Comparing two approaches for generating homogeneous multi-agent teams

Bachelor thesis
Artificial Intelligence
Radboud University Nijmegen

Eelke Boezeman
s0351954
7th of July 2008

Coordinators
Pim Haselager
Ida Sprinkhuizen-Kuyper
Contents

1. Introduction 3
2. Requirements 5
3. Task 6
4. Implementation 8
5. Experiment 9
6. Results 10
7. Conclusion 13
8. Literature 13
Appendix A 14
Appendix B 20
Abstract. In this bachelor thesis we studied the effects of using two different approaches in generating homogeneous multi-agent teams: the clonal and the a-clonal approach. The a-clonal approach was hypothesized to generate better performing teams on tasks that require the development of roles and dynamical role allocation. The task was implemented in a computer simulation. Results show that neither roles nor dynamical role allocation developed. This made a comparison between the clonal and a-clonal approach impossible, but did show why roles and dynamical role allocation might not have developed.

1. Introduction
Multi-agent teams are generally either homogeneous or heterogeneous. Both types have different properties, advantages and disadvantages, and are used for different purposes. The object of this study is the creation of homogeneous multi-agent teams. According to Matarić (1997) homogeneous multi-agent systems have three defining properties. In these systems all agents:

1. are equal, e.g. have the same hard- and software
2. are autonomous. There is no central control system coordinating the movement of the agents
3. share the same objective. They work as a team to fulfil a certain goal.

Many teams can be generated that have these three properties. Any team should maximize the third property whilst not violating the first two. Some teams are better than this than others. Generating a high performing team is a computational problem. Due to the total number of possible teams that can be generated, it is not possible to consider all teams within reasonable time.

1.1 Random selection
A solution to this problem would be to randomly generate and test teams. This approach considers only a fraction of all possible teams. It is likely that the best performing generated team is not the best if all possible teams were considered. However, this uncertainty is necessary to find an appropriate solution to the computational problem within reasonable time.

In the randomly generate and test-approach teams are selected randomly and thus no use is made of the properties of former generated high performing teams to generate new teams. If a good performing team is found the chance of finding another good performing team does not increase.

1.2 Evolutionary selection
Evolutionary algorithms do take advantage of former generated teams to create new teams. An evolutionary algorithm basically works as follows. It starts with testing a set of randomly generated teams. The worst performing teams are deleted. The best performing teams are used to generate new teams by making use of evolutionary techniques such as crossover and mutation. The new teams are evaluated and the cycle is repeated until a certain number of cycles is reached or the desired level of performance is found.
From every generation only the best teams can “reproduce” and create new teams. Every new team generated by evolution has a higher chance of performing better than former generated teams.

1.3 From agents to teams

Any team generated is a solution to the computational problem of finding a good performing homogeneous multi-agent team. An evolutionary algorithm generates high performing solutions within reasonable time. It produces these solutions as blueprints, also known as genotypes. The genotypes are actually not team specifications, as one would expect, but agent specifications. This introduces the need for two functions that (1) create teams from agents specifications and (2) map team performance to the performance of a single individual.

The seemingly most suited way to do this would be to generate all members of a team from a single genotype – the clonal approach (see Figure 1, from Quinn, 2001). Because a team generated by the clonal approach consists solely out of identical agents, team performance equals agent performance.

An alternative to this approach is the ‘a-clonal’ approach in which every team member has its own genotype (see Figure 2, from Quinn, 2001). In this approach it is much harder to relate team performance to the performance of a single individual. Because a team consists out of different genotypes it is not clear what each agent’s contribution is to the team performance. A rather computationally heavy solution to this problem is called Fitness sharing. Every genotype is tested in several different teams, each consisting out of different genotypes. The performance of a single genotype is calculated as the average performance of the teams it has participated in.

There seems to be no obvious reason to use the a-clonal approach to generate homogeneous teams. It evaluates mostly illegal (heterogeneous) teams and is computationally much harder than the clonal approach.

