Evolutionary Synthesis of Multi-Agent Systems for Dynamic Dial-a-Ride Problems

Rinde R.S. van Lon
Dept. of Computer Science
KU Leuven
Celestijnenlaan 200A
3001 Leuven, Belgium
Rinde.vanLon
@cs.kuleuven.be

Tom Holvoet
Dept. of Computer Science
KU Leuven
Celestijnenlaan 200A
3001 Leuven, Belgium
Tom.Holvoet
@cs.kuleuven.be

Greet Vanden Berghe
CODeS research group
KaHo Sint-Lieven
Gebr. Desmetstraat 1
9000 Gent, Belgium
Greet.VandenBerghe
@kahosl.be

Tom Wenseleers
Department of Biology
KU Leuven
Naamsestraat 59
3000 Leuven, Belgium
Tom.Wenseleers
@bio.kuleuven.be

Juergen Branke
Warwick Business School
The University of Warwick
CV4 7AL Coventry
United Kingdom
Juergen.Branke
@wbs.ac.uk

ABSTRACT

In dynamic dial-a-ride problems a fleet of vehicles need to handle transportation requests within time. We research how to create a decentralized multi-agent system that can solve the dynamic dial-a-ride problem. Normally multi-agent systems are hand designed for each specific application. In this paper we research the applicability of genetic programming to automatically program a multi-agent system that solves dial-a-ride problems. We evaluated the evolved system by running a number of simulations and compared it's performance to a selection hyper-heuristic. The results shows that genetic programming can be a viable alternative to hand constructing multi-agent systems.

Categories and Subject Descriptors

I.2.2 [**Automatic Programming**]: Program Synthesis; I.2.11 [**Distributed Artificial Intelligence**]: Multi-agent systems

Keywords

Multi-agent systems, genetic programming, dial-a-ride problems, decentralized control

1. INTRODUCTION

The dial-a-ride problem (DARP) is a vehicle routing problem that concerns the transportation of goods or people between an origin and destination [1]. We consider the dynamic DARP in which requests arrive over time. Real world examples are taxi companies and courier services where it is common that requests need to be handled in a short period of time. As a provider of such a service one should try to deliver all goods within a short time window. Most work in DARP has focused on the static variant [1]. Static solutions are usually centralized and computationally intensive, especially if many re-computations need to be done over time.

There is a lot of attention for decentralized MASs [11, 3, 5, 6]. Designing a decentralized system that shows acceptable global performance is hard. Most of the difficulty lies in the design of a decision mechanism that uses local and incomplete information. In this paper we address this problem by using genetic programming to create this decision mechanism.

The purpose of this paper is to investigate the applicability of genetic programming for developing decentralized MASs that solve dynamic DARPs. In this first step we will neglect the possibilities of explicit coordination within the MAS. We evaluate our evolved heuristics on a set of DARP scenarios and compare its performance with two solutions. One is a centralized static solution that has the benefit of foresight, it knows everything that is going to happen beforehand, this solution serves as a lower bound. The second solution we compare with is a centralized rolling-time horizon solution, at every time step this solution tries to find the best global solution given the information currently available.

We give a formal definition of the dynamic DARP and a standard MAS solution in section 2. A description of how GP is used to create a MAS to solve DARP is given in section 3. In section 4 related work is discussed followed by an evaluation of our approach in section 5. We conclude that GP is a feasible technique for MAS to solve DARP in section 6.

2. DARP AND MAS

2.1 Formal definition

In the DARP there are n vehicles which have to handle m requests. In the dynamic variant no information is known about the size and distribution of m, requests arrive gradually over time. A request consists of a pickup location, a drop-off location and a deadline. A request is handled when customers have been transported from pickup to delivery location. Further we assume that:

- Vehicles drive with a constant speed (infinite fuel, no driver fatigue).
- Vehicles can only handle one request at a time (vehicles have unit capacity).
- When a vehicle is carrying customers it is not allowed to divert.
- The actual pickup and delivery of customers takes no time.

The objective function for this problem is to minimize the tardiness of all requests. Where tardiness is defined as in equation 1.

$$\sum_{r=1}^{|requests|} \max(0, deliverytime(r) - deadline(r)) \quad (1)$$

Here $deliverytime(r)$ is defined as the time that request r is delivered, $deadline(r)$ indicates the deadline for request r. An implication of this equation is that a solution is considered invalid if it has not handled all requests.

