Universitat de Barcelona

Departament de Matemàtica Aplicada i Anlisi

Norm Generation in Multi-Agent Systems

by

Jan Felix Koeppen

Master's Course in Artificial Intelligence

Master's Thesis

Supervised by:

Maite López Sánchez

September 2009

Contents

List of Figures

List of Tables

1 Introduction

Norm emergence and normative system are topics of many current investigations in the area of artificial intelligence. This work will investigate a combined approaches of normative multi-agent systems with an experiences based norm generation system, based on case-based reasoning. Important concepts that are being utilized in this work are those of agency and multi-agent systems, agent-based simulation, case-based reasoning and normative MAS. The following sections will introduce these areas.

1.1 Agents and Multi-Agent Systems

The way software is designed and implemented has come a long way since the first machine instructions where programmed using microcode. In conventional software engineering, imperative architectures are prevailing, where an algorithmic structure processes instructions, starting from some given entry point. The most popular paradigm nowadays is object oriented programming, which is characterized by such an imperative style. A software program based on OOP abstracts a domain model by creating classes and instances of its containing objects, i.e. a car or a customer. These classes contain different properties (fields), which model their state, and methods, which allow another component to manipulate and or query its state. Execution of such a program usually is based on hard coded instructions, i.e. send a mail to all customers that have bought a car in the last 3 months, delete every file that has not been used for some time or similar.

With the rise of concepts and approaches revolving around the area of artificial intelligence, the need for a more differentiated paradigm emerged. In artificial

intelligence, simulations that react strictly imperative are not primarily desirable. Instead of making a software component execute a given action at a given time, one would expect of an intelligent system, that is chooses its actions independently; more autonomy is required.

Based on this idea, the concept of software agents was introduced. The term agent goes back to the concept of agency. Agency most broadly describes the representation of someone, who takes action according to the interest of someone else. In this context, i.e. an estate agent buys and sells estate on behalf of his/her client. A software agent therefore can be described that gets introduced into a software environment, where it takes actions on the behalf of its owner.

The actions that are taken by a software agent are not assigned upon the agent from an external source, but rather are deliberated by the agent itself as a result of its perception of the environment. The agent concept describes therefore a component that solely has two interfaces to interact and percept a given environment; sensors that are used to measure the state of its environment and actuators that are used to manipulate it.

Agents can be categorized by their internal complexity and the level of awareness of the environment they live in. The most basic type of an agent is a simple reflex agent. This type of agent is not capable of storing historical data, but rather reasons based on a set of rules, that lets him perform a given action based on the current state of the world. An example would be some kind of climate control for a room; when the temperature measured by the agent's sensors drops below a certain level the agent activates his actuator, which causes the climate control to start heating. As soon as second temperature level is reached and the climate control is still heating, the agent instructs it to stop heating.

The formerly introduced agent architecture can be extended in several ways. For example, the reasoning could make use of a world model, which enables prediction of the outcome of certain actions, which therefore allow for better-informed decisions. Usually it is the case, that an agent has only limited access to it's environment's state, e.g. it only 'sees' the area around it or -like the previously mentioned agent- only has access to the current state. A model can allow an agent to keep further

track of the world by remembering areas that are currently not visible or events that took place in previous steps. For example, an agent could reason that a car that is behind itself and is now nearer than before is about to overtake.

Another important aspect in the reasoning architecture of agents is defined by goals. Neither the simple reflex- nor the model based reflex agents maintain information about goals that are desirable for the agent. Goals describe a more abstract structure of the decision finding process. While a simple reflex agent has only rules what to do when a certain event occurs, the goal based agent might change its actions based on experiences made in the past - it 'thinks' in a where-i-want-to-go manner. This kind of reasoning introduces the possibility to evaluate certain states to be desirable or undesirable and therefore lets one differentiate between different plans that pass those kinds of states. Planning and searching are subfields of AI that are dedicated to create such plans. When the agent additionally makes use of utility functions, its structure is evolved enough to demonstrate high-quality behavior. The utility of a plan (a sequence of actions) can differ, even though they have the same terminal state. One plan could be considered to be of higher utility for example, when it inhabits a smaller amount of steps or reaches a goal in less time than another one.

Following rus the environment in which a single or a multitude of agents reside can be categorized by a number of different attributes.

- *Observability*: An environment can either be fully or partially visible. In fully visible environments an agent is capable of sensing the states of every 'field', whether its adjacent to his current position or not. Examples for such kinds of environments could be a backgammon game or the analysis of an image. In contrast the environment is partly visible, if the observability is limited by certain factors that block access to the perception of some area of the environment. A real world example can be walls or corners, but there are many more possible obstacles in observation. Examples for partially visible environments are a game of poker, where the participating players do not have knowledge of other players cards or the navigation of a car through a city, where the agent only seed the traffic situation around it.

- *Determinism*: An environment is deterministic if the same action at a given state always yields in the same outcome. In a deterministic environment it is therefore possible to predict the state of the world in the next step. When an environment contains uncertainty, for example if the execution of an action by the agent might not succeed, it is referred to as a stochastic environment. Non-determinism is also prevailing if the environment is populated by other agents that take actions on their own and therefore manipulate the world state in upcoming steps. Environments that are manipulated by several agents at a time but are otherwise deterministic are referred to as strategic.

- *Episodic and sequential*: When the decisions taken by an agent in a certain environment do not depend on actions from previous steps, the environment is called episodic. On the other hand, when decisions need knowledge of further steps, the environment is called sequential.

- *Static and dynamic*: The behavior of static environments can be described as being round based. When an agent needs some time before taking a decision and therefore acting, a static environment waits until the agents has executed its action for the round. If the environments might change its state while the agents is deliberating, it is dynamic. In this type of an environment the agents has to consider the length of its deliberation process and might update its sensors before the found action is executed to assure, that the world is still in an appropriate state for the action.

- *Discreteness*: The discreteness of an environment is determined by the way it handles time and space. Discreet environments usually only occur in simulations, where more difficult states are abstracted to discreet states. Real-World environments are always continuous, meaning it has continuous time and state spaces. For the different scenarios for the agents, different characteristics have to be chosen. Because some attribute choices may inflict a higher simulation complexity, it is generally advisable to reduce the complexity of the run-time environment to the point that is necessary to run it. Especially characteristics like continuousness and non-determinism raise the difficulty from the agents-perspective enormously and require techniques to handle them.

While a single agent in a simulation can be useful for basic research and prototyping, much more interesting applications are possible when several agents are interacting in one environment at the same time. Popular examples for multi-agent simulations include online-trading, disaster response or the modeling of social and political structures.

The current state of the art of multi-agent systems is influenced by a great number of other disciplines, such as philosophy, logic, game theory, economics, social sciences and ecology. While these interdisciplinary influences bring the advantage of bringing well-founded methodologies into application, it inflicts also the draw back, that there are many different views as to what the field is about.

Running in dynamic environments is a major difference to classic programs. When a program executes in a static environment there is no need to recheck and update its knowledge about it and hence does not necessarily has to supervise its own success. Execution in dynamic environments and making use of at least partly reliable sensors and actuators -like it is the case in real world situations- introduces uncertainty that has to be taken care of. There are many different techniques available to cope with this, many originating in the field of probabilistic theory. The agent has to take the possibility of failure into account and has to decide whether the execution of some action is worth the effort at the given moment.

Another essential aspect of agents that run in a multi-agent system is sociability. To communicate, the agents that live together have to speak a common agent-communication language that defines standards on how to exchange information. This is especially the case when the agents are designed by different parties. The two most popular standards in this domain are FIPA-ACL (Foundation for Intelligent Physical Agents - Agent Communication Language) and KQML (Knowledge Query and Manipulation Language). Both of these languages have definitions about how a certain message has to be formatted. Since those languages are both based on speech act theory, developed by John Searle in 1960, they cover overlapping definitions for certain performatives.

- *Representatives*: Transfer knowledge as perceived by one agent, e.g. the door is open. The receiver interprets this kind of information as "Agent a thinks,

that the door is open". KQML and FIPA-ACL call this performative "tell"

- Directives: Request to another agent to perform a certain task.
- *Commisives*: Expression of intention of the sending agent, e.g. "I promise to open the door".
- *Expressives*: The sending agent tries to express something about its mental state, e.g. "thank you"
- Declarations: Expressing the declaration of war or christening. Along with the performative of a message in an agent communication language a message contains information about communication parameters (sender, receiver, subject), the message content and message meta-data. The meta-data of the message can be used to define information about how to interpret the content by making statements about the content language (e.g. LISP, KIF, RDF, FIPA-SL), the according ontology (e.g. stock market, pharmacy or social networks) or encoding settings.

The agents used in this work are simple reflex agents, that react on a predefined pattern. The environment utilized is partial observable (an agent only has access to attributes of his current position), non-determinism (since the action in the same state may lead to different results), episodic (knowledge about MAS system is maintained externally, static (execution is synchronous) and discrete.

1.2 Case-Based Reasoning

Case-based reasoning is a technique that solves new problems based on experience from the past (Aamodt and Plaza [1994]). The idea is inspired by the natural way of solution finding that is used by humans on a day-by-day basis. As an example one can imagine somebody who tries to solve a problem with no given knowledge. This person would start to experiment by applying random solutions to the problem and observe the outcome. When one sees that an applied solution resulted in better performance as others with a similar problem, he is likely to reapply the same

solution or a slightly improved version to reachieve the formerly good results or even improve them further.

The same trial and error approach can be found in case-based reasoning. The process consists of four essential phases:

1. Retrieve: Given a target problem, retrieve cases from memory that are relevant to solving it. A case consists of a problem, its solution, and, typically, annotations about how the solution was derived. For example, suppose Fred wants to prepare blueberry pancakes. Being a novice cook, the most relevant experience he can recall is one in which he successfully made plain pancakes. The procedure he followed for making the plain pancakes, together with justifications for decisions made along the way, constitutes Fred's retrieved case.

2. Reuse: Map the solution from the previous case to the target problem. This may involve adapting the solution as needed to fit the new situation. In the pancake example, Fred must adapt his retrieved solution to include the addition of blueberries.

3. Revise: Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary, revise. Suppose Fred adapted his pancake solution by adding blueberries to the batter. After mixing, he discovers that the batter has turned blue - an undesired effect. This suggests the following revision: delay the addition of blueberries until after the batter has been ladled into the pan.

4. Retain: After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory. Fred, accordingly, records his newfound procedure for making blueberry pancakes, thereby enriching his set of stored experiences, and better preparing him for future pancake-making demands.

Figure 1.1 gives an overview of the CBR cycle.

Case-based reasoning is a common reasoning technique in experts systems but well fits the domain of multi-agent simulations. It works with the premise that similar problems have similar solutions.

