Final Solution - Multi-agent Systems

Jasper Linmans [10249060], Jasper Driessens [11349026], Diede Rusticus [10909486]

February 2017

1 How to run the code

Make sure the files `action.nls`, `agents.nls`, `auction.nls`, `init-buses.nls`. `messaging.nls`, `procedures.nls`, `protocols.nls` and `read-dijkstra.nls` are placed in the same folder as `Amsterdam.nlogo`. Now run the NetLogo simulation as normal.

To see the protocols, messaging and other system elements in action, go to `init-buses.nls` and set the various `debug` variables to `true`. This will make the agents log their actions and reasoning to NetLogo's command center.

2 Introduction

This report is structured as follows. First, we motivate the general design. We break down the problem at hand and characterize the environment. With respect to this context, we then discuss the suitability of several paradigms for agent and society design, and specify the high-level choices we eventually made. In the process, we also touch on some of the architectural considerations.

In the rest of the report, we zoom in on the various system elements. For each of those, we reiterate the goal of that part of the system, motivate the more detailed design choices, and briefly make the connection with the NetLogo implementation. We conclude by showing a quick summary of the performance results obtained with the final system.

We do not have a separate solution for the performance contest.

3 General design

3.1 Problem and environment breakdown

In order to make good design choices, we need to first become aware of the problem and environment characteristics. This allows us to formulate the main goals and considerations, together forming a measure against which we can evaluate the appropriateness of the various mechanisms, paradigms and concepts we need to choose between.

The goal of the system is to transport passengers, using buses as agents, within a network of roads and bus stops, in a run of 24 hours. The main objective is to minimize passengers' average travel times . Secondary objectives are to minimize costs and communications. The network is fixed throughout runs, but passengers' (desired) journeys can start and end at any of the stops. The locations and amounts of appearing passengers are non-deterministic, but follow a common distribution throughout the day.

Agents can pick up and drop off passengers when at a stop, drive from stop to stop along the network roads, message other agents, and buy new agents. The actions with which costs are associated are driving and buying. Agents are almost heterogeneous in their capabilities: they differ only in their capacity.

From an agent point of view, the environment contains three types of information with varying degrees of accessibility. All in all, we classify the environment as mostly *accessible* (Russell and Norvig (1995); Wooldridge (2009), Section 2). The three types of information are as follows.

Network information The stops and how they are connected with roads. Instantly, globally and completely available.

Passenger information The locations and destinations of the passengers. Instantly, globally and completely available for *waiting passengers* (passengers currently at a stop). Passenger information for *traveling passengers* (passengers currently in a bus), is only available to the agent holding the passengers.

Agent information The existence, locations and states of other agents. Agent information is hidden - agents only know their own existence, location and state.

3.2 Design considerations and decisions

With the preceding analysis in mind, we discuss our design considerations. We follow the basic principle of preferring simplicity. Complexity, while enabling more sophisticated solutions, also makes a system hard to engineer, and thus should only be added when necessary. If relevant information is globally available, there is no need for a belief system; if tasks are heterogeneous, there is no need for intention prioritization on the basis of varying desires; if a fixed routine suffices, there is no need for autonomous means-ends reasoning; et cetera (Wooldridge, 2009, Section 4).

We distinguish two concerns agents are involved in. Collectively, agents are both *workers* and *managers*. The *worker concern* is operational: navigating through the network, deciding when and where to pick up and drop off passengers. The *manager concern* is tactical: monitoring (local) network busyness and agent fleet capacity, deciding when to buy, coordinating.

Our design strategy is to solve both concerns separately with a *worker module* and a *manager module*. The worker module is reactive and has a subsumption architecture. While it does its job, it provides little adaptability. We compensate for this with the manager module, which agents use to communicate, aggregate beliefs about busyness and fleet capacity, and make collective coordination decisions. By lifting the complexity to the manager level, we can get by with individually simple agents, because the intelligence emerges from the collective. This makes for a much more manageable engineering effort than chaotically having tens of agents autonomously planning in reaction to hundreds of passengers appearing across the network.

