Some Aspects of Distributed Machine Intelligence Algorithms

Borbála Katalin Benkő
Supervisor: Sándor Imre, DSc

2012
Abstract

The goal of this dissertation was to discuss how novel, distributed, autonomic, collective machine intelligence mechanisms such as knowledge sharing, emergent self-organization and knowledge self-revision can be utilized in particular fields, and bring impressive results in means of adaptivity and performance. I extended existing algorithms, applied them to new fields, and often also utilized their emergent effects.

The dissertation summarizes my research conducted in three particular fields. The connection point between the fields is the utilization of some kind of novel, distributed and autonomic machine intelligence based approach.

In the first chapter I discuss the results achieved in the fields of open collective-adaptive multi-agent systems. I synthesized a novel approach (OLAKO) for highly dynamic environmental requirements by combining (i) online learning autonomous agents, (ii) society-wide emergent pair-wise knowledge sharing between the agents and (iii) a knowledge self-optimization mechanism to optimize and clean the knowledge. I showed that the system enables high adaptivity, knowledge sharing spares time for the individuals, and the knowledge self-optimization speeds up the learning curve for the agents.

In the second chapter I discuss the results achieved in the field of on-demand clustering in self-organizing mobile networks. I created a new on-demand clustering algorithm (Shuffling ODC) by extending the baseline On-Demand Clustering algorithm developed at BT Labs. I showed that Shuffling ODC overcomes ODC’s topological starvation problem, results in higher clusters and more clustered nodes and better load balancing performance without breaking the principle of locality, and without significant communicational overhead, compared to the baseline algorithm.

In the third chapter I discuss my results in the field of sentiment mining from natural language texts. I defined the problem of named entity centric opinion mining, I proposed a general, language independent, adaptive algorithm for the task. I showed with experiments that the model produces high precision and recall. Above that, I created an unsupervised self-analysis methodology, including a self-revision algorithm for the term dictionary, a completeness checker for the (non-annotated) test corpus and the term database, and an approximation for precision and recall of an evidence based dictionary extension.
# Table of Contents

Abstract ................................................................................................................................. i

Table of Contents .................................................................................................................... ii

Preface ...................................................................................................................................... 1

Chapter I. OLAKO: An Approach for an Open Collective-Adaptive System .................. 3
  1. Introduction and Background .......................................................................................... 4
  2. Problem Space and Terminology .................................................................................... 5
      2.1. The Showcase World .............................................................................................. 7
  3. The OLAKO Model .......................................................................................................... 8
      3.1. Online Learning ....................................................................................................... 9
      3.2. Knowledge Sharing ................................................................................................ 13
      3.3. Knowledge Self-Optimization ............................................................................... 17
      3.4. Adaptation to Rule Changes .................................................................................. 21
  4. Formal framework ........................................................................................................... 22
      4.1. Basic Elements ....................................................................................................... 22
      4.2. Success Probabilities ............................................................................................. 23
      4.3. History Based Approximation ............................................................................... 25
  5. Experimental Evaluation ................................................................................................. 26
      5.1. Simulation Setup .................................................................................................... 26
      5.2. Online Learning ..................................................................................................... 27
      5.3. Knowledge Sharing ............................................................................................... 31
      5.4. Knowledge Optimization ....................................................................................... 34
      5.5. Adaption to Rule Changes ..................................................................................... 35
  6. Applications of OLAKO ................................................................................................. 36
  7. Conclusion ....................................................................................................................... 36

Chapter II. Improved Self-organization with Shuffling ODC ............................................ 38
  1. Introduction and background ......................................................................................... 39
  2. Clustering Algorithms ................................................................................................... 40
      2.1. On Demand Clustering ......................................................................................... 40
      2.2. Spyglass ................................................................................................................ 41
      2.3. Shuffling ODC ...................................................................................................... 42
  3. The Load Balancing Task ............................................................................................... 44
  4. Evaluation Setting ......................................................................................................... 45
      4.1. Network Abstraction ............................................................................................. 46
      4.2. Evaluation Environment ....................................................................................... 46
      4.3. Evaluation Scenario .............................................................................................. 46
      4.4. Evaluation Criteria ............................................................................................... 47
  5. Evaluation ....................................................................................................................... 47
      5.1. Basic Scale-Free Network ..................................................................................... 48
      5.2. Algorithm Scaling ................................................................................................ 55
      5.3. Effect of the Physical Topology .......................................................................... 56
      5.4. Effect of the Problem Complexity ....................................................................... 59
      5.5. Effect of the Network Abstraction ....................................................................... 60
<table>
<thead>
<tr>
<th>6. Application</th>
<th>61</th>
</tr>
</thead>
<tbody>
<tr>
<td>7. Conclusion</td>
<td>61</td>
</tr>
<tr>
<td>Chapter III. Vox Populi: Named Entity based Sentiment Mining from Texts</td>
<td>63</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>64</td>
</tr>
<tr>
<td>2. The Vox Populi Vision</td>
<td>66</td>
</tr>
<tr>
<td>3. Named Entity Centric Sentiment Mining</td>
<td>68</td>
</tr>
<tr>
<td>3.1. Process Flow</td>
<td>68</td>
</tr>
<tr>
<td>3.2. The Sentiment Mining Algorithm</td>
<td>69</td>
</tr>
<tr>
<td>4. Self-Analysis Methodology</td>
<td>71</td>
</tr>
<tr>
<td>4.1. Unsupervised Term Self-Revision</td>
<td>72</td>
</tr>
<tr>
<td>4.2. Completeness Analysis</td>
<td>76</td>
</tr>
<tr>
<td>4.3. Evidence Based Dictionary Extension</td>
<td>77</td>
</tr>
<tr>
<td>5. Experimental Evaluation</td>
<td>78</td>
</tr>
<tr>
<td>5.1. Corpus</td>
<td>78</td>
</tr>
<tr>
<td>5.2. Environment</td>
<td>79</td>
</tr>
<tr>
<td>5.3. Evaluation of the Named Entity based Opinion Mining Algorithm</td>
<td>79</td>
</tr>
<tr>
<td>5.4. Evaluation of the Knowledge Self-Analysis Algorithm</td>
<td>81</td>
</tr>
<tr>
<td>6. Application</td>
<td>84</td>
</tr>
<tr>
<td>7. Conclusion</td>
<td>84</td>
</tr>
<tr>
<td>Summary</td>
<td>86</td>
</tr>
<tr>
<td>References</td>
<td>88</td>
</tr>
</tbody>
</table>
Preface

The general goal of this dissertation is to discuss how novel, distributed, autonomic, collective machine intelligence concepts such as knowledge sharing, emergent self-organization and knowledge self-revision can be utilized in particular fields. I extended existing models and algorithms, defined novel models and algorithms, and applied them to existing or new problem fields.

Structure

This dissertation summarizes my research conducted in particular fields. The connection point between the fields is the utilization of some kind of novel, distributed and autonomic machine intelligence based approach.

In the first chapter I discuss a novel approach, an open, online learning based collective-adaptive system. My model, OLAKO, extends the state of the art by using online machine learning to create an open collective-adaptive multi-agent society. The main building blocks of OLAKO are online learning autonomous agents, a society-wide emergent pair-wise knowledge sharing mechanism and a knowledge self-optimization mechanism to optimize and clean the world abstraction and the knowledge. I showed with simulations that OLAKO enables adaptivity even to some drastic environmental changes, knowledge sharing spares time for the individuals, and the knowledge self-optimization speeds up the learning curve for the agents.

In the second chapter I discuss how On-Demand Clustering (ODC) can be improved in self-organizing mobile networks. I extend a baseline ODC algorithm with the concept of link reconfiguration without direct gain in case of lookup failure. I compare the resulting algorithm, Shuffling ODC, to the baseline ODC and another ODC variant, Spyglass. I show with measurements that Shuffling ODC outperforms the baseline algorithm in all aspects, but without the significant messaging overhead observed in Spyglass.

In the third chapter of the dissertation I discuss the problem of named entity centric opinion mining from natural language texts. I propose an adaptive model (Vox Populi) for the task, and an algorithm for named entity centric sentiment mining. After that, I discuss a set of formal methods for the unsupervised self-analysis of the term knowledge base, namely: automatic, unsupervised detection of flaws (e.g. translation errors) in the term dictionary, completeness checking, and precision-recall estimation for the evidence based extension of the term dictionary.

Previous Work

A long list of precursor projects, completed between 2003 and 2011, helped acquiring the necessary theoretical and practical background for the results discussed here. The most important items are the summarized as follows.

Preface

- Local research projects conducted with the help of undergraduate/graduate students on various aspects of machine learning and self-organization. (2008-2011)
- Teaching activities: programming, system design, and data mining. (2003-2011)

Attribution

The work that also resulted in this thesis, at some points, was conducted in collaboration with students and colleagues.

Part of the work behind in Chapter I. was conducted together with Dávid Lányi, who was a student under my supervision at the time. Dávid’s contributions to the simulator framework and the learning module were vital. His valuable comments to the general vision are also acknowledged.

Part of the work behind in Chapter II. was conducted together with Márton Legény, who was a student under my supervision at the time. Márton, after graduation, continued to contribute to the discussions in the field. Márton’s work (the Spyglass algorithm) serves as a reference point for the evaluation of my algorithm. His valuable comments and hints on the simulation methodology are also acknowledged.

Part of the work behind in Chapter III. was conducted together with Dr. Zsolt T. Kardkovács. Certain modules of Vox Populi, namely the article text extractor and the named entity detection algorithm, are Zsolt’s work; I only refer to these modules in this dissertation as part of the vision, without the details. Zsolt’s useful comments, that made my part of the work easy, are greatly acknowledged.
Chapter I.