1.4 Roles and dynamical role allocation

Surprisingly, Quinn (2001) shows in a two robot coordinated movement task that the best team generated by the a-clonal approach outperforms the best individual generated by the clonal approach. In this task two identical agents must move together as far as possible. The agents have limited perception; they
easily lose track of each other. Team performance will thus be very low if the agents do not cooperate. It is critical that at one point one agent takes the lead and the other agent follows. This requirement for cooperation and role allocation increases the difficulty of the task greatly. Because the team is homogeneous it requires a single agent to (1) perform different roles in the team and to (2) ‘dynamically’ switch between these roles.

Genotypes in the clonal approach are evolved only in teams together with copies of themselves. Because only one genotype is used for the creation of every team member, a single genotype must represent all the roles that should be performed by the agents. The clonal approach requires the evolutionary algorithm to develop multiple roles and a mechanism to switch between these roles in a single genotype.

The advantage of the a-clonal approach over the clonal approach is that it allows for a separate development of the roles and the mechanism to switch between the roles. This separation in development is due to the fact that different genotypes evolve into different roles. A high team performance can be achieved early in the evolutionary process because the necessary roles are genetically determined. Later when the evolutionary algorithm converges into a single genotype a mechanism evolves that allows a single genotype to be able to perform multiple roles.

1.5. Aim of this study
The aim of this study is to investigate the performance of the clonal and the a-clonal approach in generating homogeneous multi-agent teams for tasks that require the development of roles and dynamical role allocation. We investigate the two approaches by testing the following hypothesis, derived from Quinn (2001):

The best individual generated by the a-clonal approach outperforms the best individual generated by the clonal approach used in a homogeneous multi-agent team on a task that requires dynamical role allocation.

From an experimental perspective the testing approach (clonal / a-clonal) is the independent variable. The dependent variable is the fitness score of the team created from the best performing generated individual.

In the following sections the requirements necessary to test the hypothesis are analysed and a testing environment is proposed that meets these requirements.

2. Requirements
Dynamical role allocation is the essential mechanism that underpins the success of the a-clonal approach over the clonal approach. The a-clonal approach is expected to be more capable of finding solutions for tasks that require such a mechanism to develop.

In a control study Quinn (2001) shows that the a-clonal approach does not outperform the clonal approach if the task is too trivial. It is therefore critical that the proposed testing environment requires the development of roles and
dynamical role allocation. As much as the task requires the development of these two aspects so must the agent control structure be able to represent such behaviour.

3. Task
Taking these requirements into account the following task is proposed that is considered being capable of testing the hypothesis. The task described is essentially a package routing task in which agents must transport packages from sources to goals as fast as possible within a predefined timeframe. In Figure 3 an image of the task environment is shown.

![Figure 3: A visual representation of the environment consisting of agents (red and green), sources (pink), goals (blue) and packages. The goal for the agents is to transport as many packages from sources to goals within a predefined timeframe.](image)

3.1 Specification
Agents inhabit the task environment and are the only autonomous objects in the sense that only they can change their state. Agents fly around and change their flying direction based on their perception of the environment. When an agent collides with another agent and one of the two transports a package it is exchanged between them.

At irregular intervals sources offer packages. Agents perceive sources that offer packages and upon collision between the agent and the source with a package it is exchanged between them. When the agent is transporting a package and it collides with a goal, the package is dropped off at the goal. The number of packages that are successfully transported by the agents from sources to goals within a predefined time interval denotes the fitness of the team. The better the team is at working together to transport packages the higher team performance and the obtained fitness score.

3.2 Agent control structure
The agent control structure, derived from Spector, Klein, Perry and Feinstein (2005), is basically a single layer neural network. The agent perceives the relative location of other objects in the environment represented as vectors.
Figure 4: Every agent perceives objects in the environment as vectors. Shown are vectors for Closest Agent with Package (green box), Closest Goal (top-left blue circle), Closest Source with Package (purple box) and Closest Source without Package (purple circle).