Because of their limitations, reactive subsumption architectures are not suitable for all contexts. Specifically, they require a context in which the local environment contains enough information, decision-making with a short-term view is sufficient, and the number of required behaviors can be kept small (Wooldridge, 2009, Section 5.1.5). As the behaviors are implemented as rules of the form 'situation $\rightarrow$ action', we need *accessible information that directly translates to actionable cues*.

By default, our environment is not such a context. Agent information is inaccessible, so other agents can not be reacted upon for coordination. Passenger information is accessible, but yields no actionable cues in 'raw form': as the world is open and agents receive passenger information from across the whole network, sophisticated reasoning would still be required to convert this information into navigational decisions. Inspired by the radioactive breadcrumbs of Steels (1990) however, we can augment our context to meet our requirements.

The idea is to constrain navigation and localize the relevance of information by introducing *social laws*, particularly *traffic laws* (Shoham and Tennenholtz (1995); Wooldridge (2009), Section 8.6.4). We manually partition the network into contiguous sets of stops, called *districts*. Each district has a fixed *route* that connects all district stops. Each agent is assigned to a district and simply follows its district's route, picking up and dropping off passengers along the way. Because of these social laws, agents can

reactively base their actions solely on the passengers waiting at the local bus stop (and, particularly, do not need to be aware of other agents to function properly). With this, the worker concern is now covered: a passenger can make any journey within a limited amount of time, given the agent fleet has enough capacity.

The only unresolved issue is that of *load balancing*: different parts of the network will be busy in different parts of the day, and the agent fleet capacity needs to adapt accordingly. This is the responsibility of the manager module. We again leverage the locality of the districts and have agents use their 'personal experience' to continuously maintain a *belief of district busyness*. The manager module engages in conversations with peers to make group decisions on buying and re-allocating buses. These decision are based on the beliefs of district busyness, aggregated via protocols, voting procedures and actions.

The justification for using a belief system is that, while passenger information is globally available, it does not directly translate to actionable 'busyness information'. 'Knowing where passengers are' does not equal 'knowing where capacity problems are'. The 'raw' information thus not suffices, and beliefs based on local experience must be used to arrive at intelligent decision-making.

4 Worker module

The worker module has a subsumption architecture of four reactive behaviors, ordered in the following subsumption hierarchy (Wooldridge, 2009, Section 5.1.1):

$$\text{Panic} \prec \text{Unload} \prec \text{Load} \prec \text{Drive}.$$

The behaviors do the following.

Panic If I have a capacity problem and have not performed PANIC yet, perform PANIC. This is an internal signal to my manager module, which then (in parallel to my worker module) starts to work on a solution. A capacity problem is when my bus' free capacity has dropped below some threshold.

Unload If I am at a stop, and a contained passenger wants to exit, perform DROP OFF action on that passenger. A contained passenger wants to exit either if their destination is the current stop, or if the current stop is a junction and the passenger needs to transfer. The latter is the fact if the other district comprises a stop that is further along the shortest path than any of the stops of my own district. Here, 'shortest path' means Dijkstra's shortest path from the current stop to the passenger's destination stop.

Load If I am at a stop, and one of the waiting passengers wants to enter, perform PICK UP on that passenger. A passenger wants to enter either if I am not at a junction, or if I am at a junction but the passenger needs to travel via my district. The latter is the fact if my district comprises a stop that is further along the shortest path than any of the stops of the junction's other district.

Drive Perform DRIVE TO on the next stop. This is usually the next stop along my district's fixed route, or, if I am currently not in my district, the next stop along the route towards my district (determined with Dijkstra). If I am not carrying any passengers and there are no waiting passengers on any of my district's stops, perform NULL instead.

A visualization of the subsumption architecture can be found in Figure 1. The hierarchy follows an ordering like Maslow's hierarchy of needs (Maslow, 1943): if we have capacity problems, we must resolve those first; if passengers need to get out, we drop them off to make room for new ones; if passengers need to get in, we need to do that first before driving further; if none of the previous things applies, we can safely move on.