OLAKO: An Approach for an Open Collective-Adaptive System

In this chapter I describe OLAKO (Online Learning based Adaptation, Knowledge sharing and knowledge self-Optimization), a novel model for a learning based, open collective-adaptive multi-agent system for highly dynamic environments. OLAKO synthesizes three main concepts: online learning autonomous agents, a society-wide emergent knowledge sharing mechanism between the agents, and a knowledge self-optimization mechanism to optimize and noise filter the emerging knowledge. I also discuss a formal description for the system and some resulting phenomena. The proposed models and mechanisms were evaluated in a theoretical world via simulation; where the society of adaptive agents was challenged to win board games against opponents with different strategies and changing game rules. Experiments confirmed that OLAKO enables agents to learn openly and fast both as an individual and as part of a society, and to adapt efficiently to minor and major changes in the environment.
1. Introduction and Background

The concept of autonomic ICT and adaptive systems has gained significant interest in the last decade. By definition, these systems are able to adapt their behaviour to a certain level of changes in the environment without human help [Kephart and Chess 2003]. A basic approach to achieve adaptivity is to use a feedback loop: monitor the environment, select the best action to do, carry out the action and start over with monitoring the effects [Brun et al 2009]. Compared to traditional control systems (e.g. closed loop model) an important difference in the adaptive case is that there is no static optimum to find, instead, the world, so the adaptation target changes from time to time and the system must follow it accordingly.

Another common tool for adaptive or autonomic behaviour is to use explicit models (world model, self-model) describing how the world and the system itself work. These models help in selecting the next action during normal operation, and determine the incident handling or adjustment mechanism when abnormal conditions occur. Powerful tools such as reasoning, ontology or semantics may be utilized in order to guarantee that the operation is effective and convergent. Examples are [Bencomo et al 2008] [Goldsby et al 2008] [Harel and Marelly 2005] and [Benkő et al 2008]. The success of this model based approach heavily relies on the accuracy of the system's explicit knowledge: on the completeness of the models and on the efficiency of the resulting adjustment mechanisms. As long as the environment is in line with the model, the system is guaranteed to operate and adapt efficiently; however it is theoretically impossible for this system to react to un-modelled changes or to perform out-of-model adjustment strategies. In other words, preconceptions that make the adaptation fast and effective in some cases form an obstacle in others, i.e. in the end, hard preconceptions restrict the adaptation potential of the system.

Sometimes, machine learning capabilities are utilized as well. In case of offline learning, the system first learns how to behave in a hypothetical training environment, and later uses this knowledge without further modifications in the operation phase in its real environment. Systems with online learning abilities do not distinguish between training and operation; they learn during (and throughout) their normal operation in the real environment. Even though online learning is more potent for resulting in a truly problem-specific behaviour, it has not been popular in the area of adaptive systems up to now. Examples for offline learning based adaptive systems are: [Borji and Frintrop 2010] for object tracking, [Dhiman and Simunic Roseing 2006] for power management, or [De Jong 2008] and [Martín et al 2009] who tackle with the general aspects of the field. Related online learning based approaches are: [Balch 1998] in the field of robotics, [Seshia 2007] discussing autonomic systems and [Hsien-Po Shiang and van der Schaar 2010] and [Simon et al 2009] who use machine intelligence for wireless multi-hop networks. [Kiningami and Terano 2007] gave a comprehensive overview of the field.

A question this chapter tackles with is whether the level of explicit knowledge, or preconceptions, for an adaptive system can be replaced, or at least relaxed, through excessive use of online learning. In other words, is it possible to unbind the adaptation process, and let the system openly find its way for adaptation, instead of blindly following the knowledge that was once injected, and may not even fit well for the
current problem instance? Openness may be desired on several layers: an open world model built and refined continuously, open adaptation strategies where the system builds its strategy from simple blocks instead of picking one of the pre-coded strategies; social openness to share the knowledge with others and make use of the presence of multitudes of learners (even if we are not sure that everyone faces the same problem); and, finally openness for self-revision such as knowledge cleaning, deprecation, and knowledge optimization.

Collective behaviour; swarm intelligence and emergence have often been coupled with autonomous or adaptive systems not just due to their proven efficiency in nature, but also because the use of simple connected entities or a society of entities fits well with many real world problems. Often, simple actions of a multitude of entities result in an emergent behaviour. Huneman (2008) gives an interesting theoretical discussion regarding the prerequisites of emergence in adaptation. Considerable work has been done in certain related fields, such as emergent self-organization [Saffre et al. 2009].

The goal in this area was to better understand the advantages and bottlenecks of collective learning in an open adaptive system: what gains may knowledge sharing bring, and what kind of bottlenecks should we anticipate when knowledge sharing is used excessively, in an environment of limited observability and changing requirements.

In this chapter I describe a novel, online learning based approach for collective-adaptive systems, including (i) a model for highly adaptive online learning, (ii) a model for autonomous knowledge sharing between learners and (iii) a model for knowledge self-optimization. My approach differs from the state of the art in two main points: I excessively use online learning combined with self-* properties in order to create an autonomous adaptive system, and the model allows for an intensive, even society-wide sharing of the individuals' independently developed knowledge, leading to a better learning curve.

The basic concepts and algorithms of OLAKO were published in [Lánya and Benkő 2010] and in [Lánya and Benkő 2011] and [Benkő, Lánya and Farkas 2011].

The rest of the chapter is organized as follows. Section 2 defines the problem space, goals and the preconditions I took. Section 3 discusses the OLAKO model, namely the learning, knowledge sharing and knowledge optimization models and mechanisms in the system. In Section 4, I propose a formal, mathematical, framework to describe and analyze the system. Section 5 investigates the model via a series of simulation experiments. Section 6 discusses the model’s applicability. And finally, I conclude the results in Section 7.

2. Problem Space and Terminology

In order to define the problem space, let us start from a common agent based abstraction. The problem space is a world containing agents who observe their environment and take actions from time to time. Due to the agent's actions and other reasons, the state of the world changes from time to time. Certain world states provide reward to the agent reaching it, while other states may result in penalty (negative
reward) or no feedback at all. As feedback is not guaranteed after each action, it may take several steps for the agent to find out whether a certain series of actions is successful or not. The agent’s overall goal is to maximize the total reward. These assumptions are also in line with the basic model of reinforcement learning (RL).

However, I made certain assumptions which distinguish this problem space from the classic reinforcement learning setting, and also from the classic collective agent setting.

- **Adaptation challenge (changing world rules).** The environment is dynamic in terms of the requirements, i.e. the logic behind the reward. In this model, *feedback rules may dynamically change* from time to time, so the agent is required to follow these changes with its behaviour. The change, when happens, is seamless for an agent: it receives no explicit notification but senses the change through the missing or unexpected rewards. Note that this is not an unrealistic assumption for real-life scenarios. For example, when the meta-goal is to make the human user happy, and the human’s intentions or habits change from time to time, the system also needs to change its behaviour and find out what the new requirements are. The assumption of changing world rules drastically differs from the model used in classic reinforcement learning systems where the feedback mechanism is static over time, and, as for my knowledge, also from the common adaptive system models.

- **Adaptation challenge (changing fellows).** The environment is dynamic because factors independent of the agent (e.g. other autonomous agents) keep changing it from time to time. The agent also needs to adapt to the behaviour of these fellow influencers. Fellow influencers also change from time to time. Adaptation to this kind of changes is not part of the classic RL model.

- **Collectiveness.** Learners are allowed to share their locally developed knowledge with others, and also to import knowledge from other agents. *Knowledge import* in my model is a pair-wise act: the acceptor agent, when *autonomously deciding* so, imports the donor agent’s knowledge and integrates it with its own with a certain weighting. Compared to the common RL models, this kind of operation is completely novel. Collective adaptive systems may contain a similar mechanism for knowledge import, but not for dynamically and openly built type of knowledge, as the case now is.

- **Self-awareness.** Agents possess the ability of reviewing their own knowledge along certain metrics, and are able to perform knowledge cleaning, deprecation and some kind of optimization on it.

- **Limited observability and dynamic feature set.** The agent’s ability to observe the world is limited, but dynamic. Limited, because the agent can only observe its vicinity, through measuring the presence of certain, predefined *features* there. And dynamic, because, as its knowledge evolves, the agent may decide to rearrange the original features into a new view, hoping to acquire more useful information that way. Note that this kind of openness is neither common in reinforcement learning nor in adaptive systems.

- **Reactivity and high level of non-determinism.** I also assume that the behaviour of the world includes a level of reactivity, meaning that the actions of the agent influence the reaction they receive, but, in a non-deterministic way. This assumption allows for – but does not expect – the presence of intelligence in the environment, which makes the challenge even more realistic.
• **No initial world model.** At start-up, the agent is equipped with *no knowledge* how the world works. The model emerges through learning.

### 2.1. The Showcase World

To make the problem case more concrete, a showcase world was created where the abilities of a solution can be investigated and efficiently evaluated. This showcase world is actually an extension of the simple two-player, fully observable board game known as connect-5 or gomoku.

Connect-5 is a simple board game, an extension of tic-tac-toe for bigger sized boards and longer combinations. There are two players in the game, one with mark X and the other one with mark O. The game starts with a board of tabular arranged empty square cells. Players make steps intermittently; each one places their mark onto an empty cell. A player wins the game if five of their signs are placed consecutively in one row, column, or in a diagonal line on the board. When a player wins, the other one looses. Tie is reached, if the board has no more empty cells, but no one has won.