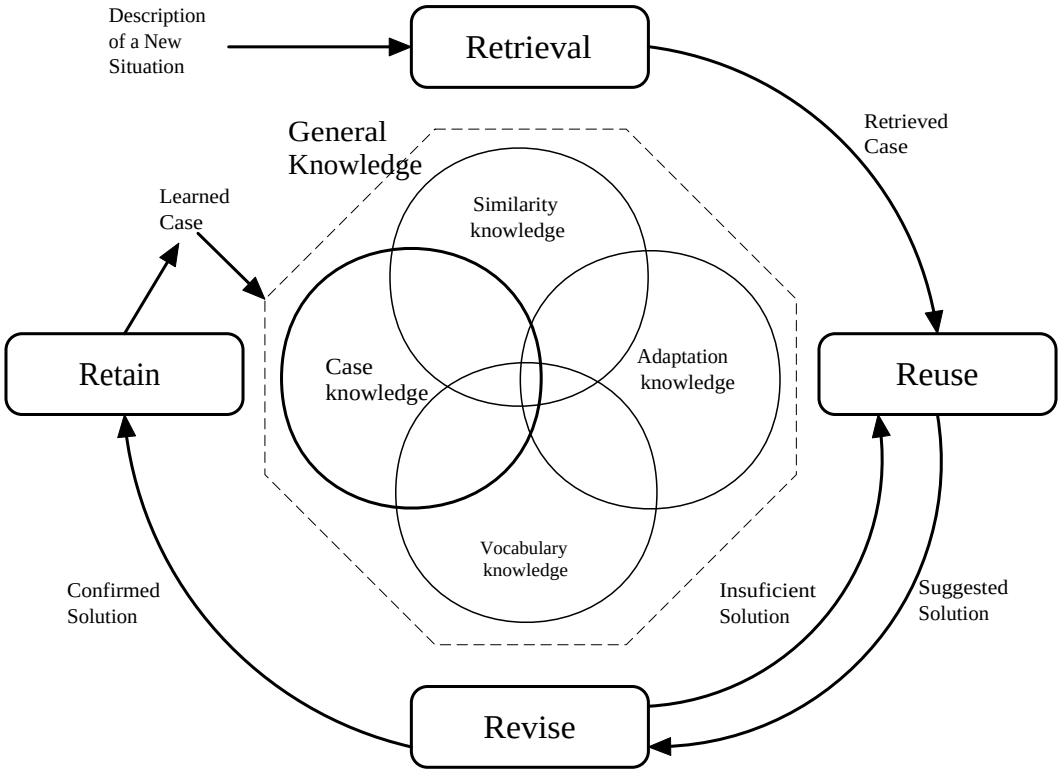

Figure 1.1: The CBR Cycle.

1.3 Agent Based Simulation

The area of Multi agent Based Simulation (MABS) area which brings together researchers active within the agent-based social simulation community (ABSS) and the multi agent systems community (MAS).

The focus of ABSS is on simulating and synthesizing social behaviors in order to understand observed social systems of humans, animals and even electronics. Their research employs development and testing of new models and concepts.

MAS, on the other hand, focuses on the solution of hard engineering problems related to the construction, deployment and efficient operation of multi agent-based systems.

As became clearer over the last years, these two communities have much to learn from each other. Real human societies are generally self-organizing, highly scalable, robust and open, and the ABSS community has developed a sizable set of techniques, observations and models that give insight into some of the mechanisms that underpin these kinds of systems.

However, ABSS has not concerned itself with applying these techniques to solve engineering problems. Conversely, the MAS community is concerned with creating working agent systems that solve real problems. This focus has forced many to abandon experimentation with large-scale systems (thousands of agents) composed of smart autonomous agents (e.g., complex adaptive learners) due to the lack of traditional techniques (and/or computational resources) for managing such complexity.

There are held MABS workshops, that try to support the dialogue of the two areas. Collaborations in this environment embraced among other things sociological issues such as cooperation, trust and power hierarchies, being broached from an engineering perspective. Hales [2003]

The different foci of the two approaches have led to different research methodologies. The development cycle of traditional multi-agent systems -especially when distributed- require technical overhead on the intrinsic agent logic. To design an

agent that is capable to join a certain MAS there has to be some agreement on the communication protocol and platform (usually based on FIPA-ACL or KQML). The ABS approach tries to reduce this overhead and concentrates on the search for patterns in the interaction between individual agents.

A fairly big set of tools and frameworks have evolved around the different disciplines. Some examples for the frameworks that emerged from MAS research are BDI[1] agent platforms, such as JADEX[2] , Jason[3] or the 3APL platform[4].

The tools and platforms on the ABS side usually stand out by their rapid prototyping capabilities. Their goal is to allow the fast implementation of simulations that allow the analysis of emerging behaviors in agent interactions. Popular ABS simulation environments are NetLogo[5], Swarm[6] and Repast[7] and MASON[8].

The effort to set up a simulation in an ABS environment usually only requires the definition of the agents behavior and some settings on the environment. Since ABS are vastly being utilized in non-technical areas such biology, ecology, economics, political science, sociology and others, the researchers usually lack a computer programming background. For this reason, many ABS platforms try to ease the agent implementation process by supplying graphical user interfaces to support the specification of agent logic. Also, many require powerful analysis features that remove the focus from a single agent to a more holistic perspective.

1.3.1 NetLogo

NetLogo is a project originated at the Center for Connected Learning and Computer-Based Modeling at Northwestern University, Illinois, USA. It is particularly well suited for modeling complex systems developing over time. Modelers can give

[1] A software model to program agents based on belief, desire and intentions
[2] see http://jadex.informatik.uni-hamburg.de
[3] see http://jason.sf.net
[4] see http://www.cs.uu.nl/3apl
[5] see http://ccl.northwestern.edu/netlogo
[6] see http://www.swarm.org
[7] see http://repast.sourceforge.net
[8] see http://cs.gmu.edu/ eclab/projects/mason

instructions to hundreds or thousands of independent agents all operating concurrently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals. Its huge user base makes it one of the most widely used platforms for ABSs.

Netlogos is design as an educational tool, since it is the easies to use of the set. Its programming language includes many high-level structures and primitives that greatly reduce programming effort, and extensive documentation is provided. The language contains many but not all the control and structuring capabilities of a standard programming language. Further, NetLogo was clearly designed with a specific type of model in mind: mobile agents acting concurrently on a grid space with behavior dominated by local interactions over short times. While models of this type are easiest to implement in NetLogo, the platform is by no means limited to them.

1.3.2 Swarm

Swarm was designed as a general language and toolbox for ABMs, intended for widespread use across scientific domains. Key to Swarm is the concept that the software must both implement a model and, separately, provide a virtual laboratory for observing and conducting experiments on the model. Another key concept is designing a model as a hierarchy of swarms, a swarm being a group of objects and a schedule of actions that the objects execute. One swarm can contain lower-level swarms whose schedules are integrated into the higher-level swarms; simple models have a lower-level model swarm within an observer swarm that attaches observer tools to the model. The software design philosophy appears to have been to include software that implements Swarms modeling concepts along with general tools likely to be useful for many models, but not to include tools specific to any particular domain. Swarm was designed before Javas emergence as a mature language. Swarm uses its own data structures and memory management to represent model objects; one consequence is that Swarm fully implements the concept of probes: tools that allow users to monitor and control any simulation object, no matter how protected

it is, from the graphical interface or within the code.

Swarm is implemented as a library written in Objective-C. The reason for this was this languages lack of strong typing (in contrast to, e.g., C++). It supports the complex-systems philosophy of lack of centralized control; e.g., a models schedule can tell a list of objects to execute some action without knowing what types of object are on the list. A strong request in the user community to have access to the Swarm library by using Java code motivated the development of *Java Swarm*. Java Swarm is designed to build a bridge to allow this access with as little change as possible. Java Swarm therefore simply allows Java to pass messages to the Objective-C library, with work-arounds to accommodate Javas strong typing.

1.3.3 MASON

MASON was designed as a smaller and faster alternative to Repast, with a clear focus on computationally demanding models with many agents executed over many iterations. Design appears to have been driven largely by the objectives of maximizing execution speed and assuring complete reproducibility across hardware. The abilities to detach and re-attach graphical interfaces and to stop a simulation and move it among computers are considered a priority for long simulations. MASONs developers appear intent on including only general, not domain-specific, tools. MASON is the least mature of these platforms, with basic capabilities such as graphing and random number distributions still being added.

1.3.4 Repast and Repast Simphony

Repast development appears to have been driven by several objectives. The initial objective was to implement Swarm, or equivalent functionality, in Java. However, Repast did not adopt all of Swarms design philosophy and does not implement swarms. Repast was also clearly intended to support one domainsocial sciencein particular and includes tools specific to that domain. The additional objective of making it easier for inexperienced users to build models has been approached in several ways by the Repast project. These approaches include a built-in simple

model, and interfaces through which menus and Python code can be used to begin model construction.

Repast Simphony is the successor of Repast. It employs a comfortable user interface that allows unexperienced users to design agent logic and simulation behavior in flow-chart like interface. The simulation setting can be modified in a broad spectrum.

Repast Simphony has made a big step toward supporting users with tools to trace the course of a simulation. The simulation environment delivers an easy way to create graphical output of the running simulation, runtime control (like step by step execution, debugging, live modification of agent setting etc.) and has interfaces to export data to a broad range of statistical and mathematical programs like MatLab and S. Interfaces to terracotta, Grass and many more are available as well (rep [2009]).

The simplified agent logic, even though it is well done and allows to setup complex agents, is not sufficient for the simulation employed in this work. For this case, Repast delivers an API that can be accessed by using Java or Groovy, which gives a developer the full flexibility of hand tailored logic implementations.

1.4 Normative MAS

In their introduction to normative multi-agent systems Boella et al. [2007], Boella et al. give the following definition:

> A normative multi-agent system is a multi-agent system together with normative systems in which agents on the one hand can decide whether to follow the explicitly represented norms, and on the other the normative systems specify how and in which extent the agents can modify the norms."

This chapter first describes the distinction among various kinds of norms and later discusses their possible integration in normative MAS.

1.4.1 Norm Types

The definition of a norm is interpreted individually in different areas. For example, in sociology, a norm is a rule or standard of behavior shared by members of a social group (Encyclopedia Britannica). According to philosophy, a norm is an authoritative rule or standard by which something is judged and, on that basis, approved or disapproved (Columbia Encyclopedia). Examples of norms include standards of right and wrong, beauty and ugliness, and truth and falsehood. According to economics, a norm (from norma, Latin for carpenters level) is a model of what should exist or be followed, or an average of what currently does exist in some context, such as an average salary among members of a large group.

From the normative system perspective, the literature distinguishes several kinds of norms:

- *constitutive norms* that regulate the creation of institutional facts as well as the modification of the normative system itself.
- *Regulative norms* that describe obligations, prohibitions and permissions (both Boella et al. [2007])
- *procedural norms* are rules governing the way in which political decisions are made; they are not concerned with the content of any decision except one which alters decision-making procedures. Procedural norms have long been considered a major component of political systems, particularly democratic systems.

Constitutive norms

Following the above description, examples for constitutive norms are such as:

> "X counts as a presiding official in a wedding ceremony"
>
> "this bit of paper counts as a five euro bill"
>
> "this piece of land counts as somebodies private property"

Cited from Boella and van der Torre [2006]

Boella et al the role of constitutive rules is not limited to the creation of an activity and the construction of new abstract categories. Constitutive norms specify both the behavior of a system and the evolution of the system... Boella and van der Torre [2004].