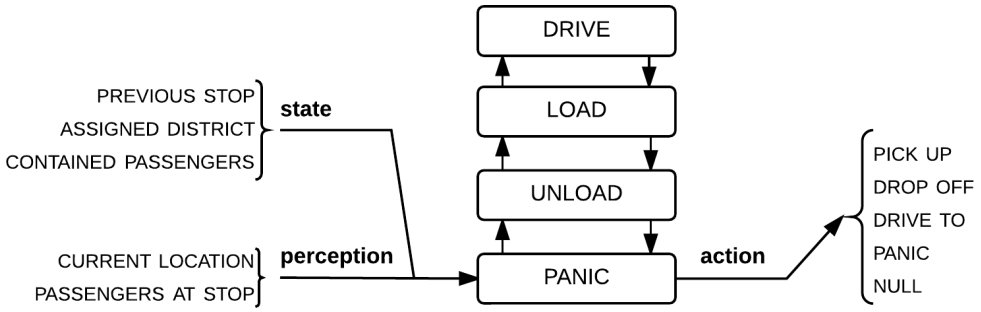

Figure 1: The subsumption architecture of the worker module. Note that Wooldridge uses this visual style to depict hybrid, layered architectures. In our case, it really is just a subsumption architecture, but we consider this style to better accentuate the hierarchical nature.

5 Manager module

As stated before: because of the district approach, the only non-trivial task left is that of load balancing. The challenge here is to, collectively, make a good judgement of local busyness and then respond in a way that is the most efficient with regards to resources. In the end, the possible solutions come down to buying a new bus (BUY SMALL, BUY MEDIUM, BUY LARGE), re-allocating an existing bus to a different district (RE-ALLOCATE) or, of course, doing nothing (NULL). It is the manager module's responsibility to make the right decisions on when to apply what solution. The module can act in three modes: deliberately (when the agent has a capacity problem), randomly (when there is no direct capacity problem, but an effort can be made to prevent that from happening) and reactively (when other agents signal they have a capacity problem). For a schematic visualization of the manager module, see Figure 2.

In this section, we motivate those processes on a high level. Formally, the conversations follow the *Panic* and *Auction* communication protocols, which are explicated fully in respectively Section 6 and Section 7.

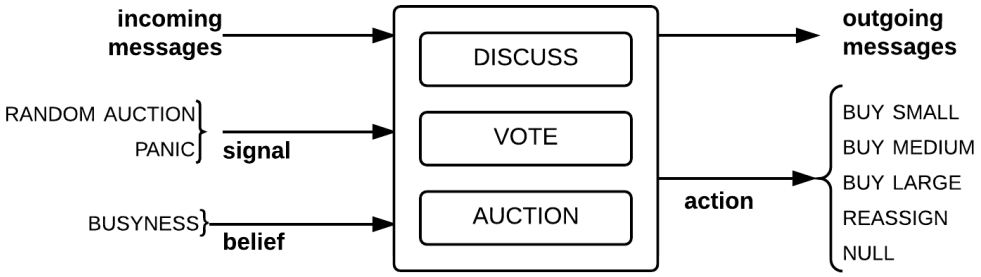

Figure 2: A schematic visualization of the manager module.

5.1 Deliberate mode

The deliberate mode starts when the worker module performs a PANIC, signaling it has a capacity problem. The idea is that this is a *belief of district busyness*: a local notion which, by itself, is not enough information to act upon. The agent therefore engages in a conversation with peers to find out what their beliefs of busyness are, aggregate those beliefs, and act accordingly. The first thing to find out is: does my district truly have a capacity problem? The agent inquires his district peers, and they tell him (based on their beliefs) whether they agree. If they don't, the agent assumes his capacity problem is temporary and ignores the PANIC signal. If they do, he inquires agents from other districts on their busyness. They, again, reply based on their beliefs, and the deliberate agent aggregates these replies to determine if there is a district with overcapacity. If this is the case, he performs a RE-ASSIGN on an agent from that district

to his own district. If this is not the case, he performs a BUY. Before buying, however, he casts a global vote to decide on the capacity of the new bus. The rationale of having all agents co-decide on the capacity (instead of just the district peers) is that, since the new bus will likely be re-assigned at some time, the decision will end up affecting all districts.