More formally, the *state* of the game is represented by the board itself (cells and their contents). The transition between states is the *action* of an actor, and the game is basically a time series of game states. Only one actor is allowed to perform action in each particular step. The action is mandatory, so as long as there are empty cells on the board, the upcoming actor must act. After each action, the *new state* is evaluated by the environment, and if win or tie state is reached, the actors receive feedback.

A possible agent based model for connect-5 is the following. The agent observes the world state (board features) after each step, and accumulates the observations into a local knowledge base. Feedback is received at win/tie/lost states. The goal is to find a strategy (series of actions) that leads to a winning state, while the environment is also dynamically and non-deterministically modified by the other player (opponent) from time to time.

While a basic connect-5 game already incorporates several important properties of the problem space, in order to fully match, it was extended into a *Generalized Connect-5 World* (the term ‘world’ is used so that to emphasise that it’s not a single game anymore but a complex collective problem space).

• **Changing game rules.** The game to play depends on the opponent type, and, *is not necessarily* connect-5. For example, it is possible for an agent to have a connect-5 playing opponent in the first \( n \) games, and then, a connect-4 player for the next \( m \) games without notification. Certainly, some level of stability in the game rules over time is expected – there would not be much to do for a fully random game --, but clearly, the rules are not required to be the same throughout the lifetime of the agent. The agent must be able to adapt to the changes on its own, without explicit notification.

• **Diversity by different opponent styles and strengths.** Even if the game is unchanged, the strategy of the current opponent may significantly differ from the strategy of the previous one; hence, the behaviour that worked in the last round may not lead to success now. The agent must be able to generalize its knowledge and in order to overcome diverse opponents.

• **Agent society.** Instead of a single agent-opponent pair, the world consists of a society of agents and a set of opponents (not agents from this viewpoint), where agent-opponent pairs are engaged in multitudes of parallel games, each on a separate board. Members of the agent society are still autonomous entities with
individual experience, strategy and decisions, but they also possess the ability of communicating with each other. Agents may share their expertise with other agents, or make use of others' shared knowledge. Note that, members of the society not necessarily face the same problem instance at the same time, e.g. same opponent strategy or the same game rules; nor do I say that the knowledge of any individual agent is guaranteed to be of help for others. However, the possibility of sharing one's dynamically built knowledge is an important property for this collective system, not just in practice, but also from the theoretical point of view. As experiments showed, even this kind of simple pair-wise knowledge sharing may lead to the emergence of society level behaviour.

- **Limited observability.** Agents have only a rough model of the board, with limited vision and an initial set of observable features. The concrete visibility distance and the initial feature set are detailed in the evaluation section.

This generalized connect-5 world fits with the description of the problem space: Agents observe an extract of the world's state and perform actions. They may receive feedback from the environment in those states when a victory, tie or loss is reached. The world changes because of the agent's action and because of factors that are outside its control (such as the opponent's action, or more basically, the opponent's game choice), ensuring a high level of reactivity and non-determinism. Both adaptivity challenges are present, i.e. changing game rules and changing opponents. Collectiveness is ensured by the agent society. Having no pre-injected knowledge may seem a selfish requirement under static conditions (where a pre-built world model could result in better behaviour from the start), but my goal here is to ensure the openness of the system and its dynamic adaptation ability for immensely new requirements set by the changing environment (e.g. changing game rules).

The state space in the showcase example, supposing a limited board size, is finite. However, this does not make the problem too week or inappropriate, as the observation ability of an agent is limited, so any board size larger than the player's vision is effectively infinite.

As a summary, the agent's job is to learn to act successfully, sometimes under changing rules and environmental behaviours, through a feedback mechanism; plus, to do this on-the-fly, without having pre-injected knowledge or a preliminary training session; possibly in a collective manner, via knowledge sharing, and, with a preference on self-optimization.

### 3. The OLAKO Model

This section describes OLAKO (Online Learning based Adaptation, Knowledge sharing and self-Optimization), my model for the open, i.e. online learning based, collective-adaptive system. I discuss each of the main building blocks, also shown in Figure I-1, in detail.

- Section 3.1 describes the open, online learning mechanism used within individual agents, an extension of the Least-Squares Temporal Difference learning method.
- Section 3.2 tackles with the knowledge sharing model, the mechanism that enables individual agents to share their independently developed knowledge with
each other. I discuss what, with whom, and how to share, and how the multitude of simple sharing steps results in emergent, society-wide effects.

- Section 3.3 discusses knowledge self-optimization aspects, i.e. the automatic reorganization of the empirical knowledge into a more efficient format. My approach is based on lightweight, on-the-fly feature extraction and removal. Optimizations may be undone in order to bring back the knowledge into a format that is understandable for other agents when sharing it.
- Section 3.4 elaborates on the question of fast adaptation to slight and drastic changes via knowledge self-deprecation.

Figure I-1. Constituents of the OLAKO model within and between agents.

The current section describes the above mentioned building blocks and the accompanying mechanisms in detail, and discusses important consequences. Section 4 provides a formal description for the model.

3.1. **Online Learning**

This subsection describes the learning model of OLAKO. It is based upon well-known basic models: on Markov Decision Process and Temporal Difference Learning. The added value lays in their application to a new field (the field of adaptive systems), and in the extensions applied to it (feature set optimization, deprecation, and knowledge import) which will be discussed in details in the next subsections. First I discuss what Temporal Difference Learning is, then detail the nature of the extensions.

3.1.1. **Starting Point: Markov Decision Process and Least Squares Temporal Difference learning**

The inspiration of the OLAKO learning model comes from reinforcement learning (RL). RL is not one specific mechanism but a dynamically improving domain of machine learning models. RL focuses on finding actions in an actual world state in order to achieve a goal desired in that context [Sutton and Barto 1998]. It is assumed, that an agent completing this task is able to sense the environment to some extent, is
able to perform actions which influence that state, and is able to receive feedback from
the environment about its success.

A widely used mathematical model describing reinforcement learning problems is the
Markov Decision Process (MDP) [Bellman 1957]. MDP is defined as a six-tuple
$(S, A, R, P, \gamma, s_0)$, where
- $S$ is the set of world states,
- $A$ is a set of actions,
- $P$ is a state transition probability function $P: S \times A \times S \rightarrow [0, 1]$, where
  $P(s', a, s)$ tells the probability of reaching state $s'$ after performing action $a$ in
  state $s$,
- $R$ is the reward function $R \rightarrow \mathbb{R}$,
- $\gamma$ is a discount factor from interval $(0, 1]$ and
- $s_0 \in S$ is the initial state.

MDP models are often extended with value functions and policies to help making
automatic decisions [Bradtke and Barto 1996]. The purpose of the policy $\pi$ is to model
the behaviour of the agent, i.e. the probability of choosing a given action in a given
state ($\pi: S \times A \rightarrow [0, 1]$). The purpose of the value function is to describe the utility
of a state in case the agent follows a given policy ($V_\pi : S \rightarrow \mathbb{R}$). Practically, the value
function calculates the expected discounted sum of the rewards along the policy driven
path in the state space. It has been shown that in case of deterministic policies the
optimal policy for a given MDP problem can be analytically calculated if the state
transition probability function $P$ and the reward function $R$ are known. This analytic
calculation uses a closed form of the value function, also known as Bellman's equation
($a_\pi$ denotes the action selected by the policy $\pi$).

$$V_\pi(s) = R(s) + \gamma \sum_{s' \in S} P(s', a_\pi, s) V_\pi(s')$$  \hspace{1cm} \text{Bellman's equation}

However, in many RL settings, including our case, the state transition probability
function $P$ and the reward function $R$ are not known, hence the analytic solution is not
possible. Instead, the agent has access to its own empirically experienced subset of the
state transitions (the result of its past actions) and the feedback observed in the visited
states. Like many others in the literature [Kolter and Ng 2009], we will use this
empirical database as an approximator.

The computational cost of MDP depends on the size of the state space ($|S|$). To
limit the size, often a simplified state space is used instead of the real one for the
calculations. One common way is to use a handful of features to describe a state
instead of all the original details. In means of terminology, a feature based linear
approximator for the value function can be defined as [Kolter and Ng 2009]

$$V_s \approx w^T \varphi(s)$$  \hspace{1cm} \text{Feature based approximation}
where \( \varphi(s) \in \mathbb{R}^k \) is a feature vector of state \( s \), and \( w \in \mathbb{R}^k \) is a parameter vector. In my model the agent is born with an initial feature set, however, this set may change. Features may be removed or new features may be introduced during the agent's life, via self-optimization or knowledge import.

While the use of feature vectors was originally suggested in order to keep the computational complexity under control and to be able to deal with large or even infinite state spaces, I also use it for two other purposes, respectively: (i) to facilitate convergence of learning with the selection/creation of relevant and useful features, and (ii) to bring openness into the model through the possibility of dynamically adding and removing features, hence refreshing the implicit world model of the agent.

With the feature based approximation we lose the ability of analytically locating the best policy [Kolter and Ng 2009], but there are other efficient ways for finding a solution, for example the Least Squares Temporal Difference Method (LSTD), and Least Squares Policy Iteration (LSPI). We use LSTD.