Norm revision by dynamics of system, certain actions add norms (e.g. amandments): the normative system must specify how the normative system itself can be changed by introducing new regulative norms and new institutional categories, and specify by whom the changes can be done Boella and van der Torre [2004].

Today US government agencies are required to invite public comment on proposed rules Lau et al. [2005] This is done through the digital government interface and allow revisions to be traced.

Another aspect of constitutive norms is organizational and structural, that is, how roles define power and responsibilities and how various hierarchies structure groups and individuals. Not only new norms are introduced by the agents playing a legislative role, but also that ordinary agents create new obligations, prohibitions and permissions concerning specific agents (Boella and van der Torre [2004]).

Regulative Norms

As stated by Boella et al., regulative norms are not categorical, but conditional: they specify all their applicability conditions furthermore legal systems are often modeled using regulative norms, like obligations and permissions. However, a large part of the legal code does not contain prohibitions and permissions, but definitions for classifying the commonsense world under legal categories, like contract, money, property, marriage.

Regulative norms express permission, rights and powers, for example access rights or voting right if you are resident for more than 5 years or born in the city for Luxembourg. Another example is for creating an online library account on the Paris internet site, a parents authorization is necessary if you are under 18 years old.(Caire [2007])

Procedural norms

Procedural norms can be distinguished by two kinds “objective procedural norms are rules which describe how decisions are actually made in political systems; A systems objective procedural norms are a primary determinant of the content of political decisions in that they specify who actually makes decisions, who can try to influence decision makers, what political resources are legitimate and how resources may be used. Subjective procedural norms, on the other hand, are attitudes about the way in which decisions should be made .

1.5 How to Generate Norms in MAS

The following chapter will explain how the mentioned techniques can be combined to build a flexible multi-agent system that employs norm to reach certain mutual goals.

Chapter 2 will describe the employed methodology. Details about the CBR setup and the simulations environment will be explained here. Chapter 3 gives a brief overview over the software design aspects of the implementation and introduce the graphical user interface. Chapter 4 follows the steps of the experimental phase while Chapter 5 compares and analyzes the experimental results. Finally, chapter 6 concludes the work, draws parallels to related works from the area and closes with a section about future work.

2 Norm Generation Methodology

A popular approach to generate norms in multi-agent systems is norm emergence. In norm emergence the agents experiment with the outcome of their actions in an unsupervised manner to gain knowledge about what decision is optimal in any given situation. When - after the simulation is running for some time- the learning process of the agents yields to some common behavior amongst the agents, this is called norm emergence. This paper, on the opposite, is exploring the approach of supervised learning.

The scenario that underlies the supervised learning architecture is an intersection that is populated by car agents, which try to traverse the intersection. The desired resulting set of norms should allow for fluid car traffic with as few as possible collision amongst the traffic participants. In order to achieve this result some architectural building blocks are necessary:

- *Car agents* are the agents that are spawning on the entry points of the intersection and start moving toward their desired destination.
- The *intersection zone* is a square matrix, which contains two intersecting roads. The roads have one lane in each direction and intersect in the middle of the matrix.
- The *norm layer* serves as a communications infrastructure between the environment and the cars. The car agents have access to the norm layer and consult the norms that the norm layer sets for a given square on the intersection.
- The norm generator is an interim piece between the norm layer and the case-based reasoning system. It's purpose is to make allow for more flexibility in

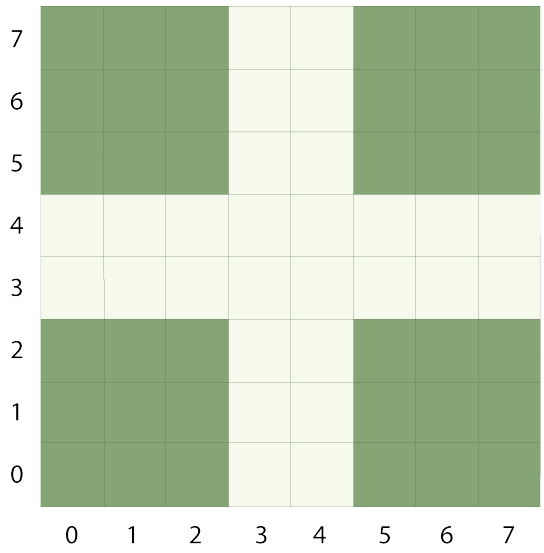

Figure 2.1: Intersection Map

the simulation. This will be discussed in detail later.

- The *case-based reasoning system* observes the current traffic situation and determines the appropriate norms. The norms get then communicated to the norm layer, where they are picked up by the agents.

- The *measuring system* is an additional part of the system that keeps track of the fluidness and general condition of the current traffic. Other components of the architecture, like the CBR subsystem, access the measuring system to gain information of the success of applied norms.

Figure 2.2 shows a structural overview over the components and their interactions.

In the following sections, those different parts of the simulation set up will be explained in detail.

2.1 Agent Behavior

In this scenario the agent behavior is simplified to the basic task of moving from one point to another and acting according to the norms generated by the underlying

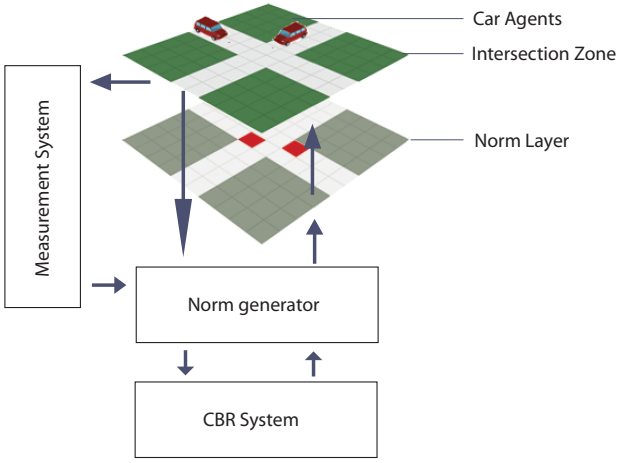

Figure 2.2: Structural Component Overview

norm system. Car agents are getting spawned into the system at one of the entry points, which are located at the beginning of each lane on the intersection scenario.

After one car agent is disposed on an entry point, it chooses a certain exit point as a destination. The choice for the exit point is deliberated by a random function. On every step of the simulation the car agent will try to advance one square into the direction of his chosen destination. The car agent has access to the norm layer, which gives the agent feedback on whether he is allowed to move in this step or not. When the norm layer sets a norm that prohibits the agent to advance, he will remain at his current position until the norm layer changes the norm and he can proceed on its way.

The only way for an agent to leave the simulation is by entering an exit point or when it occupies one square at the same time with another agent. The latter is counted as a collision and the car agents will be removed from the field on the next step.

The car agents themselves disregard one another, they will proceed their way even if another car is straight in front of them or is approaching the same field from the left or the right. Even though a collision might be probable, the responsibility to

avoid them is upon the norm system and its extensions.

The routing of the car is determined by the car itself by following the run of the appropriate lanes. After the car has chosen a destination, it moves in his lane until he reaches the intersection area. Once arrived it chooses the point where it has to turn. When his desired destination requires a right-turn, it does this on the first field of the intersection to stay on the right side of the street. If his desired destination requires a left turn, it will choose its turning point to be on the second field of the intersection. With this behavior, it always obeys the rules of right side traffic.

2.2 Intersection Zone

Since the agents act independent, the agent behavior is to a great part determined by the way the norms are applied to the environment. When analyzing the traffic flow, different zones of the map can be identified by the way agents act. The first area type is comprised of the feeder lanes, where the agents are heading toward the center. On these feeder lanes the only possibility for collisions to occur is when a car in front stops and a following car keeps driving without breaking. In this event the cars eventually will end up on the same field and therefore they will collide. The second area is the intersection area. The situation and interactions between the agents are more sophisticated, since collision might me caused by a far broader spectrum of events. Additionally to the previous condition, where a car in front may break and the following car keeps driving, a collision also may be caused by another car that comes from the sides and runs into a car that is halting on that field or two cars collide while approaching the same field. Another problem that may occur here is the 'deadlocking' of traffic situations, where cars can constellate in a circular traffic situation where only breaking avoids a collisions but by breaking series of other cars behind are forced to break as well which might keep the target field of the first car in the row blocked. Such a situation cannot be resolved until one member of group causes a collision and the traffic can resume to flow. Finally, the third and last area of distinct traffic behavior can be expected in the exit lanes (the lanes that head from the center to the border of the map). Because the end of

the exit lanes will never be blocked, the cars are never forced to break and therefore in these areas no application of norms is necessary.

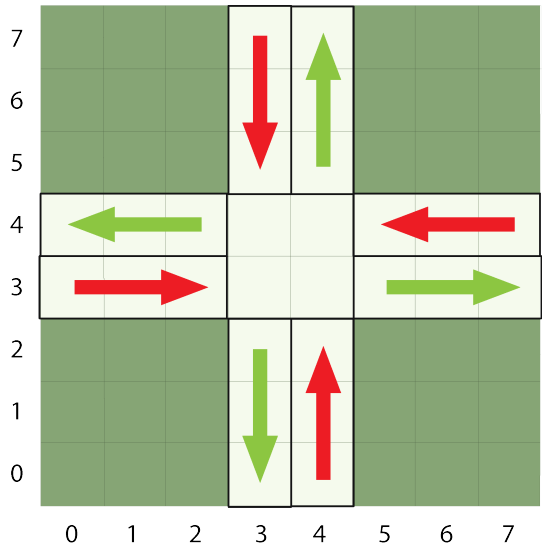

Figure 2.3: Feeder and Exit Lanes

In the search for norms, different methodologies are feasible. One possibility is to always consider the whole traffic situation and search for the optimal norms to apply. This means that every position of every car is taken into account and later a similarity between different scenarios is determined and a set of norms that was applied previously with good outcome can later be reapplied. However, considering the whole traffic situation inflicts a big state space since there are many possible traffic situations.

When applying this approach of a global scope for norm generation, it is important to try to reduce this state space as far as possible and the previously knowledge about the different areas and their characteristics can give useful hints at this task. The following list gives an overview of the distinct map areas.

2.2.1 Feeder Lanes

As mentioned before the feeder lanes display a rather simple traffic behavior. Collisions can occur here, because the end of the feeder lanes lead into the intersection

area. Since it will be required for the norm generation system to temporarily block cars from moving inside of the intersection area in certain situations and therefore backlog may occur. This backlog can lead to the described behavior of rear-end collisions caused by blocks of the feeder lanes. Avoiding these kinds of collisions is trivial: stopping subsequent cars from keep driving when a car in front has ceased to move. This area therefore is of less interest for the research of automatic norm generation, since the optimal norm is obvious. If the control of this part of the traffic can be encapsulated and later ignored by the central norm finding mechanism, the state space of the traffic that is needed to be observed can be greatly reduced.

2.2.2 Exit Lanes

The situation in the exit lanes is even simpler as the situation in the feeder lanes. As opposed to the feeder lanes, where backlogs can cause collisions, the exit lanes lead to the border of the map. Since cars are not blocked from leaving the map no backlogs and hence no collisions may occur. These exit lane areas can therefore be ignored by the norm generation system and hereby the state space for the different traffic situations can be reduced further.