5.2 Random mode

An ounce of prevention is worth a pound of cure, so in addition to the panic-induced deliberate mode, the manager module also engages in a random mode. The idea here is that it is useful to, every now and then, redistribute bus capacity among the districts according to how much they need it. A district might have bought a lot of buses at one moment but may be relatively quite at some later time; it would then be a waste to re-allocate this overcapacity only after other districts have gotten into trouble. For this reason, a bus has a random probability to spontaneously auction itself. All districts then place bids, the height of which are based on the beliefs of that district's agents. This ensures that the bus will end up in the district that needs it the most.

6 Panic protocol

While Section 5 has already motivated the idea behind the collective decision-making performed through the Panic communication protocol, this section will go into the details of how the protocol works.

Earlier, we mentioned a belief of district busyness but kept vague about what this encompassed exactly. The current Panic protocol implements this belief in the simple form of the FULLNESS state variable, which is defined as

$$\text{FULLNESS} = \frac{\text{amount of contained passengers}}{\text{bus capacity}}.$$

The belief could have a more sophisticated form in a future version of the system. For instance, instead of simply checking its fullness, an agent may keep a short history of how often it had to decline passengers from entering. Currently however, this simple implementation suffices.

We walk now through the protocol, state by state, through the eye of an initiating agent. We first consider a normal run, after which we discuss some edge cases. The protocol's full state diagram is visualized in Figure 3.

PANIC:UNINITIATED This is the protocol state an agent is in when it is currently not engaged in the protocol. Every time step, the agent checks to see if FULLNESS $\geq$ PANIC THRESHOLD evaluates to true, where PANIC THRESHOLD is set to 0.8 in the current implementation. If this is the case, the protocol is initiated. Conceptually, this amounts to the worker module performing the PANIC action. Under the hood, what happens is that the message PANIC.WANT-TO-BUY is broadcasted to all other agents, including the agent's district as payload, and the state is set to PANIC:AWAITING-RESPONSE.

PANIC:AWAITING-RESPONSE This state is simply there for the agent to wait one time step for the replies to arrive. He sets the state to PANIC:ABOUT-TO-BUY. Meanwhile, other agents have received the PANIC.WANT-TO-BUY. They, too, check if FULLNESS $\geq$ PANIC THRESHOLD. If it is false, they reply with PANIC.DO-NOT-BUY, including their district and FULLNESS as payload.

PANIC:ABOUT-TO-BUY It is now time to aggregate the responses. There are three possible scenarios.

First, there might be a PANIC.DO-NOT-BUY from a peer within the district. This is interpreted as a rebuttal of the agent's idea that the district has a capacity problem, so the agent quits the protocol and goes back to PANIC:UNINITIATED (Wooldridge, 2009, Section 16.3).

Second, there might be one or more PANIC.DO-NOT-BUY from outside the district. If this is the case, the agent takes the cumulative FULLNESS per district by summing the reply payloads he received

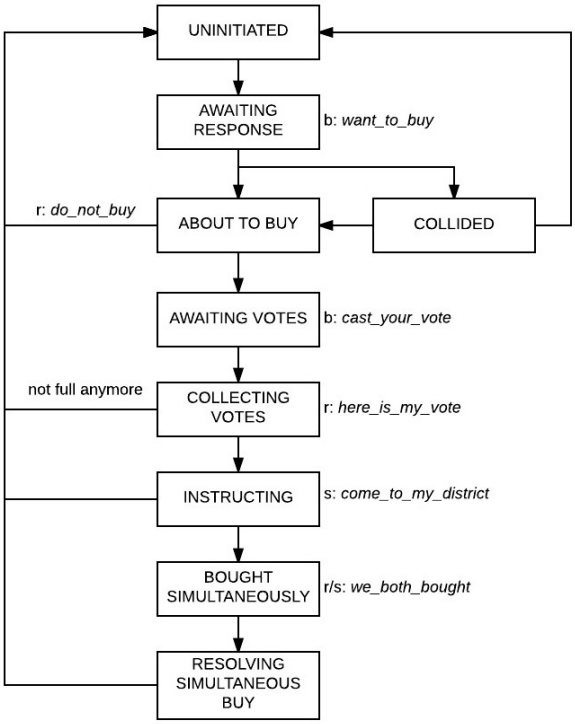

Figure 3: State diagram of the Panic protocol. b is a broadcast message, s is a private message and r is a received message.