Figure 4 shows the example environment of Figure 3 added with four input vectors: Closest Agent with Package, Closest Goal, Closest Source with Package and Closest Source without Package. The control structure weighs these vectors and sums the weighted vectors together with the current flying direction. The agent continuously updates its flying direction by changing it according to the input it receives.

In Figure 3 and 4 two sources, two goals, two agents and two packages are shown. These amounts are arbitrary and can be configured to any number to suit the task and computational resources.

3.3 Cooperation
Cooperation between the agents is encouraged by slowing down an agent significantly when transporting a package. Although this makes it possible for every agent to deliver a package without cooperation, it takes much longer than with the help from another agent. Cooperation decreases the time needed to transport a single package and thus increases team performance.

3.4 Roles
The task is expected to require the development of these two roles:

- **Locating a package.** As soon as a package is offered at a source one of the agents flies towards the source and picks it up. Because only one of the agents can pick up the package, it’s not beneficial for both agents to fly towards the same source as soon as it offers a package.

- **Helping another agent transporting a package.** When another agent is transporting a package to a goal, it’s beneficial for both when they cooperate in transporting the package to a goal location.

These roles are expected to develop because agents are expected to cooperate. Cooperation is defined as any behaviour that leads to a faster transportation of packages. Faster transportation is achieved when the agents transport the package in turn and thus sharing the slowing burden of the package.
4. Implementation

The task, evolutionary strategy and the two approaches need to be implemented in such a way that it is easy to understand what the influence of each component is. The task also requires a certain amount of flexibility so that it is relatively easy to change the task configuration, or other parameters that control the simulation.

4.1 Model

A Model is designed to implement the simulation. The Model has three main components (see Figure 5):

\[ \text{Model} = \langle \text{Evolution, Tester, Environment} \rangle \]

![Figure 5: A visual representation of the Model with its three main components: Evolution, Tester and Environment.](image)

The two main components of the Model are Evolution and Environment. The Tester component sits in between and handles input and output of these components.

*Evolution* is an evolutionary mechanism that evolves genotypes. Genotypes are parameter settings for agents in the environment. After producing a fresh genotype it remains idle until a fitness score is given back. When the genotypes fitness is received Evolution iterates through the whole process again.

*Environment* is a two-dimensional space that contains moving agents that interact with other objects in the space. It also describes a fitness function that denotes how good the performance of the agents is according to a predefined task. The Environment is set up with a team to test. It produces a fitness value which is passed to the Tester.

*Tester* connects Environment and Evolution. It receives untested genotypes from evolution’s gene pool, generates a multi-agent team in accordance to the clonal or a-clonal approach and tests the team in Environment. After a number of tests the genotype’s fitness is established and reported back to Evolution to facilitate the evolutionary progress.

For a more detailed description of the model, see Appendix A.
4.2 Software

The simulation is implemented in breve, an open-source multi-agent simulation framework (Spector, Klein, Perry, Feinstein, 2005). Breve can be used to implement the genetic algorithm and the visual simulation. There is also a command line interface available that allows simulations to be run without visuals, vastly increasing the running speed.

5. Experiment

The task is configured as follows: every simulation contains two agents, two sources and two goals. Because only one package is allowed to exist in the Environment at any time, there is a maximum of only one package to deliver at any time. This encourages cooperation between agents. The locations of all objects are varied in three configurations that have the set up as pictured in Figure 3. The configurations differ in the distance between the starting position of the agents and the position of the sources and goals, and thus are increasingly difficult. Every genotype is tested in each configuration.

The evolutionary strategy that is used is the $\mu + 1$ strategy. The gene pool is filled with 30 random genotypes. After all have been tested a new genotype is generated from the gene pool using rank based selection. The new genotype replaces current worst genotype in the gene pool. After 500 generations the best genotype is considered as the solution of the run. All simulations are run either in clonal or a-clonal mode. Every 10 generations the current best genotype of both approaches is evaluated in a homogeneous team to compare between both approaches.