2.2.3 Intersection Area

The intersection area comprises the centric part of the map. In this area, a broad range of traffic situations are prone to occur and finding the optimal policy for fluid traffic management is challenging. As mentioned before, there are two basic types of problems, the avoidance of collisions and as well the avoidance of blockages. While collisions are easier to predict, blockages need some more sophisticated technique to be found and prevented.

On the part of the collisions there are two different atomic situations that may cause collisions. Equal to the situation in the feeder lanes, the car ahead of another car may stop and a following car can crash into its back. The second possibility is when two cars approach the same field from orthogonal directions. While the former(rear-end collisions), can occur almost on the whole map, the latter only can

occur on the four fields of the intersection area. Hence, the fields that are relevant for the prediction and prevention the orthogonal collisions are the fields on the intersection are as well as the fields that might feed cars into the intersection.

The scope for the second problem -circular blockages- is limited to the intersection area (marked in blue above), since the fields of the exit lanes do not block at all and possible blocks on the feeder lane do not affect cars that are already on the intersection. A norm generation system that is able to reliably keep fluid traffic has to be able to observe at least the fields of the intersection (blue) to avoid circular blockags and in addition the spout fields of the feeder lanes (red).

Now that the observation space for the norm generation system is isolated, there are several options on how to extract norms for this sector. The simulation will experiment with two genuinely different approaches for this, the first being the generation of norms that considers the whole observation space as one traffic situation. This approach is referred to as the *global norm generation*.

As opposed to this, there exists another possibility to generate the norms, which is referred to as the *partial norm generation*. In the partial mode, the content of the observation area is not considered as a whole, but is split apart based on the perspectives of the individual cars.

2.3 The Norm Layer

The norm layer serves as a layer between the agents and the CBR system. It holds a certain norm type for the whole map, which either permits or prohibits a car agent to move. The norm types are categorized by two basic types, static and dynamic norms. The dynamic norms are only used in the intersection area, while the static norms are present on the surrounding fields.

Everywhere outside of the central area, the norm layer contains static norms. In the process of applying norms to the norm layer, only the dynamic norms in the center area might be modified by an external component such as the CBR system while the fields with static norms are blocked from external access.

Static norms can either block or allow traffic, depending on its type. On the above depiction, the fields outside of the center painted in green are static norms, that never allow car movements. The ones in grey are static norms that always allow the car agents to move.

The sub division between static and dynamic norms is transparent to the car agent. The agent will only receive a true or false notification for its movement request without having knowledge about the type of the underlying norm.

To address the previously mentioned issue of encapsulation of norm generation in the feeder lane area from the norm generation system, the norm layer may overwrite the existing norms in this area on its own behalf. While dynamic norms in the intersection area passed to the agents as they are, without further reviewing the outcome, the norm system may choose to return another value to an agent that currently resides on a feeder lane. The norm layer therefore is responsible to avoid collisions on the feeder lanes that lead to the center area and therefore liberates the norm generation system from this task. The norm subsystem checks for the car agents that are heading toward the center. If there is a car in front, which is blocked by its current norm, the car behind will also receive a denial notification for a movement request to keep it from moving and avoiding a collision with the car in front. With this technique, no collisions occur on the feeder lanes, the actual norm generation system does not have to cope with this area and the center area gets a constant car supply.

To achieve this kind of behavior the norm layer makes use of a norm template which inhabits one norm for every field. On every round, before any agents starts moving, the norm layer receives a sub with values for the dynamic norms in the center from the norm generation system and sets the values in a new norm map, according to the previously mentioned template. Finally, it checks for the mentioned situations on the feeder lanes to prevent car collisions there and afterwards it is prepared to receive norm queries from the car agents and respond to them.

2.4 Norm Generator

According to the described differences in norm generation -the global and the partial approach- the norm generator serves the function of encapsulating the different requirements and behaviors from the higher layers. The main difference between the two approaches lay within the handling of the underlying CBR system. When utilizing the global approach - those where the CBR system takes the whole traffic situation on the intersection as one single traffic description - and the partial approach - where the CBR system extracts a traffic description for each car individually, based on its current point of view. Both execution methods differ essentially in the way they have to be executed. While the global norm only requires one CBR solution request per step, the partial execution needs to execute as many CBR requests as there are cars currently present in the observation area. The same applies for the evaluation phase of the CBR system, where the global execution mode only requires the evaluation of a single case after every step, while in the partial mode does as many evaluations as solutions have been applied in the previous step.

The second difference in the handling of the two approaches is those of the break points. The break point system has been introduced to aid in the analysis of the CBR behavior and is managed by the norm generation system. The breakpoints are designed to allow pausing the execution of the simulation based on certain criteria. The different types of break points are:

The modification for the activation and deactivation of distinct breakpoints is accessible over a specially designed user interface. (see section 3.4 for details)

The handling of this different kinds of breakpoints also introduces a distinct application logic in the simulation execution.

To allow these different methods to coexist in the same simulation environment the norm generation layer is necessary. Its purpose is to encapsulate the differences between the the execution modes.

2.5 Cased-Based Reasoning System

The CBR system is responsible for finding the optimal norm set. It does so by applying random sets of norms to the intersection area and observing the resulting traffic situation. In this process, it works in close cooperation with the norm layer as well as with the measuring sub system.

The CBR component is designed to allow different modes of execution to allow for the application of both approaches that will be investigated in this work. The CBR system is designed in a flexible way and can easily be configured to work on any subarea of the map. When the component is initialized it takes the relevant parameters and creates the bases for the case storage.

On every turn, the behavior of the CBR system is as follows:

1. The CBR system is supplied with a sample of the relevant area of the map (the *traffic situation*), which contains a set of cars with their positions and directions.

2. The CBR system searches in its case base for an already existing case with a case description that is similar to the given traffic situation. Depending on the current mode, the given sample might be rotated to try to match it to the current situation. [1]

3. At this point two things might happen:

    a. An appropriate case has been found and is passed for further processing. Since every case contains a set of solutions, these solutions are crawled to find the optimal selection to adapt it to the traffic situation

    b. When no appropriate case could be found, a new case is generated and a random solution is applied and associated.

4. After the relevant solution has been selected and patched into the norm layer the car agents will be instructed to move according to their respective norms.

[1] Depending on the current execution mode -global or partial cases- this might affect one single case/solution or several (one for every car).

5. In the next execution round, after all cars have made their moves, the traffic situation is analyzed in terms of occurred collisions. The results are passed to the solutions objects which evaluate their own rating score. [1]

2.5.1 The Process in Detail

The different characteristics of the two execution methods of the simulation require a slightly different handling by the CBR system. While the global mode conceives the central traffic situation as a whole and does not implement ambiguities, one applied solution will always lead to the same outcome, hence it executes in a deterministic manner. This characteristic brings some modification for the global execution mode, which affects the way solutions are generated and evaluated. As for the partial execution mode, the traffic situation is split into smaller overlapping scopes that can occur in different combinations. Applying a solution as part of a greater set of solutions for the whole intersection area can lead to different outcomes every time one and the same solution is applied, depending largely on the solutions that are applied to other cars that reside on a collision course with the other car. This condition and its infliction for the simulation design is explained in detail in the according section for the partial execution mode.

Global Execution Mode

As stated above, considering a traffic situation as a whole brings with it the advantage of non-ambiguous cases. According to Figure 4, the relevant area for the case description consists of 8 fields. Fields on the map in general can have a set of distinct states. They can be empty (no car), there can be a car heading in one of four directions (ignoring the knowledge about the traffic flow) and they can host a set of at least two cars that comprise a collision. The state space for the car directions can greatly be reduced if the knowledge about the car routes is applied. The maximum possible directions a car can have are two for the intersection area and one on the feeder and exit lanes.

Since collided cars will be removed on the next step, they will not influence another

car that approaches the said field on the next step. Hence the collision state can be ignored in order to prepare traffic descriptions for the CBR system and the state space is further reduced.

While examining the four fields in the centric intersection area individually, they allow for three different states: no car, car heading in direction of an exit lane and car heading in direction of another field of the intersection area). The four fields that are the spout fields of the feeder lanes allow for two different states; no car and car heading in direction of an intersection area field.

This constellation allows for $2^4 * 3^4 = 1296$ different states of the observation area as a whole. This number is still very high to form the set of cases for a CBR system, since the performance of the system depends largely on the reuse of made experience.

A second measurement to further reduce the state space and hence the number of possible case descriptions is to implement rotational invariance. If the traffic description can be rotated to match another one, so that the states of the fields are on the same position and their direction as well are rotated accordingly, a different traffic situation can be considered equivalent. The according solution that gave results for the original case will lead to the same results when rotated by the same amount when applied to the rotational equivalent situation. The condition can be exploited by merging one case and its traffic description with its three rotational equivalents and share the gained experience of the applied solutions between them to increase case re-use.

With the combined optimization the state space for the intersection area finally can be reduced to $\frac{2^4*3^4}{4} = 324$ individual combination possibilities.

As stated above, the second crucial component of the CBR architecture is comprised by the solutions. Solutions are sets of norms that can be applied to the norm layer and produce some outcome, which is measured by the number of collisions it caused and number of cars it blocked. When a given solution is applied and results in collision free traffic, it should be evaluated with a good score. Solutions that do produce collision on the other hand, should be marked as improper, so that they won't be applied in future scenarios. Apart from collisions, the number of cars that

is blocked by norms is also an important part of the equation. A solution that blocks all the cars in the intersection reaches the goal of collision minimization but does not allow for traffic to occur. It is desirable to find a solution that evades collisions and blocks as few cars as possible. For this reason the number of blocking norms applied by a solution is part of the evaluation function. Herein the avoidance of collisions is the first priority, after the minimization of block norms as a second. Hence the evaluation function is designed as follows:

$$Evaluation_{global} = -(5 * col + b) \tag{2.1}$$

Where *col* is the number of collisions occurred in the step and b the number of blocking norms. Note that the measuring system (see 2.6) counts a collision between two cars not as one, but two distinct collisions. This is done to distinguish collisions between two and three cars. Hence, a solution that caused a collision will have an evaluation of at least $2 * -5 = -10$. The coefficient 5 therefore is chosen to give *col* 10 times the influence in comparison to b[2]. Since the observation area comprises a eight fields, 10 norms never can be applied and any number of applicated norms is to be prefered over a collision.

This function does not involve any form of history over a set of executions, this is conditioned by the fact the global execution mode works in a deterministic way - a solution that reached a certain score at one point will receive the same score every other time it will be applied.

The total number of possible solutions that can be appended to a case is defined by two factors: the number of cars in the case traffic description and a static constant, that limits the number of solutions to a total maximum. The latter is defined as 5 as a trade off between flexibility (more solutions=greater variance) and length of the learning phase (more solutions to experiment with=longer learning phase).

$$NumberOfPossibleNormCombinations = (2^c)$$

[2]Collisions are calculated for every car separately, the minimum number of collisions per round therefore is two

Where c is the number of cars on the observed area. This function is limited by the solution count maximum, which will be set to five to have a rapid transition out of the learning phase and also allow for a sufficient number of different solutions to find one that is suitable.