from that district, and declares the district with the lowest cumulative FULLNESS the quietest. Of this district, he then randomly re-assigns an agent to his own district.

Third, there may be no replies at all. In this case, the agent is reinforced in his belief that there is a capacity problem, and proceeds to buy a new bus. In our example, let us assume that this is indeed the case. The agent then holds a general election to decide the size of the new bus. He broadcasts PANIC.CAST-YOUR-VOTE and proceeds to protocol state PANIC:AWAITING-VOTES.

PANIC:AWAITING-VOTES Again, this state is simply there for the agent to wait one time step. He sets the state to PANIC:COLLECTING-VOTES. Meanwhile, all other agents have received the inquiry and reply with PANIC.HERE-IS-MY-VOTE, including their preference orderings as payloads. They vote according to their FULLNESS, as follows:

$$\begin{aligned} 0 < \text{FULLNESS} \leq 0.25 &\rightarrow \text{large} \prec \text{medium} \prec \text{small}, \\ 0.25 < \text{FULLNESS} \leq 0.50 &\rightarrow \text{large} \prec \text{small} \prec \text{medium}, \\ 0.50 < \text{FULLNESS} \leq 0.75 &\rightarrow \text{small} \prec \text{large} \prec \text{medium}, \\ 0.75 < \text{FULLNESS} \leq 1 &\rightarrow \text{small} \prec \text{medium} \prec \text{large}. \end{aligned}$$

PANIC:COLLECTING-VOTES In this state, the agent aggregates the votes using the social choice function (detailed in Section 6.1). This function outputs the preferred capacity. Right before buying, the agent performs a last check on FULLNESS $\geq$ PANIC THRESHOLD - maybe his situation has changed in the meantime, defeating the need for a new bus. Otherwise, he temporarily stores the ID of the youngest bus he knows, buys the new bus, and goes to state PANIC:INSTRUCTING.

PANIC:INSTRUCTING Here, the agent checks whether the youngest agent he knows of via the Discovery protocol has a different ID than he stored during the previous state. If this is the case, then this must be his newly bought bus. He sends him a PANIC.COME-TO-MY-DISTRICT message with his

district as payload, and finally quits the protocol. The new agent now knows to which district to go, and starts his work.

As multiple agents can concurrently run the protocol, there are two scenarios in which a conflict can occur. The protocol contains some additional functionality to cope with these edge cases. First, it might happen that two or more agents in the same district panic at the same time. As the purchase or re-allocation of one bus is sufficient to solve their capacity problems, it is undesirable that they both proceed. Therefore, if, while in PANIC:AWAITING-RESPONSE, an agent receives a PANIC.WANT-TO-BUY from a district peer, it goes to PANIC:COLLIDED.

PANIC:COLLIDED Now the collision needs to be resolved. Instead of doing another round of communication, which would cost time and messages, we follow the social law that the oldest agent may continue. All other agents quit the protocol and go back to PANIC:UNINITIATED.

The other scenario is that two or more agents from different districts panic at the same time. This by itself is not problematic, and all agents may proceed with the protocol. However, if the agents end up buying new buses at the same time, this forms an issue. Agents do not know the ID of their newly bought bus beforehand - they only notice their new bus at the moment it announces its existence via the Discovery protocol. This means that if multiple agents come into existence simultaneously, the buyers have no way to know which of the new buses is theirs - and consequently, they do not know who to send the PANIC:COME-TO-MY-DISTRICT to. For this reason, if an agent detects such a simultaneous buy, it broadcasts a PANIC.WE-BOTH-BOUGHT message including its district as payload, and goes to the PANIC:BOUGHT-SIMULTANEOUSLY state.