Due to lack of computational resources only five runs are executed for each approach. All parameter settings are shown in Table 1. See Appendix B for more parameter settings.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of runs executed per approach</td>
<td>5</td>
</tr>
<tr>
<td>Number of configurations on which each genotype is tested</td>
<td>3</td>
</tr>
<tr>
<td>Number of generations per run</td>
<td>500</td>
</tr>
<tr>
<td>Number of new genotypes generated every generation</td>
<td>1</td>
</tr>
<tr>
<td>Time (simulation iterations) used to test genotype on configuration</td>
<td>1000</td>
</tr>
<tr>
<td>Number of genotypes in gene pool</td>
<td>30</td>
</tr>
<tr>
<td>Number of trials after which best genotype is evaluated homogeneously</td>
<td>10</td>
</tr>
<tr>
<td>Number of sources</td>
<td>2</td>
</tr>
<tr>
<td>Number of goals</td>
<td>2</td>
</tr>
<tr>
<td>Number of agents in a team</td>
<td>2</td>
</tr>
<tr>
<td>Number of genes in genotype / Number of vector weights</td>
<td>7</td>
</tr>
</tbody>
</table>
Minimal agent velocity | 0.1
Maximal agent velocity | 1
Maximum number of packages allowed in transport | 1
Speed slowing factor for carrying a package (per simulation iteration) | - 0.01
Speed increasing factor for not carrying a package (per simulation iteration) | + 0.01

Table 1: Parameter settings of all simulations

6. Results

In Figure 6 the obtained fitness values of the best genotypes after five runs are pictured for both approaches. It is clearly visible that the genotypes produced by the a-clonal approach have a more varying performance than the genotypes produced by the clonal approach. The all-time best performing genotypes for both approaches gain a fitness value of 6.

![Fitness values of best genotypes after each of five runs](image)

Figure 6: Fitness values of best performing genotypes from both approaches. The a-clonal approach produces more varying results than the clonal approach, but both produce a genotype with a fitness value of 6.

The results show that the a-clonal approach did not outperform the clonal approach. Both approaches generated teams with a fitness value of 6. For this reason no statistical analysis was performed on the data. Table 2 summarizes the performance of both approaches and gives an indication of the difference in computational load between the two approaches.

<table>
<thead>
<tr>
<th></th>
<th>Clonal approach</th>
<th>A-clonal approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best fitness</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Average fitness</td>
<td>5.39</td>
<td>5.04</td>
</tr>
<tr>
<td>Standard deviation</td>
<td>0.37</td>
<td>1.15</td>
</tr>
<tr>
<td>Average approximate run time</td>
<td>10.5 Hours</td>
<td>21 Hours</td>
</tr>
</tbody>
</table>

Table 2: Results of both approaches after 5 runs per approach

6.1 Observed behaviour

The simulations show that the agents do not adapt to multiple roles, but instead adapt to just one single role, carried out by both agents. This role can be described as follows:
Stay close to the other agent and the nearest goal. When a package is offered at the source, go to it. When the package is picked up return to the goal location.

This role is a mix between the two roles that were expected to develop. It incorporates both locating a package at a source and helping another agent transport a package. The weight distribution in the agent control structure of the best genotypes from both approaches shows how this single role is represented. Figure 7 shows a typical evolution of the best genotype within a single run for both the clonal and a-clonal approach.

**A-clonal approach (run 1): Best genotype fitness values and weights**

![Chart showing fitness values and weights for A-clonal approach (run 1)]

**Clonal approach (run 2): Best genotype fitness values and weights**

![Chart showing fitness values and weights for Clonal approach (run 2)]

Figure 7: The evolution of the best genotype in both approaches. The weights are typical and in line with the optimal role behaviour the agents show.