How solutions are build

Since the general situation is, that a traffic situation does not have cars that are about to collide, a solution that does not restrict the traffic at all will be the optimum in the most cases. This knowledge is used the tweak the solution generation process by always trying to add and apply such a solution first. When a solution scores $SolutionEval_global = 0$ (this can only be the case with solutions without blocking norms), the case will be closed and end its learning phase. From now on every time the case reoccurs, it will apply the empty solution.

The behavior in situations where no blocking norms are applied is different in the aspect, that the first five times it is applied, it will always generate a new random set of norms. The number of norms also is chosen randomly between $1 \leq norms \leq (numberofcars - 1)$. After it has been executed five times (and hence has five associated solutions), it will choose the very solution with the best score, delete the rest and flags itself as closed.

The simulation employs two different basic phases. The first one is the learning phase, which is the timespan in the course of the simulation where the cases are experimenting with their solutions. After the cases were executed a sufficient number of times, they will end their learning phase and go over to the test phase, where they will exclusively make use of their best known solution. The definition of learning and testing phase in this scenario differs from the common definition in that the phase does not refer to the whole simulation, but to single cases. Hence, many cases can already have reached the testing phase and the system may work in a desirable fashion, but there are still cases that keep experimenting until they gained sufficient experience.

Therefore the transition from the learning phase to the test phase is fluent. While there still are cases in the case base that are not closed, collisions still are prone to happen. This is especially a challenge in relation with cases that occur infrequently.

When a given traffic situation occurs every s steps on average, the case will take $s * maxSolutions$ steps on average to terminate its learning phase.

Partial Execution Mode

The approach for the implementation of the CBR handling for the partial approach is different in a set of aspects. Firstly, the traffic situation will no longer be regarded as a whole, but extract a set of subareas of the map and consider them as distinct traffic situations. After the norm generation has been completed for these situations, a norm map is generated from the results of the partial CBR requests. This inflicts some characteristic changes in comparison to the global norm generation approach, which will be discussed below. The following lists a step-by-step overview over the process:

1. The norm generation system will extract a sub-area of the observation area **for every car** on every round.

2. It will **consider every single of these collected subareas as a distinct traffic situation** and requests solutions for them from the CBR system.

3. After the CBR system responded to every request (i.e. responses are single norms), the norm generation system will build a map of norms and apply it to a norm map. (The global approach does not have to do this, since the output of the CBR system already is the norm map in the required size)

4. A reference to the used case solutions in the CBR system is stored, for later evaluation.

5. After the norm generation system finishes its process, the car agents receive clearance to make their moves, according to the norm map settings.

6. After cars have moved, the observation area is checked for collisions. When the cars involved in collisions have been identified, the solutions stored in the reference list (see step 4) that were involved in the norm generation for these cars get evaluated accordingly. The case solutions that did not produce a collisions get evaluated as well, but in a positive manner. (The difference to

the global approach in this step is that multiple case solutions are involved, not just a single one.)

As can be seen in the list, the main difference between the handling of the global and the partial approach are the different cardinalities between steps and norm requests. While the global approach has a $1:1$ cardinality, the partial approach has a $1:n$ cardinality with n being the number of cars. The modifications to the design of the partial approach to handle these difference are as follows:

- Every step requires n CBR requests instead of one.
- Every step requires n evaluations for the applied case solutions, instead of one.
- The norm generation system has to construct a map of the norm results
- The application of this norm map is not deterministic anymore.
- Case solutions have to consider this non-determinism in their evaluation function, i.e. they have to consider a number of past experience (instead of only the last execution in the global approach.
- The size of the traffic description has to change according to the scope of a car.

2.5.2 Traffic Description

The choice of the point-of-view style design in the partial approach has been made to emulate the natural way cars in traffic behave. We assume here, that there is a driver inside of a car agent that looks in driving direction and has a certain range of sight. The exact range of sight which combines simplicity and functionality will be subject of the experimentation phase. It is therefore required that the architecture for the traffic description allows for easy changes in the experimentation phase.

To allow this to happen the system will employ a two component approach which consist of the traffic description and a range of sight mask. The traffic description,

or also traffic situation, refers to the current population of agents and their position on the map. A traffic description contains the positions and directions of these car agents. The size of the traffic description is defined as a fixed matrix TD with the size of $5 * 2$ entries.

$$TD = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \end{pmatrix}$$

$$\text{where } a_{ij} \in C \cup \emptyset$$

with C being the set of cars in the simulation. When executing, the norm generation system fills this matrix with values for the cars in the 5*2 fields that lay in front of the car.

The second component is the range of view mask M.

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} & m_{15} \\ m_{21} & m_{22} & m_{23} & m_{24} & m_{25} \end{pmatrix}$$

$$\text{where } m_{ij} \in \{0, 1\}$$

Before a simulation is executed, M will be initialized with a set of values 0 or 1, where 0 will cause the system to remove the related element in TD, which will lead the CBR system to simply ignore this field and only consider the fields of TD where the according value in M is 1.

$$TD' = mask(TD, M)$$

Since the matrices now don't have the same height as width anymore, the rotation mechanism of the global approach cannot be applied anymore. Instead, the traffic description will be rotated *before* they are converted into traffic description with orientation to north. In this process, the directions of the cars in the matrix have to be rotated as well to maintain relative direction to the viewpoint of the car.

Cars in C can have one of four states, which are determined by their directions. Another state for the field can be its emptiness, summing the total number of possible field states to five. Equally to the global approach the collision state can be ignored for the named reasons. The complexity of the state space of TD' is therefore defined as

$$complexity_{TD'} = 5^{mf}$$

With mf being the number of unmasked fields in M (i.e. containing 1).

This approach renders the rotation of cases in the CBR process unnecessary, since the cases are already equally oriented before they enter the CBR system.

2.5.3 Solution Generation

The situation in the partial approach is different to the global approach in terms of norm generation by the fact that the CBR system no longer responds with a complete map of norms. Every request of a solution in the partial approach contains only a single norm, which have to be combined into a single norm map that fits the input of the norm layer. The received norms are placed on a norm map template on the spots where the cars are residing that were the base of the traffic description generation. The system therefore resembles more of a knowledge base for the single car agents. While a solution request before gave a result that is best for the whole map, the system now outputs solutions that are best for the individual cars. A car agent can expect that his assigned norm on the norm layer (which is the only thing it sees) reflects the best decision it can make in any given situation. The optimizations from the global approach also are not applicable here. The CBR system will only return a single norm in the partial mode; hence there are only two possible solutions for every case: a blocking norm or a non-blocking norm. To account for this fact, the process on the CBR side is designed as follows:

1. The CBR system receives the traffic description from the norm generation system

2. The CBR system searches for a matching case but does not rotate the cases like in the global approach.

    a. If a case has been found, select it.

    b. If no case has been found, a new case is created **and two solutions are created, one with a blocking and one with a non-blocking norm.**

3. Scan the selected case for the solution with the highest score and return it to the norm system.

As mentioned, the non-determinism brings with it different requirements for the evaluation of solutions. Since - unlike in the deterministic case - we can no longer assume that the application of one solution always leads to the same result, the evaluation function should be able to consider some knowledge about a greater set of executions. The according evaluation function is defined as follows:

$$Evaluation'_{partial} = \frac{((averageScore * ac) + currentScore)}{ac}$$

With *ac* being the total *application count*. *score* is generated as described in equation (2.1). This allows the solution selection function to choose the one that has performed best over the entire course of the simulation. However, this function has a severe impact on the performance of the entire process that it will be subject of experimentation in the experimentation section.

2.6 Measuring System

The key metric for the simulation is number of collisions. The performance of the two different approaches as well as variations of them will be judged by the number of collided cars over time. Since the simulation's performance will rise over time, a sliding window metric is to prefer over a total count of collisions metric. For this reason a metrics/measurement system will be implemented whose task will be to

keep track of this metric. It will record the Δ of collisions in every step and sum them up for the last n steps.

Along with the collisions other interesting metrics are those of the total number of blocking norms applied. Furthermore, an indicator for the status of the CBR system for the the total number of cases and solutions residing in the case base. Since the latter two are not continuously rising they will not relay on the sliding window system but will always reflect the current state.

The mentioned four values will be supplied to the runtime environment of *Repast Simphony*, where they will be picked up and reused in two ways. *Repast Simphony* can be configured to access this data in terms of a data set. Later, the runtime environments interface allows the attachment of other elements to a data set. In this scenario, the data set is used to display a constantly updated chart and a file output. The data in the file output will be used to generate diagrams and tables for the analysis in the experimentation phase.

3 Software Design

The described components from the methodology chapter reflect the architecture of the simulation environment. The architecture contains a package for every mentioned building block: the norm subsystem (norm layer and the two different norm generators), the CBR subsystem, the metrics subsystem and several others that will be introduced in this chapter.

3.1 Package Overview

Figure 3.1 gives an overview of the package structure. The base package for the simulation is the intersectionJ package, with a row of subordinate packages. Their contents are as follows:

- intersectionJ: general utility classes, like the matrix class that is used throughout the project, the agent classes and several support classes for the interaction with Repast Simphony.
- norms: The static and dynamic norms, the two norm generators (global and partial norm generator) the norm layer and several helper classes for the norm generation.
- cbr: The contents of this package are made up by the four key elements of the cbr system (case base, case, case description and case solution) and a caseSolver, which is used for random norm generation in the global approach.
- test: jUnit test cases that were used in the development phase.
- gui: the traceFrame class which contains the user interface.

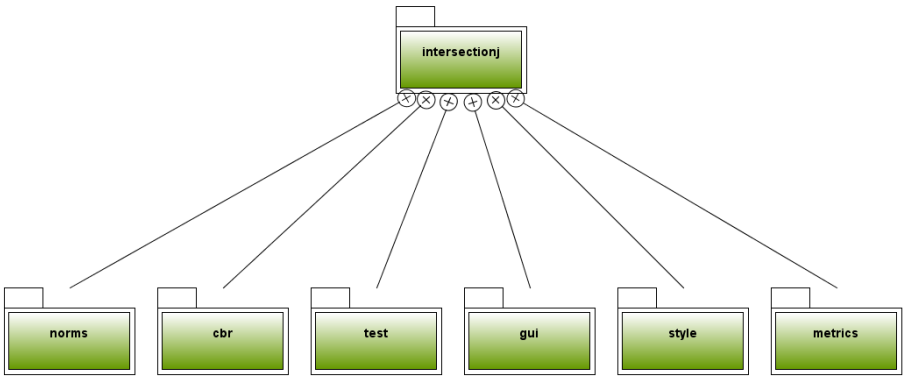

Figure 3.1: Package Diagram

- style: this package contains classes that are used to configure the visual output generator of repast.
- metrics: the slidingWindowMetric class for the measurement of simulation performance.