PANIC:BOUGHT-SIMULTANEOUSLY This state is simply there for the agent to wait one time step for the replies to arrive. He sets the state to PANIC:RESOLVING-SIMULTANEOUS-BUY.

PANIC:RESOLVING-SIMULTANEOUS-BUY At this moment, all agents involved in the simultaneous buy know each others' ID and district. Following the same social law, all but the oldest agent quit the protocol. The oldest agent, in addition to instructing his own new bus, also sends the PANIC:COME-TO-MY-DISTRICT messages on behalf of the other buyers.

The Panic protocol can in fact be regarded as an extension of the classic Contract Net (CNET), in which Panic's initiating agent corresponds to CNET's manager (Smith (1977); Wooldridge (2009), Section 8.2.1). The PANIC.WANT-TO-BUY then is a task announcement, and the PANIC.DO-NOT-BUY, including the candidate contractor's FULLNESS in the payload, is the bid. The difference is that we extend the manager's options by buying a new bus if no suitable contractor registers. The PANIC.COME-TO-MY-DISTRICT that follows is essentially CNET's directed contract.

6.1 Social choice function

As mentioned above, the Panic protocol uses a social choice function to aggregate preferences for the capacity of the newly bought bus. The social choice function is thus of the form

$$\{\text{set of capacity preference orderings}\} \rightarrow \text{winning capacity},$$

for instance

$$\left\{ \begin{array}{l} \text{large} \prec \text{medium} \prec \text{small} \\ \text{medium} \prec \text{large} \prec \text{small} \\ \text{large} \prec \text{medium} \prec \text{small} \end{array} \right\} \rightarrow \text{small}.$$

Various such functions can be used, all with their own characteristics (Wooldridge, 2009, Section 12.2). One particularly interesting function for our purpose is the *Borda count*, since it is conceptually simple,

takes into account the full preference orderings and satisfies the Pareto condition. The drawback of the Borda count however is the *Borda paradox*: the phenomenon that adding or removing a candidate may drastically change the resulting relative orderings of other candidates.

We use an adapted version of the Borda count which we call the *Nauru count*, as it is used on the small island of Nauru. Unlike the Borda count, the scores of the Nauru count decrease increasingly quickly as the ranks go down (see Table 1). This way, adding or removing candidates within the lower ranks does not influence the outcome as much, dampening the effect of the Borda paradox.

Function	Borda count	Nauru count
Points for first rank	2	1
Points for second rank	1	1/2
Points for third rank	0	1/3

Table 1: Scoring schemes of the Borda and Nauru social choice functions.

7 Auction protocol

As mentioned in Section 3, agents randomly make themselves available for re-allocation, in order to redistribute capacity among the districts. Every time this happens, the agent at hand is considered a scarce resource, and as such, a competition takes place to decide which district gets it, in the form of an auction.

There are various types of auctions, each having their unique characteristics (Wooldridge, 2009, Section 14.1). An example of an auction protocol with attractive properties is the Vickrey auction (Vickrey, 1961). This second-price, sealed-bid auction has the special property that it is incentive-compatible, meaning it is any participant's dominant strategy to bid truthfully (Atallah and Blanton, 2009). This forms a countermeasure towards participants that attempt to strategically overbid or underbid.

However, as we are working with benevolent agents, we can simply program the agents to bid truthfully anyway, defeating the need for a Vickrey auction. Instead, our goal is to aggregate all individual agents' beliefs of district busyness, in order to truly give the auctioned agent to the district that is the most in need of an extra colleague. We therefore use a first-price, sealed-bid auction in which we average participant's bids over their districts. The bids themselves are simply the individual busyness beliefs (implemented as FULLNESS as explained in Section 6). The result, thus, is that the auctioneer receives an average belief of busyness per district, and selects the district with the highest average as winner.

The protocol works as follows.