The Least Squares Temporal Difference algorithm provides a way for finding a parameter vector \( w \) that approximately satisfies Bellman's equation, while tolerating the use of features and with empirical transition and reward data. Without the full deduction of the method discussed in [Sutton 1988], [Bradtke and Barto 1996], [Baxter and Weaver 1998], and [Kolter and Ng 2009], we recall the main idea and the resulting formulae. LSTD attempts to find a fixed point for \( w \) in the feature based approximation of the value function using the experienced (and feature based) state transitions and the observed rewards. The term to be minimized contains Euclidean norms only, so the optimal fixed point can be analytically determined by solving a linear system of the form \( A^{-1}b \). The exact form of the matrix \( A \) and vector \( b \) is the following (\( s_i \) denoting the \( i^{th} \) state):

\[
A = \sum_{i=1}^{n} \varphi(s_i)(\varphi(s_i) - \gamma \varphi(s'_i))^T \\
b = \sum_{i=1}^{n} \varphi(s_i)r_i
\]

Matrix \( A \) of LSTD  
Vector \( b \) of LSTD

where, \( n \) is the number of observed states up to now, \( \Phi(s) \) is the feature sample matrix containing the feature vector of each observed state (one feature vector per row), \( \Phi(s') \) is the feature sample matrix of the observed destination states after the transitions, \( r \) is the reward sample vector containing the observed rewards, and \( s_i \) and \( r_i \) denotes the \( i^{th} \) state and reward.

In other words, the only knowledge required by the agent for selecting the desirable next state is only a vector \( b \) and a matrix \( A \). The model uses the feature based approximation of the observed state transitions and rewards.

- **Vector \( b \)** gives a picture about the perceived goodness of each state (feature), based on the total (positive or negative) reward experienced there. If the number of features is \( m \) then \( b \) is an \( m \) long vector.
- **Matrix \( A \)** holds information about the state (feature) transitions. If the number of features is \( m \) then \( A \) is an \( m \times m \) matrix. The transitions model the effect of the agent's action along with the effect of the opponent's action, in one unit.
(For example, one state transition in out showcase world is the step of the agent plus the step of the opponent, unless one of them wins or the board is full.) Note that this abstraction does not include any preconception about the number or nature of opponents, so the model is also applicable for $x > 2$ players, and for changing opponent styles.

From our point of view, the most important property of vector $b$ and matrix $A$ is that they can be constructed iteratively; each new experience means a minor addition to them ($O(1)$ cost).

3.1.2. The Basic Learning Model

OLAKO's learning and adaptation model uses LSTD as a basis.

The knowledge of the agent is directly incorporated in matrix $A$ and vector $b$, and indirectly in the current feature set (which is dynamic). $A$ and $b$ are built iteratively.

To select the next action we use LSTD's maximum likelihood decision maker. In each turn, the agent evaluates two effects:

- The provisional direct effect of its own action, i.e. the immediate value of the reached state after performing the action. This is needed in order to recognize when the agent is able to win the game in one step, since in this case, the opponent's reaction will never happen (and will not matter).
- The provisional indirect effect of the action (the provisional reaction of the opponent). This is needed in order to estimate the short and long term consequences of the step, and is realized via the matrix inversion $A^{-1}b$, as described above.

The extensions that distinguish my model from LSTD are the following.

- **Dynamic features.** The feature set behind the knowledge base is not static, new features may be added or removed at runtime, by the Knowledge optimization mechanism. When the feature set changes, $A$ and $b$ get modified accordingly – rows/columns are added or removed.

- **Systematic knowledge review.** OLAKO includes a systematic optimization and deprecation mechanism, in order to optimize and compact the knowledge or remove failed or outdated branches.

- **Collective layer.** OLAKO includes a collective layer, enabling agents to exchange and combine their knowledge ($A$, $b$).

In terms of actual details for the showcase world, 54 heuristic initial features were defined for the learner, describing the number of open-ended 2, 3, 4, and 5 long series on the board belonging to the agent and to the opponent. (These features later, during self-optimization, turned out to be non-optimal, and were reduced to some 28 features in case of a connect-5 game.) The initial goodness value for each feature within vector $b$ is 0 by definition. Matrix $A$ starts as a unit matrix, ensuring that it is transposable. In terms of rewards, a victory means +1, a loss means -1, and a tie results in 0.

3.1.3. Convergence and Generality

When facing large and complex problems in form of reinforcement learning, the path the agent follows in the space of world states within the games clearly influences how easy or hard it is for them to learn the problem. For example, a smart opponent with an efficient strategy will soon teach an untrained agent the kind of situations to avoid.
However, a less effective opponent may produce inconclusive games where the same situation sometimes leads to victory and sometimes not, or, even worse, they may spend a large amount of time on exploring irrelevant parts of the state space that are far away from feedback positions. In the gaming scenario, the path is half-way determined by the opponent, so the opponent's behaviour plays an important role in the 'learnability' of the problem. It is an interesting question what kind of opponent is the best from learning point of view. Against a strong opponent, the agent can easily learn what to avoid, but most probably, will not learn so easily how to gain victory. On the other hand, against a weak opponent, the agent may reach the reward situations without obstacle – first by chance, later willingly –, however, its knowledge will not incorporate information about preventing the opponent from winning. Some experiments later, in the evaluation section, will tackle with the question of opponent choice from the point of view of the learning curve.

The agent's own strategic convergence may also hinder the emergence of good knowledge. Too fast convergence in the knowledge may be dangerous because it develops over-specialized strategies that work well against the current opponent but include no information even about the nearby part of the state space. To avoid overspecialization, the agent may choose prevention strategies, such as picking second-best action instead of the best one. Such a strategy leads to a better coverage of the problem space, which may be suboptimal in the current round, but could help against future opponents with yet unknown strategies.

3.2. Knowledge Sharing

The term 'collective learning' in recent works often refers to two distinct paradigms: one is to use a collection of entities to build a common knowledge base that is above the knowledge of the individual entities; and the second meaning is to utilize a collection of entities to improve the local knowledge of each member of the society. Certainly on the long term the two approaches converge, i.e. a high quality, collectively built knowledge will emerge, but in terms of realization they cannot be more different. In the first approach only the totality of the entities is smart, whereas in the latter case smartness is present in each individual member locally. While clearly the first approach is also very interesting and useful, I dwelve into the second direction because that fits better with the vision of autonomic agents, as here each entity remains able to operate individually, without the necessary presence of others, while utilizing the society in an opportunistic manner.

Collective learning, in my model, means that autonomous agents based on their autonomous decisions share their locally developed knowledge with selected others, realized as a multitude of pair-wise shares in a centralized or self-organizing manner but without stashing common knowledge centrally. This requires the followings:

- **Agent coupling mechanism.** A simple, preferably scalable and self-organization-like mechanism that initiates and controls knowledge sharing by coupling two agents, one playing the donor and the other the acceptor role.

- **Decision making mechanism.** A metric or mechanism for the agent to evaluate itself, the current sharing partner and the actual environment in order to make a decision whether or not to execute knowledge sharing. Both the donor and the acceptor are autonomous agents, so it is their free decision what and
with whom to share/accept, and how seriously to take an imported bit of information. I made the simplification that agents playing the donor role, in order to serve the common goal, are always willing to share their knowledge.

- **Knowledge integration mechanism.** A mechanism to integrate external knowledge into one's own knowledge base. This is a problematic part, as the donor and the acceptor may have experienced completely different requirements (opponent styles, game rules), and, even if that is not the case, their individually developed strategies may follow highly different paths.

As for the general flow of the society-wide knowledge sharing mechanism, the following assumptions were made.

- In each round, an agent plays n games against the same opponent.
- Knowledge import happens at the start of the round (when no actual game has been played with the current opponent yet).
- The agent does not have access to the meta-data of the opponent, all that is known at decision time is the unique identifier of the opponent, and a limited list of the opponent's recent gaming partners.
- The agent coupling mechanism assigns a single donor to the acceptor agent which can either accept or refuse it. When accepting, the donor provides its knowledge and the acceptor imports it. When rejecting, the acceptor starts the round without knowledge import.

### 3.2.1. Agent Coupling Mechanism

The way donors and acceptors get assigned may be modelled in a centralized or in a decentralized manner. In the first case, a central assignment logic selects donors for acceptors; while, when decentralized, each agent seeks for a suitable donor without central help or even synchronization. I defined three main directions.

- **Random donor assignment.** A centralized logic selects a donor for each agent. Implementation wise, this can be achieved in several ways, for example pure random pick where the same donor may get selected for several acceptors, or a homogeneous blending, where, after shuffling the list of agents, each agent gets its right-side neighbour as a donor. The two realizations do not only differ in means of a donor's possible spreading speed, but, also in sense of the purity of the donor's knowledge, i.e. if the donor has already imported knowledge in this round from someone, diluting its essence.¹ However, measurements pointed out that the implementation details do not influence results visibly when using a large enough population and randomized opponent distribution. In the evaluation phase, the homogeneous blending was used within the experiments.

- **History guided donor assignment.** The acceptor agent gets the opponent's last known partner, if any, as a donor. When the opponent has no previous partner, or if that agent is not available any more (e.g. disappeared), as a fallback mechanism, a random donor can be picked. The main idea behind this model is to enable the creation of opponent-specific knowledge lines.

---

¹ For example, supposing random pick and linear order, even if both agent A5 and agent A10 get A8 as donor, they may not get the same material. Hence A8, meanwhile, after donating for A5, had a chance to import knowledge from someone, diluting its original information content.
**Location based donor assignment.** Agents have a physical location assigned, and are able to detect other agents within their visibility area. The donor is one random agent, or, alternatively, the nearest agent from their neighbourhood, if any. The difference between picking a random donor or specifically the nearest one is not the same as for random pick: when using nearest picks, there will always be agent pairs that pick each other in both directions, as they are each other's nearest neighbours. So, picking a random agent from the neighbourhood helps preventing the overblending of physically nearby agents.