3.2 Core Classes

To get an overview of the relations between the core classes, see diagram 3.2

The architecture of the simulation environment is too complicated to implement via the GUI-editor of Repast Simphony; therefore, the manual mode will be used for the implementation. The manual mode allows the interaction with the *Repast Simphony* runtime environment by supplying a user-defined set of Java (or groovy) classes. For the interaction a custom *context generator* has to be provided that creates a context with the necessary components. In automatic mode this would have been done entirely by the Repast Simphony runtime environment based on a configuration file (model.score). With repast comes a set of predefined components that can be used in the context creation. The most important ones are the predefined perspectives. The intersection simulation does not require the use of

continuous coordinates since it uses a grid with discreet (integer) coordinates. The internal coordinates therefore will be mapped to a simple *grid* perspective.

The second important modification to the context builder is the creation of agents. The simulation setup requires a distinct handling for the creation of car agents and the orchestration of the partial norm retrieval phases on every step. The component that does all this handling is the scene manager *FSM*. Enabling this class to receive a call on every simulation step before the agents are able to move requires a special set up, since only agents are being called by the Simphony scheduler. When employing the scheduler there can be unpredictable variations on which agent gets called first on every step. Even though the scheduler allows some control over the priorization of agent calls by adding scheduler annotations, it is preferable to handle agent execution manually to always assure that the norm layer is ready before the first agent moves. For this reason, the FSM class is masked as an agent and added to the context at initialization. Later it will create agents randomly and giving them the clearance for movement manually, when the norm generation procedure has terminated.

As a last component, the context is supplied with a value layer. Value layers in Simphony serve the purpose of colorizing the map. This is used in the simulation to graphically highlight certain fields on the map. The applied value layer is modified by classes of the style package, which colors fields of the map in relation to the norm that is currently active. The color codes are white for non-blocking norms, red for (dynamic) blocking norms and green for static blocking norms. The final result of this color coding can be seen in figure 4.1.

The two important components that are used by the FSM component are the *CarMap* class and an implementation of the *NormGenerator* interface. The *CarMap* is a helper class that encapsulates useful function to gain aggregated information about the current traffic situation. The creation of new car agents and their placement on the feeder lanes is also handled by the FSM.

Figure 3.2: Core classes.

3.3 Norm and CBR subsystems

In figure 3.3 the connection between the core class architecture (figure 3.2) and the norm generation subsystem with its CBR appendage becomes clear. The two implementations of the *NormMapGenerator* interface (*GlobalNormMapGenerator* for the global approach and *CarLinkNormMapGenerator* for the partial approach) handle the calls to the CBR system[1] independently. Between the *CaseBase* and the *Case* class consists a one to many cardinality. The individual cases contain a single *CaseDescription* and maintain a one to many cardinality to a set of *CaseSolution*s

3.4 User Interface

The design of the graphical user interface is planned to give analysis information on the state of the CBR system.

3.4.1 Case Base Access

The contents of the case base will be displayed in the user interface in a tree structure (a *JTree*) with a "CaseBase -▷ Cases -▷ Solutions" hierarchy. Selecting one item in the tree outputs the return value of its *toString* method in a text field on the right of the interface. The information given for the case contents are its name, it's state (states may be 'open' or 'closed', see 4.1 for an explication), the number of times the case has been applied to traffic situations and a string representation of its assigned case description. The string representation of the case description are 4x4 (respectively 5x2 in partial mode) lines of characters, with '*' representing an empty field and four directional characters to represent car directions, if a car agent is present on a field ('∧' for up/north, '>' for right/east, '∨' for down/south' and '<' for left/west).

When a case solution is selected, the output in the text field contains its name, the number of applications and also a representation of its contents. In the global

[1] The CBR system is incorporated by the *CaseBase* class

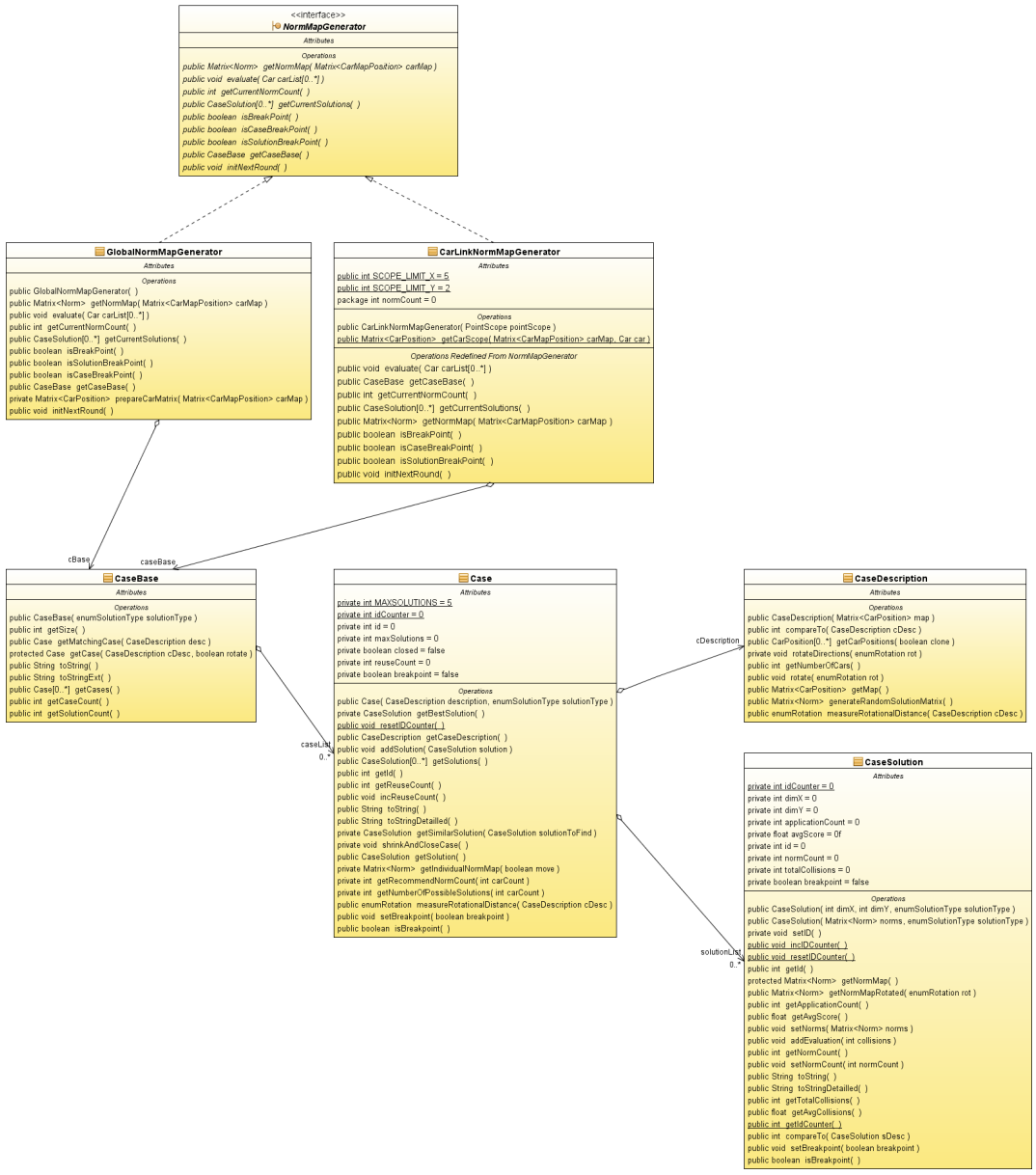

Figure 3.3: Relation between *norm* and *cbr* packages.

```
Global:         Case 71 (Closed)
                Appl:91
                Matrix plot 'CaseDesc71'(inverse):
                ****
                *v<*
                >***
                ****

Partial:        Case123
                Appl:42
                Matrix plot
                'CaseDesc123'(inverse):
                **^**
                *>***
```

Figure 3.4: Example case tracer output.

approach the solution is represented by a 4x4 matrix, encoded with '+' for a non-blocking and '-' for a blocking norm. In partial mode the solutions contain only a single field with the same character encoding. See figure 3.6 for an example.

3.4.2 Breakpoint Control

Another function of the graphical user interface is the control of breakpoints. It allows to set breakpoints of two different types; object related and collision related. The object related breakpoint cause the simulation to halt, when either a case or a certain solution gets applied in the current step. After the solution has halted, the interface gets the focus and automatically selects the respective object in the tree.

The second type of breakpoint, the collision breakpoint, halts the simulation on every collision that occurs. This is useful in the later phase of the simulation, where collisions occur rarely and a step by step search for them would be time-consuming. Similar to the object breakpoint, the activation causes the simulation to halt, focus the interface and select the applied object. In this context it becomes important to know, which solution was applied in the previous step when the system failed to prevent the collision. For this reason the 'Prev Solution' button was added which

```
Global:         Solution ID=120
                apps:86 avgScore:-2.0
                Matrix plot 'normInstance'(inverse):
                N++N
                +--+
                ++++
                N++N

Partial:        CaseSolution123
                Appl:42
                Matrix plot
                'CaseSolution123'(inverse):
                +
```

Figure 3.5: Example case solution output.

exposes the culpable solution in the tree.

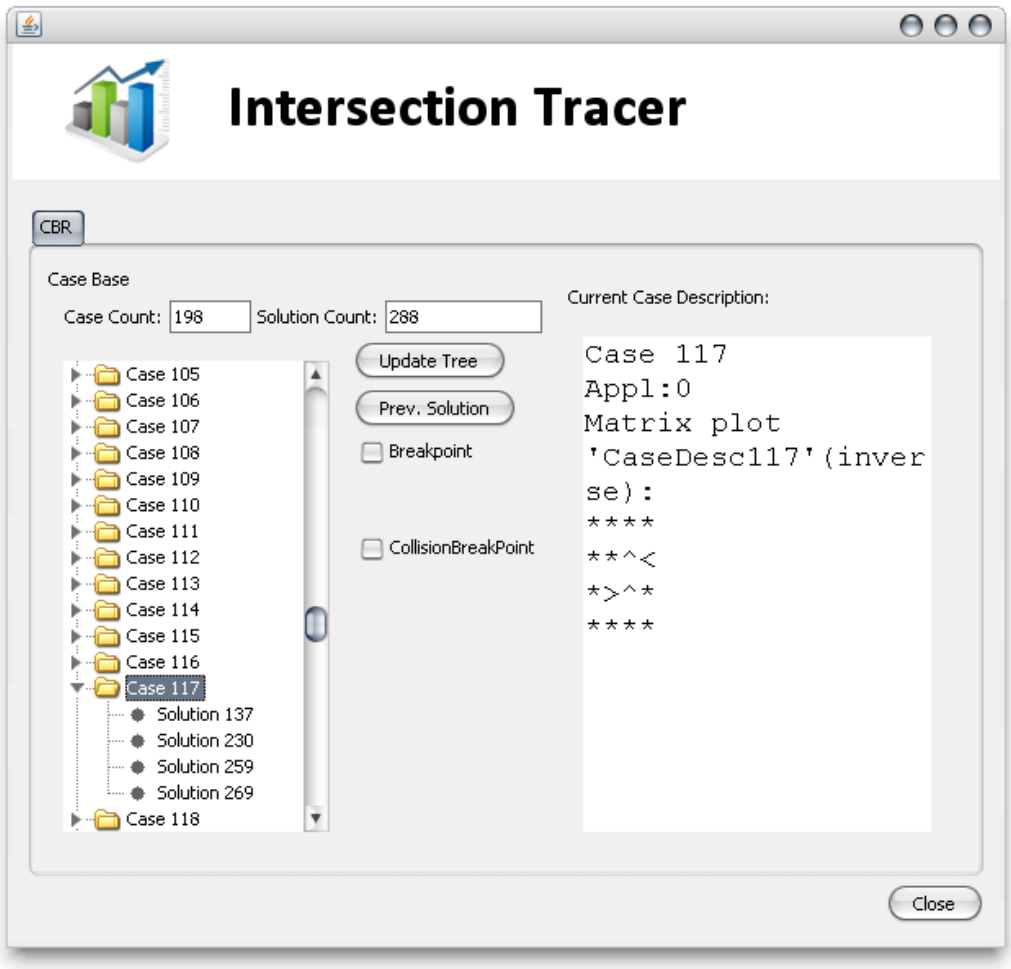

Figure 3.6: The *Intersection Tracer* interface.