AUCTION:UNINITIATED This is the default state in which the agent finds himself when he did not initiated the protocol yet. Every tick, a bus determines whether it is auctioning itself, based on a biased coin flip. When the auction begins, the message AUCTION.INITIATING-AUCTION is broadcasted to all agents There is no reservation price given. The state is set to AUCTION:AWAITING-RESPONSE.

AUCTION:AWAITING-RESPONSE This state is simply there for the agent to wait one time step for the replies to arrive. He sets the state to AUCTION:ABOUT-TO-AUCTION. Meanwhile, the other agents have received the AUCTION.INITIATING-AUCTION. They are all potential buyers, meaning they all respond with their offer by AUCTION.THIS-IS-MY-BID and including their FULLNESS as payload, indicating the attractiveness of the good for that particular bidder.

AUCTION:ABOUT-TO-AUCTION In this state the agent aggregates the responses. The auctioned bus checks whether there is at least one colleague in its current district, namely, there should always be at least one bus riding in a district. If this is the case, the auctioned bus finishes the auction by

averaging all offers per district, and selecting the district with the highest average bid. It could be the case that this is the auctioneer's own district, in which case the bus will just stay where it is. Otherwise, it relocates to this new district.

8 Discovery protocol

As agent information is not directly accessible in the environment, a communication protocol is needed to ensure agents are aware of each others' existence, such that they can broadcast messages when required. The simple Discovery protocol does this job. Each agent maintains the state variable DISCOVERY/COLLEAGUES, which is a list of agent IDs. When an agent comes into existence, it knows its own ID. In addition, available to every agent is the constant DISCOVERY/OLDEST-BUS, which is set to 24. The DISCOVERY/COLLEAGUES is therefore initialized to include all numbers from 24 up to the agent's own ID, since these agents are guaranteed to exist. Next, the new agent broadcasts the DISCOVERY.THIS-IS-MY-ID message. Upon receipt, other agents add the sender's ID to their DISCOVERY/COLLEAGUES list. This constitutes a simple and robust way to ensure all agents know of each others existence.

9 Results

Now that the full system is specified, it is interesting to see how well it performs on the various measures within a single run of 24 hours. The results can be found below in Table 2. For an overview of the amount of waiting passengers throughout the run, see Figure 4.

Passenger's average travelling time	187.81
Buses' expenses	705301.5
Number of messages sent by the buses	41967
Number of passengers waiting for a bus	2434
Average travelling time remaining	300.43
Final average travelling time	217.43

Table 2: Final results after 24 hours.

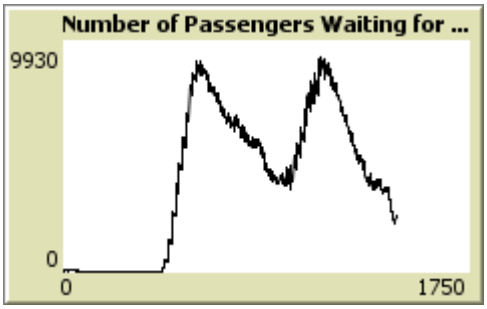

Figure 4: An overview of the amount of waiting passengers throughout the day.

References

Atallah, M. J. and Blanton, M. (2009). *Algorithms and theory of computation handbook, volume 2: special topics and techniques*. Chapman and Hall/CRC.

Maslow, A. H. (1943). A theory of human motivation. *Psychological review*, 50(4):370.

Russell, S. and Norvig, P. (1995). A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27.

Shoham, Y. and Tennenholtz, M. (1995). On social laws for artificial agent societies: off-line design. *Artificial intelligence*, 73(1-2):231–252.

Smith, R. G. (1977). The contract net: A formalism for the control of distributed problem solving. In *Proceedings of the 5th international joint conference on Artificial intelligence-Volume 1*, pages 472–472. Morgan Kaufmann Publishers Inc.

Steels, L. (1990). Cooperation between distributed agents through self-organisation. In *Intelligent Robots and Systems' 90.'Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on*, pages 8–14. IEEE.

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37.