In both approaches the vector locating the Source with Package vector has the highest weight and the Goal vector always has a lower weight than Source with Package. This is in line with observed single role behaviour. If no package is offered at any source than Source with Package is 0 and the agents circle the goal. When a package is offered, the source offering the package has a stronger attraction and the agents both move to the source. As soon as one of the agents
picks up the package Source with Package becomes 0 again. Now Goal has the highest attraction and thus the agents move to the goal. The agents remain there until another package is offered at a source and the process repeats. The weight distributions in Figure 7 explains how the single role behaviour is represented in agents.

The reason this behaviour develops is because it is the optimal strategy within the modelled environment. Development of multiple roles or even dynamical role allocation does not increase team performance on the task. There are various aspects of the simulation that could have caused this.

6.2 Fitness function
The used fitness function rewards agents for positive behaviour. The more packages transported the higher the fitness value of the team. No negative feedback, i.e. the distance travelled by the agents, was incorporated in the fitness function. The lack of negative feedback allows the agents to travel the same distance together without punishment. If negative feedback were implemented in the simulation travelling together would receive negative feedback from the fitness function. In that case cooperative behaviour would still be favoured, but only if the agents would not travel together. It can be imagined that this would aid the development of multiple roles.

6.3 Agent control structure
Another reason why no multiple roles develop could be that the agent control structure is not capable of representing roles or even dynamical role allocation. The control structure used in this simulation can be considered as a single layer neural network. It is possible that one layer is not complex enough to represent sophisticated behaviour and can only lead to single role behaviour.

In the current control structure input vectors are weighed and compete for influence in the flying direction of the agent. No interaction between inputs is possible, so no combined evaluation of multiple inputs is possible either. This causes the agent to be blind for events or combinations of events in the environment that indicate a certain relevant state of the environment.

The current control structure is not capable of representing many states of the environment. Because the agent control structure is reactive, perception is directly related to action. Consequently, the behaviour of the agent is a product of the states it can represent. If the agent is able to represent more states it is possible that the agent is also capable of showing more complex behaviour.

6.4 Scope of perception
The agents in this simulation have an unlimited scope of the world. Their perception exists out of input vectors which represent the location of other objects in the world, such as the closest other agent, source or goal. An agent knows the state of almost the entire world no matter where the agent itself is located within this world. Because of this “global perception” there is almost no uncertainty in the perception of the agent. Roles are likely to develop if agents depend on each other. With a global scope an agent does not need another agent for anything, but to increase the speed of transportation. No information is
exchanged between agents in a team, because every agent knows the entire state of the world. If the scope of the agents were limited agents would need to cooperate to exchange information, e.g. the location of a source with a package. This could contribute to the development of roles.

7. Conclusion
In this bachelor thesis we studied the effects of using two different approaches in generating homogeneous multi-agent teams: the “standard” clonal approach and “alternative” a-clonal approach. The a-clonal approach was hypothesized to generate better performing teams on tasks that require the development of roles and a mechanism to switch between these roles, dynamical role allocation. A simulation environment and a task were implemented and experiments were run in order to test this hypothesis. The results show that neither roles nor dynamical role allocation developed. This makes a comparison between the clonal and a-clonal approach impossible because development of such behaviour is a fundamental element of the hypothesis.

Although this research has tried to gain insight into the workings of the a-clonal and clonal approaches, it remains somewhat unclear what exactly constitutes an environment in which roles and dynamical role allocation are required to develop. Further research is necessary to discover and define the properties of such an environment. We hypothesize that in our study roles and dynamical role allocation did not develop due to a number of factors, such as the task, the fitness function, the agent control structure and the scope of perception of the agents.

Of these factors, the fitness function and agent control structure are the two most obvious aspects worth evaluating first. The fitness function should incorporate negative feedback encouraging the agents to cooperate but not to develop the same role. The agent control structure should be designed to be able to perceive and act on more subtle differences between environmental states. Combinations of input vectors must be weighed and used as triggers for certain behaviours. This can only be achieved if the control structure is more complex. Adding more layers to the current single layer network is a likely approach.

With these amendments it is possible that roles and dynamical role allocation do develop. The development of these properties would enable the clonal and a-clonal approach to be compared in generating homogeneous multi-agent teams.