4 Experiments

This chapter gives an overview of the experiments that were done on the simulation setup. The methodology described in chapter 2 represents to a great part the final simulation structure, while many of the design decisions were based on experiments that suggested the modification of parts of the simulation architecture.

The addition of functionality happened in the following order:

1. Phase 1: Rotation when searching for cases to reduce number of cases.
2. Phase 2: Ignoring collisions in case description.
3. Phase 3: Consider car directions
4. Phase 4: Closing of cases

The following sections will analyze the simulation behavior for the different approaches based on the definitions in chapter 2. The first section starts with the description of a simulation execution in the global mode and discusses important features in the metrics diagram. The subsequent section will display the characteristics of the simulation in partial mode and experiment with different scopes for the car agents.

4.1 Global Approach

The big scope of the traffic descriptions in the global approach has the ability to work with patterns in a greater area than those in the partial approach. Its architecture is designed to handle complex car constellations and resolve interdependencies.

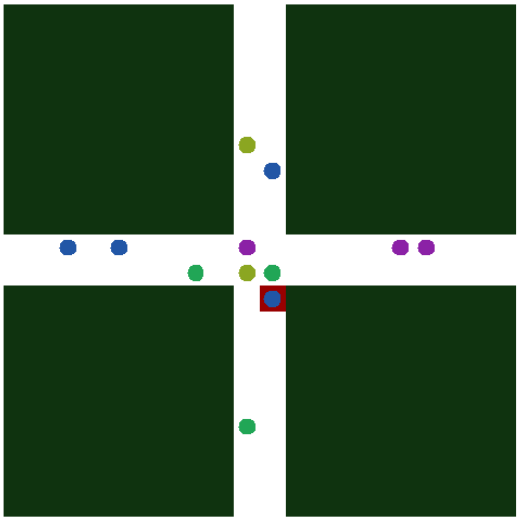

Figure 4.1: Example traffic situation in global mode

Figure 4.1 shows a sample traffic situation from the learning phase which will be used to demonstrate the occurrences inside the case base.

The chosen figure displays a rather tricky situation. The only way to prevent a collision is to block the car that is approaching into the intersection from the lower feeder lane (blue). The shown situation was captured at around step 460.000 in the course of the simulation. This was the third time this traffic situation happened in total. Please refer to figure 4.2 to see the contents of the case base to get information about the car directions and the representation of this situation in the case base.

It can be seen that the algorithm was able to find the best possible solution for the problem on this third step. Since the length of the learning phase per case is set to five, the case will occur two more times, apply random solutions and finally will close itself and chose solutions number 3 with id 409 as the standard solution for this specific traffic situation. Please remember that situations that are rotational equivalent are considered the same case and will apply a rotated version of the same solution.

The mentioned case is one of the rarer ones, since it happens roughly around every 150.000 steps. Given the five solutions limit, this case needs 750.000 steps to terminate it's learning phase. While this might be an acceptable value, an

observation of the contents of the case base after 3 million steps shows, that there are still many cases, that just have a single solution applied. This means that their traffic situations might occur every 1.5 million or more steps on a average.

Table 4.1 contains an overview of the case base status in the test simulation. The first four lines of the table list the overall number of open and closed cased as well as the total number of solutions. An open case is one, that is still in the phase of experimenting with different solutions. After a case has been called five times, it will chose the best solution and converts itself to a closed case (i.e. its learning phase has terminated and it will not experiment any longer, but always will apply the best known solution). Therefore the 'Cases' are an aggregation of open and closed cases ($|Cases| = |Cases_{open}| + |Cases_{closed}|$). *Solution* contains the total number of solutions, where $|Cases_{closed}|$ of them are the only ones assigned to one of the closed cases. The values in row 6 gives better insight over the solution distribution of the open cases. The columns contain the number of open cases with n assigned solutions. The first column '1 sol.' states that there is a total of 40 open cases in the case base, that only have one assigned solution. This indicates that the respective traffic situations for the cases are occurring rarely, but will also convert into closed cases, given a sufficient amount of time to execute. The second column '2 sol.' contains the number of cases with 2 solutions and so forth.

The total number of solutions therefore comprised of

$$|Solutions| = |Cases_{closed}| + \sum_{i=1}^{i=n} |Cases_i| * i = 387$$

where $Cases_i$ is the number of open cases with i assigned solutions.

Rows 7-12 of Table 4.1 analyze the cases that are closed. The values for the indicated attributes are split up by the number of cars the relevant case has. The first value column contains the number of cases that contain 2 cars in total, the second 3 and so on. Row 8 (cases with impending collisions) are cases that contain a traffic situation where a collision are imminent, if no norms blocking are applied. Line 9 shows the numbers of the above situations, where the norm generation system was able to find an appropriate solution to avoid the collision. As can be seen, the

row	attribute	values				
1	Cases	326				
2	Cases (Open)	74				
3	Cases (Closed)	252				
4	Solutions	387				
5	Open cases with n solutions	1 sol.	2 sol.	3 sol.	4 sol.	5 sol.
6		40	17	9	6	2
7	Closed cases with	2 cars	3 cars	4 cars	5 cars	6 cars
8	with impending collision	3	12	34	1	
9	Resolved collisions	3	12	31	1	
10	Unresolved collisions	0	0	3	0	
11	Used unnesessary norms	0	0	7	0	
12	Involve circular relations	0	1	7	0	

Table 4.1: Case status after 3 million steps (global mode)

system was able to find adequate solutions for 47 out of 50 problematic cases. Also, it is evident that the system seems to have problems with resolving 4-car-situations. This underlines the fact, that the difficulty of resolving traffic situations raises with the number of car agents involved.

Row 11 indicates the number of cases, where more solutions were applied than necessary (e.g. a car was blocked, that was not involved in an impending collision). Again, cases with fewer cars were more accurate with the appliance of blocking norms. They found for all the cases the perfect solution with minimal blocks and minimum collisions. Row 12 contains the number of cases, where circular relations were involved. This means that a misplaced norm can block the whole intersection area or cause a row of subsequent collisions. This row also points out that traffic situation with 4 or more cars get much more complicated to solve, since traffic situation with circular relations occur on a higher frequency.

4.2 Partial Approach

The experimentation with the partial approach began with a mid-size scope of the three fields in front of the car (see formula 4.1 for the masking matrix). The idea is to see, whether these three fields are sufficient to keep traffic fluid. In the negative

case the mask ought to be enlarged, otherwise reduced to the minimum that is possible.

$$M_{front3} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \tag{4.1}$$

I expected this three field approach to be inferior to the global approach due to the fact that more complex traffic situation were not captured as a whole. Surprisingly, the partial approach turned out to perform extraordinary good, displaying a steep learning curve. The first three field mode led to the total avoidance of collisions just after about 15.000 steps. Indeed, in several test runs the total number of collisions was always between 20 and 35 before the 20.000 step mark was reached and stood this way until the tests were aborted at 1.000.000 steps.

It could be observed that a certain structure of behavior had been emerged from the cases based learning process. From a total of 31 cases after 1 million steps, there where only 6 that favored a blocking norm over a non-blocking norm. Five of those had one thing in common, the square in the front left of the car position was occupied by a car heading (relatively) eastwards - in the direction of the field the current car is steering. The system had by established a 'priority to left' rule.

The test was repeated several times to see, whether it was just chance and that a 'priority to right' rule might emerge as well, but in ten test runs always the same occurred with minor variations. It is interesting that - unlike to real world traffic systems, where driving on one side of the road usually comes along with a priority to other participants approaching from the very same side - a 'cross over' priority emerged.

Seeing the results of this norm-set in action reveals an advantage of the 'cross-over' priority: cars that come from the left are those that are already on the intersection area. By letting them pass, the intersection area stays virtually blocking free, since the occupying cars are always given priority to exit.

After observing that only the left field directly in front of the car seems to have an influence on the norm generation, the logical consequence was to shrink the mask

to only this single field and let the system do a test run.

Indeed, the system worked as expected. The only difference was that the total number of collision over one million was now one, instead of 20 and the learning process was terminated after 25 steps. The system ran stable and did not result in any other collision until the test was canceled at 1M steps.

```
Case:           Case 191
                Appl:3
                Matrix plot 'CaseDesc191'(inverse):
                ****
                *v**
                *>^*
                **^*

Solution 1:     Solution ID=272
                apps:1 avgScore:-10.0
                Matrix plot 'normInstance'(inverse):
                N++N
                ++++
                ++++
                N++N

Solution 2:     Solution ID=377
                apps:1 avgScore:-13.0
                Matrix plot 'normInstance'(inverse):
                N++N
                +-++
                ++-+
                N+-N

Solution 3:     Solution ID=409
                apps:1 avgScore:-1.0
                Matrix plot 'normInstance'(inverse):
                N++N
                ++++
                ++++
                N+-N
```

Figure 4.2: Example Case Base Content in Global Mode.

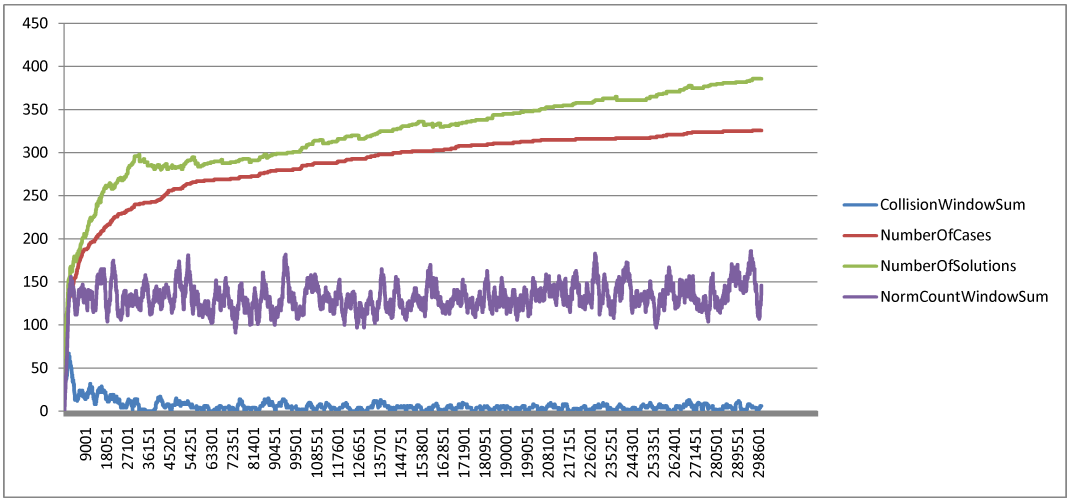

Figure 4.3: Simulation data for the global approach.

