (142%)</td>
<td>699 708 (101%)</td>
<td>0 34 (n.a.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>465&lt;sup&gt;d&lt;/sup&gt; 826 (177%)</td>
<td>5564 6039 (109%)</td>
<td>540 1162 (215%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>386 579 (150%)</td>
<td>402 684 (170%)</td>
<td>614 1125 (183%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>68 235 (346%)</td>
<td>867 990 (114%)</td>
<td>3 125 (4167%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>53 83 (157%)</td>
<td>297 272 (92%)</td>
<td>1 41 (n.a.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>506 896 (177%)</td>
<td>1566 1946 (124%)</td>
<td>618 1291 (209%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>352 601 (170%)</td>
<td>455 710 (156%)</td>
<td>629 1215 (193%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>139 240 (172%)</td>
<td>853 1417 (166%)</td>
<td>2 163 (8150%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>98 89 (90%)</td>
<td>303 352 (116%)</td>
<td>1 53 (5300%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>589 930 (158%)</td>
<td>1611 2479 (154%)</td>
<td>632 1431 (226%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>359 575 (160%)</td>
<td>434 770 (177%)</td>
<td>760 1324 (189%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>38 224 (589%)</td>
<td>1104 1459 (132%)</td>
<td>6 152 (2533%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31 76 (245%)</td>
<td>348 354 (102%)</td>
<td>5 62 (1240%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>428 875 (204%)</td>
<td>1886 2522 (134%)</td>
<td>711 1538 (216%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>348 529 (152%)</td>
<td>495 1049 (212%)</td>
<td>694 1255 (181%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75 225 (300%)</td>
<td>4121 4463 (108%)</td>
<td>0 152 (n.a.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>52 78 (150%)</td>
<td>687 680 (99%)</td>
<td>0 49 (n.a.)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>476 832 (175%)</td>
<td>5303 5913 (112%)</td>
<td>694 1455 (210%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>357 563 (158%)</td>
<td>452 722 (160%)</td>
<td>635 1188 (187%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>78 228 (292%)</td>
<td>2267 2585 (114%)</td>
<td>2 140 (7000%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>58 81 (140%)</td>
<td>467 473 (101%)</td>
<td>1 48 (4800%)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>493 872 (177%)</td>
<td>3186 3780 (119%)</td>
<td>638 1376 (216%)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<sup>a</sup> Travel time, <sup>b</sup> Lateness, <sup>c</sup> Overtime, <sup>d</sup> Total

nets of champions contain in between 20 and 100 connections. We observed that performance usually stops increasing after the number of connections reaches approximately 20. After that, genome sizes keep on increasing without gains in performance. Although we only report on four runs, our experience with other runs is that NEAT always finds heuristics of approximately the same quality.

Table 3 compares the costs of the routes created using our mechanism with the 15 costs reported in [9]. We use the champion heuristics of run two to four to construct routes. These heuristics have been trained on scenarios of the correct class, but were not trained on the specific scenarios used by [9]. The difference in cost is smallest for the scenarios in 4h-33, where our routes are only 19% more expensive. For scenarios in 7.5h-24, our routes are 116% more expensive than those of [9].
5.4 Analysis

We identify two trends in the results of the previous section. After arguing for these trends, we compare our approach to the one of [9]. This section concludes by identifying possible improvements. We want to stress that results are still preliminary, and that more experiments are required to support most of the points below.

Trends The evolved heuristics perform better when fleet sizes are smaller, both relative to the input heuristics and the routes constructed by [9]. This can indicate a need for better coordination between vehicles.

The evolved heuristics also perform better when request intensity is higher, again both relative to the input heuristics and relative to [9]. The latter is not surprising, since [9] uses an interruptible anytime algorithm that optimizes routes until new requests arrive. In contrast, our approach does not use the extra available computation time. The first can be an indication that there is more room to optimize when the number of open requests is larger. We hypothesize that this is mostly due to a lack of the ability to evolve cooperative behavior. For example, when many vehicles are idle and the list of open requests contains only a single request, the first vehicle to get the opportunity to claim this request will always do so; this is part of the hardwired agent behavior (Sect. 4.1) on which evolution has no influence. Such situations occur less frequently with higher request intensity per vehicle.

Comparison The comparison with [9] shows that our system cannot yet compete with the state-on-the-art on the given problem set. Improvements are necessary to make our system competitive; we already identified some possible issues above. However, there are also several points to be made in favor of our approach.

First we note that the performance of the champion of run 4h-24 is only 9% worse than the performance of the overfitted heuristic. This indicates that heuristics generalize well within their scenario class. Although our mechanism is very simple, we seem to have successfully evolved something akin to instinct for our agents: it allows them to choose one request out of hundreds in mere milliseconds in a reasonable way. We see it as a huge accomplishment that our system allows to shift much of the computational effort off-line, requiring very little online computation time. Also, that we currently require so little online computation time can indicate that there is still much room for improvement.

Another important point is that we compared the performance of our algorithm to that of [9] in a setting for which their algorithm is optimized. As an illustration, the artificial constraint that vehicles may not divert was part of the problem description and is crucial for [9]. Supporting diversion requires only minimal change of our algorithm and almost certainly improves performance. Also, although requests arrive during the day, most of them only have to be picked up and delivered by the end of the day (Fig. 1). This gives [9] plenty of time to optimize their routes without incurring a tardiness penalty. We hypothesize that our approach would perform relatively better in more ‘dynamic’ settings that require a fast response time.
Figure 1. Empirical cumulative distribution function (ECDF) of the announce time and the end of pickup and delivery time windows for a typical instance of 7.5h-24.\textsuperscript{10}

**Improvements** The most pressing improvement is to add some form of coordination to the mechanism. We expect this to lead to much better results, especially because all well-performing MASs in the PDP-domain that we know of coordinate [8, 21]. There are several ways to incorporate coordination in our evolved heuristic approach. For example we can try to combine it with an auction mechanism, or we could add information about other agent’s intentions into the set of inputs.

6 Conclusion

We investigated the feasibility of using NEAT to create a MAS for the dynamic PDPTW and compared our approach to a specialized tabu search heuristic by Gendreau [9]. Our NEAT and MAS based approach produces routes that are between 19% and 116% worse compared to the results reported by Gendreau. However, our approach uses only very little computation time and presumably performs relatively better in highly dynamic situations such as those encountered by emergency services. Results indicate that NEAT is working well.

Although our approach is very similar to that of [3, 19, 40], we are the first to compare our results with those of a state-of-the art algorithm. Unfortunately we therefore are also the first with a worse result, which is unexpected based on earlier work [3, 19, 40]. Based on these preliminary results we conclude that a thorough, fair and independent evaluation is necessary to gain insight in the performance of algorithms and MASs. In future work we plan to increase the number of evolutionary runs and to compare the performance of NEAT to that of GP.

Acknowledgements

This research is partially funded by the Research Fund KU Leuven.

\textsuperscript{10} The graph corresponds to scenario number one in 7.5h-24 used by [9].
References