### 3.2.2. Decision Making Mechanism

The acceptor, when its next opponent and next donor is known, needs to make an autonomous decision whether or not to execute knowledge import, and, if executing it, what weight to give to the imported knowledge.

Two basic approaches were distinguished: random decision and evaluation based decisions. In case of **random decision**, the agent imports the donor's knowledge with a fixed probability; the decision does depend neither on the actual donor nor on the actual acceptor. In an **evaluation based decision**, the agent applies some evaluator metric in order to make the decision. The following metrics were identified:

- **Sliding window success rate**: the empirical success rate over the window of the last $h$ games. The success of the agent largely depends on the current opponent (e.g. a medium smart agent may consistently perform superbly against a dummy opponent but may just so consistently fail against a strong opponent). Longer windows depict the success rate more objectively – the effect of any specific opponent is weakened with a larger sample –, but it also burdens the agent in sensing when its knowledge gets deprecated.

\[
\text{success} \_ \text{rate} = \frac{1}{h} \sum_{x=1}^{h} \text{reward} \_ x \_ \text{games} \_ \text{ago}
\]

- **Age**: the number of games the agent played so far.
- **Consolidated age**: Measures the maturity of the knowledge rather than the physical age. Consolidation means that the effect of knowledge deprecations and imports are also taken into account. **Algorithm I-1** describes how the consolidated age of the agent should be updated after each step of finishing a game, carrying out a knowledge deprecation step as part of the self-revision, and after knowledge import.

```plaintext
Algorithm I-1. Maintaining the consolidated age of an agent

function update_consolidated_age(event, consolidated_age) {
    if event type is game_played_event then
        return consolidated_age +1;
    if event type is knowledge_deprecation_event then
        return consolidated_age * event.deprecation_factor;
    if event type is knowledge_import_event then
        return event.donor_age * event.import_weight +
        consolidated_age * (1-event.import_weight);
}
```

---

15
As for determining the weight of the imported knowledge, both random weighting and evaluation sensitive weighting were considered.

3.2.3. Knowledge Import Model

The actual knowledge is manifested in matrix $A$ and vector $b$ of the agent, where the size of these data structures depends on the number of features ($A$ is $m \times m$ and $b$ is $m$ long, supposing that the number of features is $m$). This subsection covers the basic case where the donor and the acceptor use the same feature set. The algorithm is extended for any other case in the next subsection.

As $A$ and $b$ hold the agent's knowledge, this is what the donor shares with the acceptor. As shown previously, $A$ and $b$ are built up iteratively, thus, they are additive. The knowledge import mechanism is defined as: the new knowledge of the acceptor is a weighted combination of its old and the donor's shared knowledge. Supposing that the import happens with weight $w \in [0,1]$: 

\[
A_{\text{new}} = wA_{\text{import}} + (1 - w)A_{\text{original}} \\
b_{\text{new}} = wb_{\text{import}} + (1 - w)b_{\text{original}}
\]

Knowledge import model

Weights define the influence of the imported elements. An extreme case is suppressive import where the original knowledge gets zero weight meaning that the imported knowledge replaces the acceptor's own. In this case the acceptor becomes the donor's equal copy or clone. This may be desirable if the imported knowledge is guaranteed to be of high value for the future, while the knowledge of the acceptor is clearly non-performing. However, suppressive import may easily lead to a drastic drop in the population's diversity which may become dangerous when the environment changes. Non-suppressive or blending import of good knowledge may perform somewhat weaker on the short term, but it keeps the population diverse which is a useful property on the long term. Blending import may also accumulate a more general knowledge than the suppressive one, because it tends to store information about uncommon parts of the state space, which may become handy when usual strategies stop working.

Even blending knowledge import, when run on the large scale, may influence the overall behaviour of the society. When there is only a small percent of agents in the society capable of, or willing to, importing, the presence of independently developed, pure knowledge is naturally guaranteed (non-importing agents are bound to have independent, non-diluted knowledge). However, when all society members are overly busy with blending knowledge with each other, the blend soon overtakes the place of the individual information, and all agents, sooner or later, end up with the same common blend, and once again, the population loses its diversity. I call the above phenomenon knowledge overblending. However, if knowledge import is used carefully in some self-controlled way and is equipped with a proper self-revision mechanism, then it may, as experiments point out (see later), result in a highly improved success rate both on the individual and on the common level.
3.3. Knowledge Self-Optimization

Knowledge optimization is a mechanism that restructures the agent’s knowledge and its way of sensing the world into a format that is presumed to be more ‘optimal’ than the actual one. One cardinal point in this definition is that not only the actual direct knowledge, so the data in $A$ and $b$, is affected, but also the way the agent observes the world, i.e. the feature list (so the dimensions of $A$ and $b$). The actual knowledge may indicate that certain features are obsolete (then let us stop observing them), redundant (then let us not observe all of them), or seem to make more sense when combining them in some way (then, let us create a new, combined feature). As discussed before, the knowledge and the feature list are in a strong relationship, hence, they must remain in sync during this process. The other crucial point in the knowledge self-optimization is that as the knowledge in question is empirical its completeness and correctness cannot be guaranteed, hence, the reorganization should happen with care.

The basis of the self-optimization mechanism is to re-organize features: remove redundancy (feature extraction) and meaningless items (feature removal).

3.3.1. Feature Extraction

Feature extraction is rather a purpose than a specific tool; it is a common name for several, often really un-similar methods that all serve the same goal: to make the set of features better. ‘Better’ usually means less redundancy, as explicit or hidden redundancy is a factor that machine learning algorithms cannot handle effectively\(^2\). Feature extraction typically starts with taking a representative sample from the data and analyzing the feature matrix of this sample. In the feature matrix, each row refers to one line of data, and each column refers to one feature (e.g. $m(i,j)$ means the $j^{th}$ feature of the $i^{th}$ data sample). Traditional feature extraction techniques, such as Principal Component Analysis [Pearson 1901], [Jolliffe 2002] or Singular Value Decomposition [Golub and Van Loan 1996], cannot be applied to our model directly, because we do not collect data samples. On the other hand, we do collect a more sophisticated extract from the world states: the state transition matrix $A$.

The basis of my feature extraction mechanism is the actual matrix $A$ that, in some sense, holds a sample of the world. I propose a cheap and intuitive approach to detect redundancy in $A$ based on correlation groups.

The algorithm is motivated by the observation that in a well trained agent, the rows of the matrix $A$ often seem to be very similar to each other. I model this similarity by calculating the correlation between any pair of rows, defined as:

$$ corr(A[i], A[j]) = \frac{cov(A[i], A[j])}{\sigma_{A[i]} \sigma_{A[j]}} = \frac{E[(A[i] - \mu_{A[i]})(A[j] - \mu_{A[j]})]}{\sigma_{A[i]} \sigma_{A[j]}}, $$

Correlation group

where $A[i]$ means the $i^{th}$ row of $A$, $cov$ is the covariance operator, $E$ is the expected value operator, $\mu$ means the expected value and $\sigma$ denotes the standard deviation.

Correlation group in matrix $A$ is a set of rows whose pair-wise correlation with each other is very high, for the showcase world the limit is 0.9.

\(^2\) Redundancy, for example, violates the assumption of feature independence, and biases the distance function.
The outline of the feature extraction algorithm is the following: first, identify correlation groups based on A, then, for each identified group, remove the concerning original features from the feature list, rows/columns from A and cells from b, and replace them with a single item that objectively represents the group. For example, if the first phase identified row1 and row2 as a correlation group, then, when reorganizing vector b, the first and the second element will be removed from the vector and a new element will be appended to the end of it representing the two removed values (for example, the average of the two original values). Algorithm I-2 describes the steps of the algorithm in detail.

There are several possible ways to identify correlation groups; the algorithm below describes a greedy approach. After picking the first item for the correlation group candidate, the algorithm tries to expand the group with new members in a way that the new member correlates with all existing members. The algorithm does not take into account that the order of joining the group is significant, and checking the candidates in a different order could result in a completely different constitution. However, it’s not necessary to have maximal correlation groups, so a greedy algorithm can be an acceptable approximation.

In order to reorganize the features, the agent must maintain a post-processing mechanism that creates the new features from the originally observed ones. I created a feature calculation logic based on the composite design pattern: features can be either simple or complex, where simple features refer to the physically observed elements (the original properties the agent was able to observe at its start-up), while, complex features refer to a set of features (e.g., the average of the components). After n rounds of self-optimization, there may be at most n-1 levels of complex features present in the feature set, as in each self-optimization round, one new level may show up. The agent maintains its current feature set in a list called the feature list. When observing the world, each feature on the feature list calculates its current value.

When reorganizing vector b, the original members get removed and a new aggregate element is added.

When it comes to reorganizing matrix A, it must be taken into account that the matrix is quadratic, so it is not enough to remove some rows and add a new one instead, but also the columns must be handled the same way. However, the step of removing/adding rows and removing/adding columns do not interfere, so they can be performed in an arbitrary order.

Algorithm I-2. Feature extraction

1 Find correlation groups in A. Greedy algorithm:
   1.1 For each row i in A:
       1.1.1 If i is already assigned to a correlation group, then continue with the next i.
       1.1.2 Create a new correlation group G_i. Put i into G_i.
       1.1.3 For all rows j > i:
           • If j is already assigned to a group, then continue with the next j.
           • If (for each row r in G_i, corr(A[r],A[j]) > CORRELATION_LIMIT), then put j into G_i.
       1.1.4 If the cardinality of G_i is 1 then throw G_i away. Otherwise save G_i to the list of correlation groups.

2 For each correlation group, update the feature calculation logic.
2.1 For each rowId in the group: remove the corresponding feature from the feature calculator logic.