2.2 MAS solution

The distributed nature of the DARP allows for a logical mapping of agents: each vehicle is represented by an agent. When a vehicle is not already carrying customers it has to decide which request it is going to handle first. A vehicle can make this decision based on information it has about available requests, such as location and time to deadline. Creating a decision mechanism for an individual agent is non trivial as it needs to take into account the global objective as defined in equation 1. As a decision mechanism one can devise a heuristic that computes a priority value for each request. Then, the agent simply handles the request with the highest priority value first. For creating this heuristic we present a genetic programming solution in the next section.

3. GP OF MAS FOR DYNAMIC DARPS

In this section we present our GP solution for evolving agent heuristics that solve the dynamic DARP.

3.1 Heuristic

Similar to [2] we define an agent heuristic as a function which assigns a priority value to a request:

```
heuristic = (agent, request) -> priority
```

This heuristic is deployed on every agent. For all available requests the agent executes the heuristic. The agent always decides to handle the request with the highest priority value first. The priority values for all available requests are recomputed periodically, which means that agents can divert from their paths. To avoid infinite recurring diversions, agents are disallowed to divert back to a previous target.

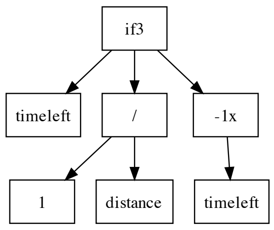
Since every agent is autonomous and has no means to communicate with other agents, conflicts can arise. A conflict can occur when two agents aim to handle the same request. The agent that arrives earliest will handle the request, the second agent will divert to another target as soon as it recognizes that its first target is no longer available.

Note that by using the above described heuristic the agents are only scheduling one request in advance. This is different from more traditional approaches which usually plan multiple requests in advance. Since we consider dynamic DARPs long term planning is not beneficial because of the rapidly changing dynamics. Also, because each agent is autonomous, it is possible that vehicles aim for the same request. This is implemented such that the agent that first reaches a request will handle it.

A simple example heuristic is shown in Figure 1(a). This defines a priority function that returns a value inversely proportional to the distance in case the request falls within the time window, otherwise it returns a value proportional to the lateness. This function prioritizes late requests if available, otherwise it prioritizes requests which are close.

```
if( timeleft > 0 ){
  return 1 / distance
} else {
  return -1 x timeleft
}
```

(a) pseudocode



(b) tree representation

Figure 1: An example heuristic shown as pseudocode (a) and as a GP tree (b).

3.2 GP of heuristics for agents

The main difference between genetic algorithms (GA) and GP is the choice of representation [4]. In GAs individuals commonly represent numerical values while in GP they represent a program. Our approach differs from a regular GP approach in that it requires a simulation to compute the fitness of an individual.

For creating the above described heuristic we use GP, individuals are represented as a syntax tree. Figure 1(b) shows the same heuristic as described previously but now as a syntax tree. The nodes in the tree are functions and the leafs are terminals (constants or variables). Table 1 gives an overview of the functions and terminals that we use.

3.3 Simulation-based fitness evaluation

Since the quality of a heuristic can not analytically be deduced, we are using a simulation-based fitness evaluation. This means that for every individual in the evolutionary run we need to perform a simulation that tests the heuristic on a problem instance. For simulating the dynamic DARP we use a multi-agent discrete event simulator called RinSim[1]. In this simulator vehicles can drive over roads and pickup

[1]https://github.com/rinde/RinSim

Table 1: GP functions and terminals, the first three terminals return a value which is relative to the current request c.

Name	n args	Description
distance	0	Bird flight distance to c measured from vehicle
timeleft	0	Time-left to handle c, negative when late
nearby-packages	0	Returns the accumulated distance of the five closest requests measured from the drop-off location of c.
$\times, /, +, -, max$	2	Mathematical operators
$-1\times$	1	Negates the argument
if3	3	if($arg0 \geq 0$) then $arg1$ else $arg2$
$0, 1, 2, 4, 10$	0	Constants

and deliver customers. The road map is a connected directed graph, only the nodes of the graph can serve as pickup and delivery locations. The agents in the simulator receive updates periodically, in our case the agents receive an update every second (simulation time).

4. RELATED WORK

There are two main research tracks which are directly related to our work. Firstly, the field of agent-based modeling applied to DARPs. And secondly, the field of GP aimed at developing decentralized control systems.

4.1 MAS and DARP

Problems such as DARP have distributed characteristics which allow natural modeling of the problem using MASs. For example, in [5] a MAS is used to solve the planning problem for a taxi service company in the city of London. Although this solution is a MAS it is not entirely decentralized since it contains a centralized dispatcher which assigns tasks to vehicles. A similar system is implemented in [3]. In [11] agent based modeling was used for an underground freight transportation system. It was shown that their MAS solution was more robust to demand fluctuations compared to several traditional optimization methods.