8. Literature


Appendix A: Model description

This section gives an exact description of the Model. The Model contains three elements: Environment, Evolution and Tester. Each of these elements and their sub-elements are described in detail to give an exact account of the implementation.

1. Environment

Environment = < World, Agents, Task >

The Environment contains three main objects: a spatial virtual World, Agents that move around within and interact with the World a Task that specifies the desired state of the World.

1.1 World

- The virtual space of the World is two-dimensional.
- The World contains several different objects. Every object has a state that is altered by pre-set events or interaction with other objects.
- Passive objects (Packages, Sources and Goals) cannot execute actions or initiate interaction with other objects and get altered by pre-set events or actions of active objects.
- Active objects (Agents) function autonomously and perform actions on other objects.
- The state of every object is updated every iteration of the Environment.

1.1.1 Package

- A Package is an object that can be transferred from Agent to Agent, Source to Agent and Agent to Source. It is passive in the sense that it cannot perform actions that alters its own state.
- The number of Packages that exists in the World cannot exceed $N_{packagesAllowedInWorld}$.
- Packages can only come into the World at Sources. $N_{packagesOffered}$ denotes the number of Packages instantiated at Sources.
- Packages disappear at Goals when delivered by an Agent. $N_{packagesDelivered}$ denotes the number of Packages delivered at Goals.

1.1.2 Source

- A Source is a passive object that offers Packages.
- There are $N_{source}$ Sources in the World. The Tester’s Configurator determines the location of every Source.
- $N_{packagesOffered}$ denotes the number of Packages offered by all Sources in the World. If a Source offers a Package $N_{packagesOffered}$ is increased by one.
- A Source offers a Package on two conditions:
  - the number of Packages currently existing within the World does not exceed $N_{\text{maxPackagesAllowedInWorld}}$.
  - A random amount of time (not bigger than $T_{\text{maxWaitingTimeToOfferPackage}}$) has passed.

1.1.3 Goal

- A Goal is a passive object that receives Packages from Agents.
- There are $N_{\text{goal}}$ Goals in the World. The Tester’s Configurator determines the location of every Goal.
- If a Goal receives a Package $N_{\text{packagesReceived}}$ is increased by one.

1.2 Agents

An Agent is an object that exists within the World but is different from the Source, Goal and Package in the sense that it actively changes the state of the World. It is considered autonomous and acting to obtain a certain objective.

$N_{\text{Agent}}$ denotes the number of Agents that exist within the Environment.

Every Agent has the following properties:

- An Agent has a spatial location and moves around within the two dimensional Environment. Both the Agent’s location and flying direction are represented by vectors that get updated every iteration of the Environment.
- There is no maximum value for the location of an Agent. The flying velocity must be between $\text{minVelocity}_A$ and $\text{maxVelocity}_A$.
- An Agent’s energy (between $\text{minEnergy}_A$ and $\text{maxEnergy}_A$) affects the flying velocity of the Agent.
- An Agent can transport a Package. Transporting the Package lowers the Agent’s energy by $\text{energyStep}_A$ per iteration. Vice versa, not transporting a Package raises the Agent’s energy by $\text{energyStep}_A$.
- A Package is exchanged between an Agent and another object in the following situations:
  - An Agent not transporting a Package collides with a Source offering a Package.
  - An Agent transporting a Package collides with a Goal.
  - An Agent collides with another Agent and one of them transports a Package.
After an Agent exchanges a Package with another object, being another Agent, Source or Goal, it cannot exchange for a certain number of iterations denoted by collisionTransferDelay.

An Agent is aware of the location of the following other objects in the Environment. These are the Agent’s input vectors:

- The nearest other Agent with a Package
- The nearest other Agent without a Package
- The nearest Source offering a Package
- The nearest Source not offering a Package
- The nearest Goal
- Random location
- Centre of the World

The control structure of an Agent specifies a weight for every input vector. Every iteration the Agent’s flying direction is computed by weighing the input vectors together with the Agent’s current flying direction.