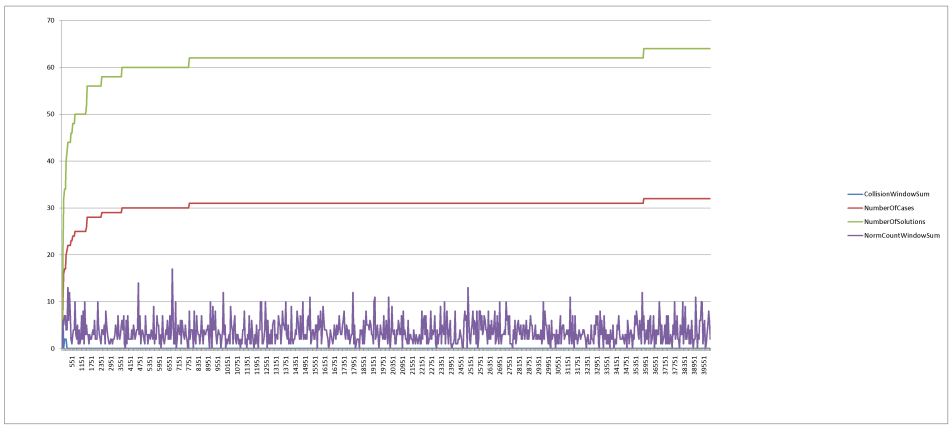

Figure 4.4: Simulation data for the partial approach.

5 Results

The experimental setup employed two generally different approaches to investigate the possibilities for the application of case-based reasoning for norm generation. For both approaches the same objectives were given: optimizing traffic flow by avoiding collisions and minimize the times cars are impeded on the way to their target. The simulation was repeated several times for each scenario to assure permanence of observed characteristics.

The first approach is based on a global traffic observation and employed a deterministic behavior with non-ambivalent case descriptions. Because of the deterministic nature of this approach, the solution evaluation did not keep track of any experience except the very last one. This approach was tailored especially to be able to cope with complex traffic situations, that may or may not include circular relations.

The second approach, which employs a partial recognition of the traffic situation, oriented at the view of the car agents is non-deterministic and therefore employs a memory for the outcome of solutions based on the total experience that was gained with any solution.

From the case-based reasoning perspective the main differences between the two approaches is the size of the case description. The case description utilized by the partial approach is far simpler than the one used by the global approach. Table 5.1 compares the state space complexity for the different approaches. As can be seen, the 3 and 1 point scoped partial complexities are far smaller compared to the global approach. Along with the the higher complexity of the latter comes the fact, that certain traffic situations occur sporadic. The solution determination process requires every case to be applied 5 times before the learning phase terminates which resulted in a slowly converging learning phase.

	global	partial (3 point mask)	partial (1 point mask)
state space complexity	324	32	2

Table 5.1: Comparison of State Space for Case Descriptions

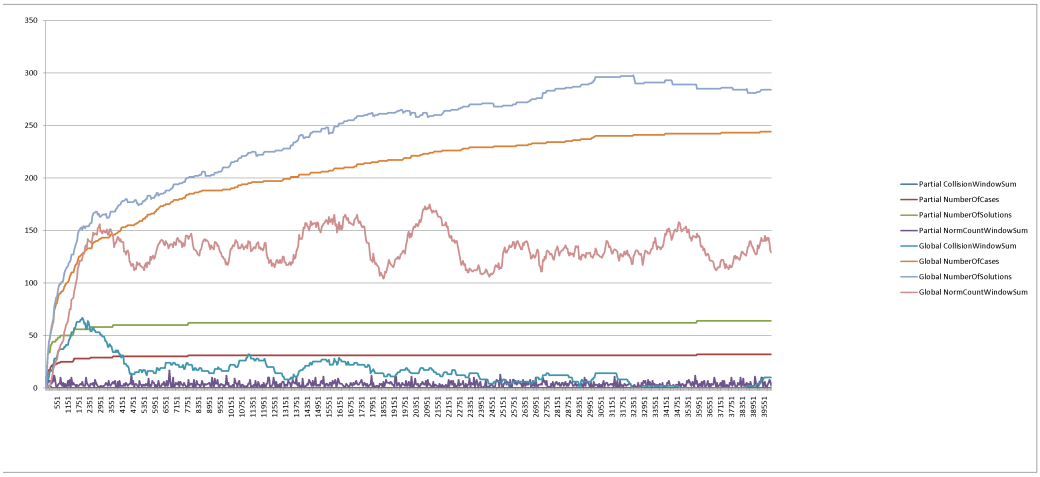

Figure 5.1: Runtime comparision of global and partial approach.

The partial approach -even though considering a far smaller area the map- was able to converge almost instantly and completely avoid collisions after the learning phase was terminated. Results from the simulation execution showed a noticeable pattern which could be used to optimized the scope to a single field approach. The implementation based on this observation enabled further optimization of the considered scope and brought an additional reduction of the convergence from from learning to application phase. Figure 5.1 contrasts the course of the simulation and the key indicators.

6 Related Work

The task of adaptation is usually assigned to agent. Agents that share the same habitat have to find ways to interact in a socially compliant manner that minimize their impact on their environment and does not harm the mutual goals of the MAS. Also, they are responsible to reorganize when required to adapt to changes in their assigned tasks or environment.

Based on this idea, some works in the area are the following. Excelente-Toledo and Jennings (Excelente-Toledo [2003]) propose in their puplication a decision making framework that enables agents to dynamically select the coordination mechanism that is most appropriate to their circumstances. Hbner et al. Hübner et al. [2004] propose a model for controlling adaptation by using the MOISE+ organization model, and Gteau et al. Gâteau et al. [2005] propose MOISEInst as an extension of MOISE+ as an institution organization specification of the rights and duties of agents roles. In both models agents adapt their MAS organization to both environmental changes and their own goals.

The stated works differ in the point that the reorganization is handled by the agents itself. In this work, the organization and reorganization of agents is handled by the norm generation system which takes responsibility for the coordination of the individual agents

6.1 Conclusions

Chapter one gave an introduction to the underlaying concepts and technologies employed in this work. Chapter two discussed the details of the methodology that was employed, chapter 3 described the software design utilizing Repast Simphony and Chapters 4 and 5 analyzed the observations made by execution the developed simulation scenario.

This work tried to investigate the possibilities of norm application to a dynamic multi-agent system by a specially tailored case-based reasoning approach. In the course of the investigation process many interesting observation could be made and it became clear the performance of the application of a CBR approach relies heavily on the adjustment of key factors like case complexity and reuse.

The size of the case base is directly related to the case complexity. A high number of possible case descriptions enlarges the timespan required to end the learning phase and enter the test phase, given that no appropriate similarity function is employed. The learning rate of a CBR system is strongly coupled to the case reuse rate, which could well be observed in the course of the experimental phase.

Eventually can be stated that both, the global as well as the partial approaches, performed good on their task to find optimal solutions to conflicting traffic situations.

The global approach was able to find optimal solutions for almost every conflicting situation where 3 or less cars were involved and only showed weaknesses with situations involving more (mostly 4) cars. In situations were circular relations were involved it managed in most of the cases to resolve the conflict without causing intersection blockages.

The partial approach performed excellent on the task of traffic optimization with a minimal set of case complexity and rapid learning phase termination.

Concluding it can be stated that a CBR approach is capable of managing the task of norm application in conflict prone scenarios like the given intersection environment. The approach bears a big area of possible extensions and adjustments that might

be able to transfer this approach to more complex multi-agent system.

6.2 Future Work

During the experimentation phase a set possible optimizations and extensions for the used architecture became evident. It would be interesting to experiment with the developed powerful, flexible and well observable framework on more complex scenarios and compare the results with the ones gained in this work. The most critical part of the application to more complex systems is the design of key components like the similarity and evaluation function as well as the case description and solution design.

One interesting source of information would be a comparison between the employed techniques with a solution utilizing MARL (Multi-Agent Reinforcement Learning) to retrieve insight about the differences in terms of the learning phase duration and traffic performance in the given scenario.

The big differences in performance that where observed during the different subphases of the experimentation phase gave an insight about the importance of mentioned key factors. The big number of possible adjustments suggest the appliance of techniques like genetic algorithms. Genetic algorithms could be used to explore a wider set of settings automatically or fine tune them. Tasks like the automated search for the best possible masking matrix for case description in the partial approach in this direction is a subject of future work.

Bibliography

Repast simphony official reference, August 2009. http://repast.sourceforge.net/docs/Reference.pdf.

Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun.*, 7(1):39–59, 1994.

G. Boella and L. van der Torre. Constitutive norms in the design of normative multiagent systems. In *Computational Logic in Multi-Agent Systems, 6th International Workshop (CLIMA VI)*, volume 3900. 3900 of *LNCS*, page 303319. Springer, 2006.

Guido Boella and Leendert van der Torre. Regulative and constitutive norms in normative multiagent systems. In *IN PROCS. OF KR04*, pages 255–265. AAAI Press, 2004.

Guido Boella, Leendert van der Torre, and Harko Verhagen. Introduction to normative multiagent systems. In Guido Boella, Leon van der Torre, and Harko Verhagen, editors, *Normative Multi-agent Systems*, number 07122 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL http://drops.dagstuhl.de/opus/volltexte/2007/918.

Patrice Caire. A normative multi-agent systems approach to the use of conviviality for digital cities. In *Normative Multi-agent Systems*, 2007.

C.B. Excelente-Toledo. The dynamic selection of coordination mechanisms in multi-agent systems. February 2003. URL `http://eprints.ecs.soton.ac.uk/7814/`.

Benjamin Gâteau, Olivier Boissier, Djamel Khadraoui, and Eric Dubois. Moiseinst: An organizational model for specifying rights and duties of autonomous agents. In *EUMAS*, pages 484–485, 2005.

David Hales. *Multi-agent-based simulation III : 4th international workshop, MABS 2003, Melbourne, Australia, July 14, 2003 : revised papers*. Springer, Berlin ; New York, 2003. MABS 2003 (2003 : Melbourne, Vic.) David Hales ... [et al.]. (eds.). ill. ; 24 cm. Lecture notes in computer science ; 2927. Lecture notes in artificial intelligence.

Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. Using the moise+ for a cooperative framework of mas reorganisation. In *SBIA*, pages 506–515, 2004.

Gloria T. Lau, Kincho H. Law, and Gio Wiederhold. Analyzing government regulations using structural and domain information. *Computer*, 38(12):70–76, 2005. ISSN 0018-9162. doi: http://dx.doi.org/10.1109/MC.2005.397.