2.2 Add a new feature that is the combination of the original (removed) features. I used the average function as the combination operator.

\[ f_{\text{new}} = g(f[\text{rowId}_1], f[\text{rowId}_2], \ldots, f[\text{rowId}_n]) \]

3 For each correlation group, update vector \( b \).

3.1 For each rowId in the group: remove the corresponding value from \( b \).

3.2 Add a new value that is the combination of the original (removed) values. I used the average function as the combination operator.

\[ b_{\text{new}} = g(b[\text{rowId}_1], b[\text{rowId}_2], \ldots, b[\text{rowId}_n]) \]

4 For each correlation group, update matrix \( A \).

4.1 For each rowId in the group: remove the corresponding row from \( A \).

4.2 Add a new row that is the combination of the original (removed) rows. I used the average function as the combination operator.

\[ A_{\text{new}} = g(A[\text{rowId}_1], A[\text{rowId}_2], \ldots, A[\text{rowId}_n]) \]

4.3 For each rowed in the group: remove the column from \( A \).

4.4 Add a new column that is the combination of the removed columns, using the same combination metrics as for the rows.

The optimization algorithm has \( O(n^2) \) computational cost and \( O(n^2) \) space cost where \( n \) is the number of original features. Note that matrix \( A \) contains \( n^2 \) values so it is impossible to process it in less than \( O(n^2) \) time. The most significant factor in the computational complexity is the 1st step where the correlation of \( O(n^2) \) correlation pairs of \( n \) items long rows need to be calculated.

The feature extraction influences the feature count, therefore the size of \( A \) and \( b \). Each correlation group adds a new feature and removes as many old features as the cardinality of group. If \( FC \) denotes the feature count, \( |z| \) means the cardinality of a group, then the feature count can be calculated as follows:

\[ FC_{\text{new}} = FC_{\text{old}} + |z| - \sum |G_i| \]

New feature count after feature extraction.

3.3.2. Feature Removal

Feature removal aims at removing obsolete features that only consume space and processing power to get observed, but do not help the agent in finding a good strategy. The prudent identification of such obsolete features is a hard task, however, simple ‘rules of thumb’ may help in achieving a similar effect.

Useless feature according to a matrix \( A \) is where the corresponding row and column in the matrix are highly unfilled, i.e. these features are practically never used.

Algorithm 1-3 describes how to remove useless features. Note that both the horizontal and the vertical usefulness need to be taken into account, otherwise features occurring in reward states could only be removed mistakenly (for these features, the row corresponding to them will remain unfilled forever, but the corresponding column is in active use).
**Algorithm I-3. Feature removal**

1. For each row $r$ in $A$: determine if $r$ is unnecessary
   1.1 Count sum of values in the $r$th column (sum_vertical) and in the
       $r$th row (sum_horizontal).
   1.2 If sum_vertical $<$ USEFULNESS_LIMIT and sum_horizontal $<$
       USEFULNESS_LIMIT then mark $r$ as unnecessary.

2. For each unnecessary row $r$: remove the corresponding row and column.

**3.3.3. Control of Self-Optimization**

Self-optimization uses the actual knowledge as starting point, so the main success
criterion is the goodness, i.e. completeness and stability of this knowledge. Self-
optimization must not be performed on underdeveloped or unstable knowledge because
the faulty starting point results in even more faulty results. However, as the agent only
has subjective tools to evaluate itself in order to determine the stability of its
knowledge, it is inevitable that the self-revision will, at certain times, occur when it is
not optimal. When the agent detects that the self-revision results in performance
degradation, it should roll back the optimization.

**3.3.4. Feature De-optimization**

Feature de-optimization is needed either for rolling back the optimization, or, in order
to bring the feature set of two agents to a common format.

The de-optimization of complex features is described in Algorithm I-4. Each
complex feature on the list is decoupled into simple ones. Each feature that was
considered useless is brought back to the observation and knowledge.

**Algorithm I-4. Complete feature de-optimization**

1. Decouple the topmost complex features. For each complex feature
   $cf=(f_1,...,f_n)$ on the feature list decouple the feature
   1.1 Decouple it in the feature list
      1.1.1 Remove $cf$ from the list.
      1.1.2 For each $f_i$ in $cf$ add $f_i$ to the end of the list.
   1.2 Decouple it in vector $b$.
      1.2.1 Remove the corresponding row from vector $b$.
      1.2.2 For each $f_i$ in $cf$ add the removed value to the end of the
            vector.
   1.3 Decouple it in matrix $A$.
      1.3.1 Remove the corresponding row from $A$.
      1.3.2 For each $f_i$ in $cf$ copy the removed row to the end of the
            matrix.
      1.3.3 Remove the corresponding column from $A$.
      1.3.4 For each $f_i$ in $cf$ copy the removed column to the end of
            the matrix.

2. If there are still complex features on the feature list (e.g. a $cf$
   was decoupled to further complex features), let us re-run step 1.

3. Add features that were thought useless.
   3.1 For each original feature of that does not appear on the
       feature list
      3.1.1 Add of to the feature list.
      3.1.2 Add a new element to vector $b$ with 0 value.
      3.1.3 Add a new row and column to matrix $A$ in a way that all
          new cells are 0 except for the right bottom one which is
          set to 1.
3.3.5. Knowledge Import for Non-Identical Feature Sets

At knowledge import, for the case of different feature sets at the donor and acceptor sides, feature-de-optimization is the solution that enables knowledge import. First, a common de-optimization stage must be found where the knowledge import is possible, and after the import, re-optimization can be executed.

Algorithm I-5 describes how a common feature set can be found with gradual de-optimization.

Algorithm I-5. Finding common feature set

```
1  While true
1.1  Construct a list of differences in form of (fi, side).
1.2  While the list is not empty.
   1.2.1 Pick the element with the highest level of indirections from the list. If there are more than one such elements pick one of them.
   1.2.2 If it is a complex feature, decouple it on the side it belongs to (using steps 1.1-1.3 in Algorithm 4).
   1.2.3 If it is a simple feature, add it to the other side than it belongs to (using step 3.1 in Algorithm 4).
```

Statement 1. Algorithm I-5 is guaranteed to find a minimal common feature set.

Proof (indirect): Let us suppose that at a certain stage the algorithm decouples feature \( f \) when it is unnecessary, i.e. \( f \) is part of the minimal common feature set. But, as decoupling only occurs when \( f \) has the highest level of indirections from the non-identical features, it is guaranteed that \( f \) is not present indirectly as part of a more complex feature on the other side (only identical features are there with indirection level higher than that of \( f \)). And, it is also guaranteed that \( f \) is not present on the other side directly, as, then it could not be on the difference list. That is why it is impossible that \( f \) is part of the minimal common feature set, so the precondition is false.

3.4. Adaptation to Rule Changes

The agent may react to minor or major changes in the world with two tools: continuous knowledge deprecation and (drastic) instant knowledge deprecation.

In case of continuous knowledge deprecation, the agent decreases the value of each knowledge element with a small percentage (e.g. by 1% after every round). This helps in pruning old and outdated experiences with time.

At instant knowledge deprecation, the self-revision process decrements the value of each knowledge element drastically, or even prunes the knowledge and resets the values of \( A \) and \( b \) to the original null vector and unit matrix. Practically, pruning should also reset the feature set.

\(^3\) Level of indirection: distance of the root and the farthest simple feature constituent.
4. Formal framework

This section discusses the mathematical formalism for the OLAKO model along with statements that can be concluded from the formal analysis.

4.1. Basic Elements

The world consists of n agents and m >= n opponents, where an agent at a given time is modelled by its actual knowledge \( K_a \), its history \( H_i \), and some kind of self-evaluation measure \( M_i \). History means a hs long sliding window of the last games in terms of (opponent - gained reward) pairs, where reward means the outcome of the game, which can be victory (+1), tie (0) or defeat (-1).

\[
A = \{a_1, ..., a_n\} \quad \text{Agents}
\]
\[
O = \{o_1, ..., o_m\} \quad \text{Opponents}
\]
\[
t = \{1, 2, ...\} \quad \text{Discrete time}
\]
\[
A_{i,t} = (K_{i,t}, H_{i,t}, M_{i,t}) \quad \text{Agent: knowledge, history, self-evaluation}
\]
\[
H_{i,t} = \{h_{i,-1}, ..., h_{i,-hs}\} \quad \text{Sliding window history}
\]
\[
h_{i,x} = (o_{i,x}, r_{i,x}) \quad \text{History item: opponent and reward}
\]
\[
r = \{-1, 0, +1\} \quad \text{Reward}
\]

The game assignment function \( GA \) assigns the next opponent to the agent at time \( t \). A game \( G \) means that the agent, with its current knowledge, plays with its opponent, until reaching some feedback state. Sometimes I also use a simplified notation \( G_{i,t} \) for \( a_i \)'s next game which is played against the opponent assigned by \( GA \). The outcome of the game is an improved knowledge and history at the agent's side, and the reward gained. Optionally, the reward of the game is sometimes denoted as \( res(G) \), and the effect of the game on the agent is denoted as \( effect(G) \). In any case, the next game means that the knowledge of the agent changes, the (current opponent, result of the game) pair gets added to the history, and, the self-evaluation may also change.

\[
GA : A \times t \rightarrow O
\]
\[
GA(a_{i,t}, t+1) = a_{i,t+1}
\]
\[
G : A \times O \rightarrow (A_{t+1}, r)
\]
\[
G(a_{i,t}, o_{i,t+1}) = G(a_{i,t}) = G_{i,t}
\]
\[
res(G) = r
\]
\[
effect(G) = A_{i,t+1}
\]