1.3 Task

The Task specifies what the Agents in the World should do, thus what behaviour is good and what behaviour is not. This vague notion is captured by the Fitness function that maps Agents’ behaviour to a performance value.

Task has the following properties:

- The Fitness function defines the performance of the Agents as the number of Packages delivered at Goals successfully. The higher the number of Packages delivered, the higher the fitness of the team of Agents.

- The Task ends when a certain amount of time has passed defined by $T_{timeToTestTeam}$.

2. Evolution

Evolution = < Gene pool, Evolutionary Strategy >

Evolution has two main components. Gene pool can be considered as a dynamical complex data container and the Evolutionary Strategy as the set of actions that are performed on the data every evolutionary iteration.

2.1 Gene pool

A Gene pool is a collection of Genotypes. The number of Genotypes in the Gene pool is equal to $N_{genotype}$.
- There is only one Gene pool at any given time. The state of the Gene pool changes every evolutionary iteration. In such an iteration the current Gene pool is extended with new Genotypes that are created from Genotypes within the current Gene pool.

- A Genotype consists of a number of Genes denoted by $N_{\text{genesPerGenotype}}$. $N_{\text{genesPerGenotype}}$ is equal to $N_{\text{agentWeights}}$, the number of weights in the control structure of an Agent.

- A Gene is a floating point number and its value is within the interval of $\text{minGeneValue}$ and $\text{maxGeneValue}$.

- For every Genotype a Fitness value can be calculated that indicates the quality of the genotype with respect to the Task defined in the Environment.

### 2.2 Evolutionary Strategy

If the Gene pool is empty, initialize the Gene pool with random Genotypes and test all Genotypes to obtain a fitness value for each Genotype.

When given all Genotypes are tested new offspring is generated by applying the $\mu + 1$ algorithm. Mu stands for the size of the Gene pool and + 1 means that every evolutionary iteration one new child Genotype is generated from parents in the current Gene pool.

The following is executed every evolutionary iteration:

- Increase the size of Gene pool ($\mu$) by 1:
  - Select parents through executing a tournament selection for each parent. Parents must be different Genotypes.
  - Perform crossover on the two genotypes: there is 50% chance for every Gene of each parent’s Genotype to end up on the child Genotype.
  - Every gene of the child Genotype has a chance of $P_{\text{mutateGene}}$ to get mutated. The amount of mutation is computed in the following way:
    - $\text{newGeneValue} = \text{currentGeneValue} \times \text{mutationFactor}$
    - $\text{mutationFactor}$ is the percentage of change applied to the Gene and lies within the interval of $\text{minMutationFactor}$ and $\text{maxMutationFactor}$.
  - Test the new Genotype and obtain a fitness value.
  - Bring the size of Gene pool back to $\mu$ by removing the Genotype with lowest fitness from the Gene pool.
3. Tester

Tester = < Generator, Configurator, Evaluator >

Tester links Evolution and Environment together. It executes iterations in both and relays genotypic information between them. The approach (clonal or a-clonal) influences the way teams are generated and tested. Independent from the current approach Configurator defines the configurations in which teams are tested.

3.1 Generator

Two approaches, clonal and a-clonal, differ in how teams are generated from an untested Genotype. The untested Genotype is obtained from Evolution.

- Clonal approach: create team with $N_A$ clones of the untested Genotype.

- A-clonal approach: create $N_{aclonalTeamsPerGenotype}$ number of a-clonal teams in such a way that no Genotype is represented in more than one team, with the exception of the untested Genotype that is represented once in every team.

3.2 Configurator

To get a good performance measure and minimizing the influence of coincidence multiple starting configurations are specified.

- $N_{configuration}$ is the number of configurations specified for which each team is tested.

- Every configuration defines:
  
  - $N_{Source}$, $N_{Goal}$ and $N_{Agent}$ that denote the number of Sources, Goals and Agents in a team respectively.
  