The described methods above use MAS in either of two ways: (a) they introduce a dispatcher agent which assigns requests to agents or (b) they let agents autonomously decide which request to handle next. In case of (a) the dispatcher agent usually applies a problem specific heuristic or an optimization method for assigning requests to agents. When using (b) the autonomous agents are given a hand constructed heuristic and conflict resolution mechanism based on specific domain knowledge.

Our approach differs from the described MAS literature in that it implements a decision mechanism in a completely decentralized way. Also, instead of using hand constructed decision mechanisms we have developed the agent decision mechanisms using genetic programming.

4.2 GP for MAS and decentralized control

Genetic programming (GP) is a well known general problem solver which is capable of finding creative solutions [8]. Past research shows that GP is capable of producing collaborative behaviors in MASs. For example, in [10] a virtual world was defined containing lions and gazelles. GP was used to program the lion controllers such that they collaboratively hunt for the gazelles in an efficient manner. In [14] it was shown that GP can be used to evolve agents that only use communication to find each other in a virtual world. An application of GP in a more complex environment is presented in [9]. Here the controllers for a team of software soccer robots were evolved, the resulting team was able to coordinate basic maneuvers such as scattering itself over the field and defending the goal. More recently in [15] cooperative strategies for unmanned aerial vehicles were evolved using GP. In [7] neural networks were evolved for swarming micro air vehicles.

The idea of using GP for developing a heuristic which can be used to coordinate a decentralized control system has been researched before. In [13, 2] it was shown that by applying GP new heuristics can be obtained for the dynamic job shop scheduling problem which outperform state-of-the-art heuristics.

To our knowledge we are the first to combine MAS and GP in the area of dynamic DARPs.

5. EVALUATION

Our approach is evolved on a set of scenarios on a real world map. For testing we use a hold-out set of scenarios on the same map. We compare our solution with the performance of a selection hyper-heuristic on the hold-out set.

5.1 Scenarios

In typical optimization approaches for DARP diversion is not considered. As such, all major benchmarks do not contain information about the road structure. We consider diversion to be an integral part of the problem, as such we decided to create our own set of scenarios which are situated on actual maps.

A scenario is a sequence of requests that arrive during a period of time. Our scenario creation method is very similar to that of [16, 17]. Instead of a Euclidean plane we use actual map data. We run our simulations on a simplified map of the city of Leuven (Belgium) which was obtained from OpenStreetMap[2]. In Table 2 the parameters are shown that we have used for the creation of our scenarios. We made the scenarios and map available online, more information can be found on the accompanying webpage[3].

Table 2: Scenario creation parameters

Parameter name	Parameter value
number of vehicles	6
number of requests	200
request deadline	30 minutes
incoming requests time period	2 hours
map	Leuven, Belgium

[2]`http://www.openstreetmap.org/`

[3]`http://people.cs.kuleuven.be/rinde.vanlon/data/GECCO2012/`

5.2 Comparison: selection hyper-heuristic

Since our scenarios are relatively large (in terms of requests) exact optimization solutions are not feasible. Therefore as a comparison method we opted for a particular selection hyper-heuristic that has shown good performance on a set of problem domains [12]. The pseudocode for this method is:

```
current best solution = random

while(termination criterium not met){
  current heuristic = randomly select heuristic from
                                set of heuristics H
  new solution = use heuristic for changing the current
                                best solution
  if( new solution >= current best solution){
    current best solution = new solution
  }
}
```

In the pseudocode above, a `solution` is an assignment of requests to vehicles in a particular order. The set of heuristics `H` is a set of methods that 'mutate' a solution, the used heuristics are shown in Table 3. Each heuristic guarantees that it always produces complete solutions, i.e. solutions where all known requests are handled by a vehicle.

Table 3: The set of heuristics `H` used in the selection hyper-heuristic.

Name	Description
precedence change	pick a vehicle, pick one of its requests and change its precedence
precedence shuffle	randomly pick a vehicle and shuffle its requests
move	pick a vehicle, pick one of its requests and move it to another vehicle
swap all	pick two vehicles and exchange all requests
swap one	pick one request of one vehicle and move it to another vehicle

We have used this algorithm in a static (`hh-s`) and a dynamic (`hh-d`) way. Both variants are executed at a centralized entity which coordinates all vehicles. In the static variant the entire scenario is known beforehand, a global plan for all vehicles is computed. This is of course an unfair comparison, as such we use this value as a lower bound. In the dynamic variant this algorithm uses a rolling-time horizon, meaning that it optimizes the solution based on the information available at that moment in time. Note that this algorithm is executed every time new information comes available. The algorithm uses the current vehicles positions as input for optimizing the solution. The parameters we used for executing the algorithm are shown in Table 4, the results are shown in Table 5.