Game assignment function

The next opponent of \( a_i \), as \( a_{i,t+1} \)

Game (with learning)

Alternative notation for \( a_i \)'s next game

Reward of the game

Effect of the game on the agent
OLAKO: An Approach for an Open Collective-Adaptive System

\[ \text{effect}(G_{i,t}) = (K_{i,t+1}, (o_{i,t+1}, res(G_{i,t}), h_{i,j}, ..., h_{i,j-h+1}), M_{i,t+1}) \]

While games are primarily defined for agents, it is easy to see that, as neither the history nor the self-evaluation metric are really utilized during the game, it is also possible to define the game upon the knowledge of the agent. (This view neglects the change in history and self-evaluation). This is the knowledge view of the game.

\[ G : A_i \times O \rightarrow (A_{i+1}, r) \]
\[ G : (K_{i}, H_{i}, M_{i}) \times O \rightarrow ((K_{i+1}, H_{i+1}, M_{i+1}), r) \]
\[ G : K_i \times O \rightarrow (K_{i+1}, r) \]

Agent view and knowledge view of the learning

To make the model complete, a non-learning view of the game was also defined. In this kind of game, the knowledge of the agent does not change.

\[ NLG : A_i \times O \rightarrow (A_i, r) \]

Non-learning game

4.2. Success Probabilities

Let us note that a game between the same agent \((a_{i,j})\) and opponent may not always result in the same reward, mainly due to the non-deterministic nature of opponents. Hence, while it is not possible to predict the result of a specific game with perfect precision, it is possible to model it probabilistically, as suggested by the weak law of large numbers.

The probability of winning a game against a concrete opponent can be defined as the empirical average of \(k\) repeated theoretical games between the current agent and the selected opponent. The knowledge view of the success rate can be defined similarly.

\[ p_{\text{succ}}(a_{i,t} \mid o_x) = \frac{1}{k} \sum_{i=1}^{k} res(G(a_{i,t}, o_x)) \]  

Probability of success against \(o_x\)

\[ p_{\text{succ}}(K_{i,t} \mid o_x) = \frac{1}{k} \sum_{i=1}^{k} res(G(K_{i,t}, o_x)) \]  

Knowledge view of the success

The probability of the agent’s success in the next game is defined as the weighted average of the conditional probabilities of winning a game against each specific opponent weighted by the probability of meeting that opponent in the next round. So, if the distribution of opponents for the next round is known, it’s possible to calculate the expected success rate.

\[ p_{\text{succ\_next}}(a_{i,t}) = \sum_{i=1}^{m} p_{\text{succ}}(a_{i,t} \mid o_i) p(o_i) \]  

Success in the next game

The same way, a long-term success rate for the \(j^{th}\) game from now is defined as:
\[ p_{\text{succ} \_t}(a_i, t, j) = \sum_{i=1}^{m} p_{\text{succ}}(a_{i,t+j} \mid o_i) p(o_i) \]  
Success \( j \) games away

**Statement 2.** *(Long and short term success without learning)* When learning is switched off, and the distribution of opponents does not change with time, then the probability of short term success (success in the next round) and long term success (success in the \( j \)th round from now) are equal.

Proof. Let us use the knowledge view of the success. When learning is switched off, the \( K_{t}\) and \( K_{t+t+j} \) are equal, so the two sums are equal too.

Let us define the probability of a game’s teaching value as the difference between the agent’s success rate after and before the game. When the agent learns from the game, the value is positive.

\[ \text{val}(G_{t+i}) = p_{\text{succ}}(a_{i,t+i+1}) - p_{\text{succ}}(a_{i,t}) \]  
Value of the next game

When the next opponent is not known, the utility of learning in the next round can be defined as a weighted sum of the values of possible next games, weighted with the probability of the opponent’s occurrence.

\[ \text{utility}_{\text{learn}}(a_{i,t}) = \sum_{i=1}^{m} \text{val}(G(a_{t,i}, o_m)) p(o_m) \]  
Utility of learning

**Statement 3.** *(Utility of learning with time)* If the distribution of opponents is constant, the utility of learning probabilistically decreases to a low value on the long term.

Proof sketch. Case 1: when the system manages to learn the problem. As the problem to learn does not change, sooner or later the system will reach its limits in learning it, which means that the utility of any new learning round is low. Case 2: when the system is not able to learn the problem (e.g. circular repetition rounds). For any large enough \( x \), when the system does not manage to learn the problem in \( x \) rounds, it is unlikely that it will learn it in \( x+1 \) rounds, so the probability of learning is low.

The agent’s long term success rate may also be defined through the value of each game until that.

\[ p_{\text{succ} \_t}(a_i, t, j) = p_{\text{succ}}(a_{i,t}) + \sum_{x=1}^{j} \text{val}(G_{t,i+x}) \]  
\[ \sum_{x=1}^{j} \text{val}(G_{t,i+x}) = \sum_{i=1}^{m} \sum_{k=1}^{x} \text{val}(G(a_{t+i}, o_k)) p(o_k) \]  
Success \( j \) games away (iterative view)
Statement 4. (Short and long term success over time) Over time, if the distribution of opponents is fixed, the long term success probability of the agent approaches the short term success probability.

Proof. As, sooner or later, the utility of learning decreases, so learning does not increase the success probability of a knowledge, i.e. it will not significantly matter.

4.3. History Based Approximation

As the agents use their experience to improve their knowledge, and, the experience depends on their opponents only, let us use the agent’s history to approximate the agent’s success. Note that statements tackle with probabilities, and not with single games. While the flow, so the actual utility and even the result, of each single game is unique, on the large scale, it is possible to make probabilistic statements on it.

The success of an agent is approximated as a function of its history.

\[ p_{\text{succ}}(a_t) \approx f(h_{t-1}, \ldots, h_{t-h}) \]

History based approximation of success

Let us define the history fingerprint of the agent as a \( k \)-long vector where \( k \) is the number of possible opponent types and the \( i^{th} \) cell of the vector represents the number of games played against the \( i^{th} \) opponent type. Let us extend this basic definition with knowledge depreciation: knowledge depreciation proportionally decreases the value of each cell in the fingerprint.

\[ fp(a_t) = \{v_x\}_{x=1}^{k} \]

\[ v_x = \sum_{j=1}^{h} \left[ \begin{array}{c} 0 \mid o_x \not\subset h_{t-j} \\ 1 \mid o_x \subset h_{t-j} \end{array} \right] \]

History fingerprint

Statement 5. (Fingerprint shape) If the fingerprint based success approximation is valid, then similar fingerprints suggest similar success probabilities for old enough agents in case of a suitable history size and for fixed opponent distribution.

Proof. Step 1: Proof for equal fingerprints. As the history is the same, and fingerprint based approximation is valid, the success rates must be equal. Step 2: Proof for similar but not equal fingerprints. The difference between the two fingerprints is a small percentage of games, which, for old enough agents, does not matter.

Statement 6. (Fingerprint distribution of the population with time) If history based success approximation is valid, then with time, in case of probabilistic opponent assignment, the history of each member of the society will be similar. Hence, their success rate is expected to be similar too.

Proof. Fingerprints depict the distribution of opponents, which is constant for all agents. The second part of the statement is an implication of statement 5.

\(^4\) While a probabilistic statement may say that the success rate is 0.4, the concrete result of the game will be either -1, 0, or +1 but never 0.4.
History based approximation enables us to model the knowledge sharing between agents. Let us approximate knowledge sharing with history sharing. So, in this sense, sharing knowledge is considered as sharing the history. This is clearly just an approximation, as learning does depend on the actual knowledge, so, the amount the donor learnt from its historic opponent may not be the same as the acceptor would have learnt. The history based model of knowledge sharing is a weighted sum over the fingerprint vectors.

\[ fp_{w,a_i}^{a_j} = w \cdot fp(a_j) + (1-w)fp(a_{i,j}) \]  

Fingerprint for knowledge sharing

**Statement 7.** (Knowledge sharing and the learning process) If knowledge sharing can be approximated with history sharing, then knowledge sharing speeds up the learning process for the case of fixed probability opponents.

Proof. The precondition says that knowledge sharing acts the same way as meeting the donor's opponent. So, importing knowledge has a similar effect on the acceptor as having played a game with the donor's each opponent. This saves time for the acceptor (gives information about games without playing them), hence, speeds up its learning process.

**Statement 8.** (Knowledge sharing and the society) When knowledge sharing can be approximated with history sharing, and all members of the society participate in a probabilistic knowledge sharing, then, all agents are bound to reach a very similar success probability at an accelerated speed.

Proof. Probabilistic sharing means that all agents have the same chance of receiving the knowledge. So, when sharing is used society-wide, everyone will get a very blended history fingerprint, hence, similar success rate. The increased speed is because agents spare learning rounds.

### 5. Experimental Evaluation

The online learning, knowledge sharing and self-revision model was evaluated in series of simulation based experiments. Four groups of experiments tackle with the four aspects of the model. I examine (i) the basic learning indicators of the model first for a single agent, later for a static and dynamic society of learning (but not knowledge sharing) agents, (ii) the consequences of knowledge sharing with different sharing models and society dynamics, i.e. birth and death characteristics, (iii) the effects of knowledge self-optimization, and finally (iv) I show an example for the system's ability to adapt to drastic changes through continuous knowledge deprecation.