  - Start Location, orientation and velocity of Agents
  
  - Start Location of Sources and Goals.

3.3 Evaluator

Generated teams are evaluated in the Environment. After a predefined time the test ends and the fitness value is recorded. The number of tests per team depends on the approach and the number of configurations specified. To further eliminate the influence of coincidence teams a retested a number of times per configuration.

- Test team in configurations:
  
  - Clonal:
- Test team for every configuration specified by Task a number of times defined by $N_{testConfigurationClonally}$.

- Number of tests = number of task configurations $\times N_{testConfigurationClonally}$

  o A-clonal:

    - Test team for every configuration specified by Task a number of times defined by $N_{testConfigurationA-Clonally}$ from each of the different Agent starting configurations.

    - Number of tests = number of task configurations $\times$ number of Agent starting configurations per task configuration $\times N_{testConfigurationA-Clonally}$

  o The number of tests that is executed to calculate the Genotype's fitness thus depends heavily on the approach that is used.

- An average fitness value is calculated from the obtained fitness values of each of the tests and reported back to the Evolution.
Appendix B: Parameter settings

The following table shows the parameter settings as they were used in the experiments. See Appendix A for parameter definitions.

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>$N_{\text{packagesAllowedInWorld}}$</td>
<td>1</td>
</tr>
<tr>
<td>Source</td>
<td>$N_{\text{source}}$</td>
<td>2</td>
</tr>
<tr>
<td>Source</td>
<td>$N_{\text{maxPackagesAllowedInWorld}}$</td>
<td>1</td>
</tr>
<tr>
<td>Source</td>
<td>$T_{\text{maxWaitingTimeToOfferPackage}}$</td>
<td>300</td>
</tr>
<tr>
<td>Goal</td>
<td>$N_{\text{goal}}$</td>
<td>2</td>
</tr>
<tr>
<td>Agent</td>
<td>$N_{A}$</td>
<td>2</td>
</tr>
<tr>
<td>Agent</td>
<td>$\text{minVelocity}_{A}$</td>
<td>1</td>
</tr>
<tr>
<td>Agent</td>
<td>$\text{maxVelocity}_{A}$</td>
<td>4</td>
</tr>
<tr>
<td>Agent</td>
<td>$\text{minEnergy}_{A}$</td>
<td>4</td>
</tr>
<tr>
<td>Agent</td>
<td>$\text{maxEnergy}_{A}$</td>
<td>1</td>
</tr>
<tr>
<td>Agent</td>
<td>$\text{energyStep}_{A}$</td>
<td>0.01</td>
</tr>
<tr>
<td>Agent</td>
<td>$T_{\text{timeToTestTeam}}$</td>
<td>1000</td>
</tr>
<tr>
<td>Task</td>
<td>$T_{\text{timeToTestTeam}}$</td>
<td>1000</td>
</tr>
<tr>
<td>Evolution</td>
<td>$N_{\text{genotype}}$</td>
<td>30</td>
</tr>
<tr>
<td>Evolution</td>
<td>$N_{\text{genesPerGenotype / agentWeights}}$</td>
<td>7</td>
</tr>
<tr>
<td>Evolution</td>
<td>$P_{\text{mutateGene}}$</td>
<td>0.2</td>
</tr>
<tr>
<td>Evolution</td>
<td>$\text{minMutationFactor}$</td>
<td>0</td>
</tr>
<tr>
<td>Evolution</td>
<td>$\text{maxMutationFactor}$</td>
<td>0.1</td>
</tr>
<tr>
<td>Tester</td>
<td>$N_{\text{configuration}}$</td>
<td>3</td>
</tr>
<tr>
<td>Tester</td>
<td>$N_{\text{testConfigurationClonally}}$</td>
<td>2</td>
</tr>
<tr>
<td>Tester</td>
<td>$N_{\text{testConfigurationA-Clonally}}$</td>
<td>4</td>
</tr>
</tbody>
</table>