Table 4: Experiment settings for the selection hyper-heuristic.

Parameter	`hh-s`	`hh-d`
iterations	1000000	250000
computation duration (for each scenario)	30-34 hours	9-12 days

Table 5: Results for the two variants of the selection hyper-heuristic on the ten hold-out scenarios. The first two columns show tardiness (in hours) and the last column shows the ratio between the dynamic and static variant.

Scenario	`hh-s` (hours)	`hh-d` (hours)	`hh-d/hh-s` (ratio)
0	0.69	23.888	34.63
1	4.339	26.565	6.12
2	0.85	26.306	30.96
3	3.283	28.183	8.58
4	0.415	23.42	56.43
5	3.165	27.173	8.59
6	1.267	26.763	21.12
7	5.672	38.138	6.72
8	7.087	43.206	6.1
9	12.576	46.884	3.73

5.3 Comparison: hand constructed heuristics

For comparison, we constructed two trivial heuristics. One is called `distance` and is defined as `-1 distance`. When using this heuristic vehicles will always go to the closest request first. The second heuristic is called `timeleft` and is defined as `-1 timeleft`. When applying this heuristic on a vehicle the most urgent requests are handled first.

5.4 GP evaluation settings

For our GP implementation we used the ECJ library[4]. We have extended the default GP implementation in two ways. Firstly, we added a simulation-based fitness function that performs simulations for an individual. Secondly, since the simulations are computationally intensive we hooked ECJ up with a framework that distributes the simulations over a large number of computers such that many simulations can be executed in parallel.

The fitness value of an individual is computed by taking the summed tardiness of all requests in a simulation as was defined in equation 1. Since the tardiness needs to be minimized, lower fitness values indicate better performance. When either:

- the simulation time has exceeded a predefined limit;
- the simulation *computation* time has exceeded a predefined limit;
- or, not all requests were handled,

the fitness value takes an infinitely high value instead. These constraints imply that there is implicit fitness pressure for large (computational intensive) individuals. The computation time constraint has a stochastic nature since it depends on the hardware used for execution and the load of the hardware at that time.

In each generation we compute the fitness of our individuals by doing simulations for multiple scenarios. To compute the fitness of an individual we take the average fitness for all scenarios, unless the fitness for one of the scenarios is infinite, then the individual's fitness is also infinite. Each generation we use different random generated scenarios (using the same settings as explained in section 5.1). In the

[4]`http://cs.gmu.edu/~eclab/projects/ecj/`

final generation of the evolutionary run we do a more extensive fitness calculation by using a larger number of scenarios. From this final generation we select the individual with the best fitness value and then test it on a set of 10 hold-out scenarios.

We have done experiments with two different parameter settings, shown in Table 6. For computing these results we used in total 80 computers. During computation the computers could theoretically also be used by other users, however, since the computations took place during nights and weekends this happened rarely. The total computation time for `gp 51` was 10 hours, for `gp 101` it took 44 hours. In Appendix A the resulting heuristic of `gp 51` is shown.

Table 6: Genetic programming settings

name	`gp 51`	`gp 101`
generations	51	101
population size	1500	
# scenarios	3	5
# scenarios at last generation	10	25
simulation time limit	24 hours	
simulation computation time limit	120 seconds	
tournament selection size	7	

5.5 Results

Table 7 shows the results. It can be seen that for every scenario the heuristic obtained through genetic programming outperforms the solution obtained by the dynamic variant of the hyper-heuristic. As expected, the evolved solutions never outperformed the lower bound generated by the static selection hyper-heuristic. An interesting and unexpected result is that `gp 51` performs slightly better than `gp 101` even though `gp 101` used about four times as many fitness evaluations. This can be explained by the fact that both GP runs (`gp 51` and `gp 101`) were run only once because of the high computational cost. However, when examining the results per scenario, it is clear that `gp 51` does not always outperform `gp 101`. Further, it can be seen that the `distance` heuristic performs much better than the `timeleft` heuristic. However, the evolved heuristics are much better than the `distance` and `timeleft` heuristic.

Table 7: The results for the evolved and hand constructed heuristics on the set of 10 hold-out scenarios shown as ratios compared to the best known static solution as was displayed in Table 5. The results shown in bold are the best results found for that scenario.

Scenario	`hh-d`	`gp 51`	`gp 101`	`distance`	`timeleft`
0	34.63	**5.72**	8.95	19.44	597.23
1	6.12	**1.46**	2.19	3.11	98.85
2	30.96	7.73	**5.04**	15.59	516.25
3	8.58	3.01	**1.75**	4.99	147.65
4	56.43	**12.23**	14.81	28.48	1181.3
5	8.59	**1.67**	1.92	4.27	149.84
6	21.12	5.02	**4.56**	8.93	382.2
7	6.72	**1.57**	1.81	2.7	85.46
8	6.1	**2.15**	2.24	3.3	73.16
9	3.73	**1.23**	1.43	1.66	46.5
mean	17.8	3.8	3.9	8.7	327.4

6. CONCLUSION

We have presented a method to automatically generate a multi-agent system that can solve the dial-a-ride problem for a specific class of scenarios. The resulting MAS is decentralized, meaning that each agent makes its decisions autonomously based on its situation. We have used GP to generate a heuristic which guides each agent at making decisions. Our results show that our generated MAS is effective at solving the dial-a-ride problem when compared to a centralized rolling-time horizon solution. These results indicate that genetic programming is a good method for programming decentralized multi-agent systems.

Based on this first step, various directions for future work are possible. An interesting approach is to extend the GP functions and terminals to allow agents to communicate with each other. Adding communication, agents can coordinate their actions to avoid conflicts when aiming for the same request. We expect coordination to improve the solution even further.

The current MAS is homogenous, every vehicle has the same characteristics and every agent uses the same heuristic. It would be interesting to investigate whether a heterogenous MAS performs better than a homogenous MAS. In a heterogenous MAS agents can specialize, e.g. some agents can be optimized for a certain region or a specific kind of vehicle.

Acknowledgements

We would like to thank Mustafa Misir of CODeS research group, KaHo Sint-Lieven, Belgium for his help with implementing the selection hyper-heuristic. This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven.

7. REFERENCES

[1] G. Berbeglia, J.-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, Apr. 2010.

[2] J. Branke and C. W. Pickardt. Evolutionary search for difficult problem instances to support the design of job shop dispatching rules. *European Journal of Operational Research*, 212(1):22–32, July 2011.

[3] K. Dorer and M. Calisti. An adaptive solution to dynamic transport optimization. *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems - AAMAS '05*, 2005.

[4] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer-Verlag Berlin Heidelberg, corrected edition, 2007.

[5] A. Glaschenko, A. Ivaschenko, G. Rzevski, and P. Skobelev. Multi-agent real time scheduling system for taxi companies. In *Proc. of 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, pages 29–36, 2009.

[6] S. Hanif, R. R. S. van Lon, N. Gui, and T. Holvoet. Delegate MAS for large scale and dynamic PDP: A

case study. In *Intelligent Distributed Computing V*, volume 5, pages 23–33, Delft, 2011.

[7] S. Hauert, J.-C. Zufferey, and D. Floreano. Evolved swarming without positioning information: an application in aerial communication relay. *Autonomous Robots*, 26:21–32, 2009.

[8] J. R. Koza. *Genetic programming II*. MIT Press, Cambridge, MA, Mar. 1994.

[9] S. Luke. Genetic programming produced competitive soccer softbot teams for robocup97. *Genetic Programming*, pages 214–222, 1998.

[10] S. Luke and L. Spector. Evolving teamwork and coordination with genetic programming. In *Proceedings of the first annual conference on genetic programming*, pages 150–156. MIT Press, 1996.

[11] M. Mes, M. van der Heijden, and A. van Harten. Comparison of agent-based scheduling to look-ahead heuristics for real-time transportation problems. *European Journal of Operational Research*, 181(1):59–75, Aug. 2007.

[12] M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. Design and Analysis of an Evolutionary Selection Hyper-heuristic Framework. Technical report, 2012.

[13] C. Pickardt, J. Branke, T. Hildebrandt, J. Heger, and B. Scholz-Reiter. Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In *Proceedings of the 2010 Winter Simulation Conference*, 2010.

[14] A. Qureshi. Evolving Agents. In *GECCO '96 Proceedings of the First Annual Conference on Genetic Programming*, pages 345–352, Cambridge, MA, USA, 1996. MIT Press.

[15] M. D. Richards, D. Whitley, J. R. Beveridge, T. Mytkowicz, D. Nguyen, and D. Rome. Evolving cooperative strategies for UAV teams. *Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05*, page 1721, 2005.

[16] S. Ropke, J.-F. Cordeau, and G. Laporte. Models and Branch-and-Cut Algorithms for Pickup and Delivery Problems with Time Windows. *Networks*, 49(4):258–277, 2007.

[17] M. Savelsbergh and M. Sol. DRIVE: Dynamic routing of independent vehicles. *Operations Research*, pages 474–490, 1998.

APPENDIX

A. VISUALIZATION OF GP 51