#### 5.1. Simulation Setup

The simulation environment, written in JavaSE 6, consists of learner agents, a set of fixed strategy opponents, and a central logic for assigning opponents to agents to play
games and agents to agents to share knowledge (unless it is done in a self-organizing manner). When not stated otherwise, I used connect-5 opponents following a mathematically optimal strategy with certain failure rates, meaning that in that percentage of the steps the opponent fails to take the mathematically optimal action, and chooses a random step instead. When the failure rate is 100%, the opponent is a pure random agent. The board size was 10x12 cells.

The basic experiments were executed on a MacBook with 4GByte memory and 2.5 GHz Intel dual core processor, society involved experiments were batched on an 4x2 core 2GHz Intel processor based server with 64 GB RAM and on a 2x2 core Sun server with 8 GByte RAM, and feature optimization tests took place on a desktop PC with 2GB RAM and Intel dual core 2 GHz processor.

5.2. Online Learning

5.2.1. Role of the Opponent's Strategy

The first experiment was designed to point out the effect of opponent strategy on the learning path of the agent. The setting consists of an adaptation phase and an evaluation phase. In the adaptation phase the new born agent plays a very high number of games (340) with the same opponent in order to learn the rules\(^5\). Then, in the evaluation phase, learning gets switched off, and the agent is evaluated against a perfect strategy opponent (0% failure rate) with another 100 games. This setting was repeated five times, for five newborn agents and five different opponents with 100%, 15%, 5%, 1%, and 0% failure rates respectively (100% failure rate means random steps). Note that, as it has been shown before, in a limited board connect-5 game the player who starts, when following a mathematically optimal strategy, is unable to lose. In my setting, the starting role was assigned randomly, meaning, the agent is bound to have at least 50% non-winnings in the evaluation phase.

Figure I-2 shows the success rate of each agent in the adaptation phase (left side) and evaluation phase (right side). The agent trained with a random opponent is able to win 99% of its adaptation games\(^6\), but its gained knowledge turns out to be very weak against a perfect opponent (83% lost). The smarter the training phase opponent is the fewer lost games the agent produces in the evaluation phase. However, none of the agents is able to beat the perfect opponent in the evaluation phase, which points out, that having a static opponent strategy may not be optimal from the point of view of learning. An intuitive explanation for that is that some opponents are in favour of teaching what to avoid, while others tend to teach how to reach victory, but they are usually unable to provide both kinds of information.

\(^5\) The long learning phase guarantees that the agent has time to pick up as much knowledge as the opponent can provide. Previous experiments, not detailed here, pointed out that the agent's learning rate slows down significantly after ~50 games, and the knowledge completely stabilizes after at most 100 games when playing against medium failure rate opponents. Based on these observations, we picked the number of games to be more than 3 times as high as the reference stabilization time.

\(^6\) Why is the success rate of this agent so high against a random opponent? The key to the answer is the observable world state, or rather, the feature set the agent uses to observe the world. The random opponent's action is unlikely to change the world state (it is unlikely that the opponent builds even 2-long series). This means that the agent is able to freely move between world states, and, as there's no reward in the current world state, it tries to move away into another world state (one with different feature values). Hence, the feature set will easily lead the agent into having a 2, 3, 4, and finally 5 long series.
5.2.2. Effects of Changing Opponents

The second experiment was designed to demonstrate the effect of changing opponents and continuous knowledge deprecation. I used a pool of 100 agents and a pool of 100 opponents (17pcs of 1%, 17pcs of 5%, 17pcs of 10%, 17pcs of 20%, 16 pcs of 50% and 16pcs of 100% failure rate static players). In one round each agent plays $k$ games with its assigned opponent. Then, agents get a new, randomly selected opponent for the next round. I measured the performance of the population at the end of the rounds by summarizing the number of won, tied and lost games of the individuals. The experiment was run for a totality of 100 games per agent ($100/k$ rounds). The experiment was repeated three times separately, always resulting in the same curve shapes.

The results of this experiment will be used as baseline for knowledge sharing.

Figure I-3 displays the win and non-lost (win+tie) statistics of the society for two different $k$ values and knowledge deprecations. The first line pair (solid red + dotted red) shows the win and win+tie curves for $k=1$ games per opponent. The two curves are very near to each other in the beginning, then separate after 20-30 rounds, meaning that with time agents tend to reach ties instead of losing games. While the win curve stabilizes after 25 rounds at $\approx 40\%$, the win+tie curve keeps increasing until the 80th round where it stabilizes at $\approx 70\%$. The second line pair (blue solid with $x$, blue dotted with $x$) shows a $k=1$ setting but with a continuous (1% per round) knowledge deprecation. Knowledge deprecation has no visible effect on the curve shape. The third data pair (green rectangles and triangles) depict the results of the $k=10$ setting, where the 10 games are played per opponent, and results are also collected once per each 10 games. This less frequent opponent change results in a slightly lower performance, compared to the $k=1$ case.

I use $k=1$ and deprecation=1% in all following experiments.
OLAKO: An Approach for an Open Collective-Adaptive System

Figure I-3. Total success rate of a society. 100 agents, 100 opponents, random assignment, no knowledge sharing. Red: 1 game per opponent. Blue: 1 game per opponent and knowledge deprecation. Green: 10 games per opponent.

When examining the collective behaviour of the society towards one specific opponent, a kind of stabilization can be observed. Figure I-4 displays another view of the previous $k=1$, deprecation=1% experiment. I measured the average success rate of the society per opponent type in every 10 rounds (i.e. the number of wins divided by the number of games played against that opponent type in the given 10-rounds time frame). The average success rate stabilizes in 1-6 time frames (10-60 rounds) depending on the opponent type; after that, only slight deviations from the stable value can be observed. The fastest stabilization occurs in case of the 100% FR (random) opponent, where the society is able to win 100% of the games right after the first time frame.

Figure I-4. Performance stability against opponents. The common success rate against the same opponent stabilizes after a certain time.

5.2.3. Society Dynamics and Learning

The next experiment examines the learning dynamics (still without knowledge sharing) of the society when the age of agents is different. I defined a birth-death process with the following parameters: initial society size, new agents per round, maximal age of an agent (in terms of rounds). For example a 10.2.15 society means that there are 10 agents in the society initially, 2 newborn agents occur in each round, and agents die
after reaching the age of 15 rounds. Again, the results of this measurement serve as a reference point for comparison with knowledge sharing.

Figure I-5 displays the number of wins and wins+ties in case of 6.1.1000 and a 6.1.40 birth-death process. In both cases, the initial society size is 6, one new agent is born in each round, and agents leave the society at the age of 1000 and 40 games, respectively. Practically, 6.1.1000 is a birth only society, as agents do not die within the measured time frame of 100 rounds. Analyzing the pure number of won or non-lost games in this case is not enough, as the size of the society changes over time.

Figure I-5. Success curve of two birth-death characteristics \( \text{initialSize.grows.maxAge} \) without knowledge sharing. Red: birth only society. Blue: birth-death society.

A precise approach for the analysis could be to consider the age distribution of the society. Figure I-6 visualizes a not so precise but simple approach where the number of wins and non-losts is normalized by the size of the society. The drawback of this pure norm is that it does not differentiate between old and new agents, while, naturally, older, i.e. more knowledgeable, agents are expected to provide a higher success rate than youngsters. The role of age can be observed in the chart: the society with more old agents (6.1.1000) outperforms the society with fewer old agents (6.1.40) in means of win+tie games.

Figure I-6. Normalized success curve of two birth-death societies without knowledge sharing.
5.3. Knowledge Sharing

Knowledge sharing is first evaluated on the agent’s level, then, on the society level.

5.3.1. Individual Knowledge Sharing

The first experiment examines the effect of knowledge blending when the knowledge to import is of different origin (trained by a different opponent) than of the acceptor's knowledge. Four differently trained agents got knowledge injections, and then got evaluated offline against the perfect opponent. The four combinations are: (1) untrained acceptor + best trained (0% FR) donor, (2) random opponent trained acceptor + best trained donor, (3) connect-4 trained acceptor + best trained donor, and (4) best trained acceptor + random trained donor. Note that while in the first three cases, the knowledge to import is intuitively of high value, in the fourth case the randomly trained knowledge, which is to dilute the strongest acceptor’s own knowledge, may be, intuitively, thought to be of low value.

As Figure I-7 shows, knowledge import had some means of positive effect in all cases. Not surprisingly, in the first three cases when the injected knowledge was of ‘high quality’, the blend is better than the starting point. In the fourth case, a good agent receives ‘weak’ knowledge, and, while the percentage of non-lost games slightly drops, the imported knowledge helps the agent actually win a game against the strongest opponent. This proves that, if the problem is of non-additive nature, the principle of superposition is not valid, hence, even a good and a bad knowledge may result in a better-than-original blend. However, the opposite may also occur: the blend of two relatively good knowledges may result in a worse-than-original blend.

![Figure I-7. Knowledge import in individual agents. Performance of agents against the 0% FR opponent before and after the import.](image)

5.3.2. Society Wide Knowledge Sharing

Society wide knowledge sharing was evaluated with three donor assignment and decision making mechanisms.

- The first one (Random) is random donor assignment, where after playing with several variants (true random vs. shuffled list, fixed acceptance vs. random

---

The fourth experiment was re-run three more times to confirm the results, and the victory consistently occurred in all repetitions.
acceptance probability); I choose a shuffled list based implementation with a fixed acceptance and 0.2 as import weight a.

- The second mechanism (Opplast) is a history based decision without fallback, where the donor is the opponent’s last partner if it exists (otherwise no import occurs). The import weight is also 0.2 here just as in the first mechanism.

- The third mechanism (Age) is an age based decision, where a (shuffled list based) random donor is accepted if it is over 30 games of age or if it is at least 6 rounds older than the acceptor. This heuristic was inspired by the shape of the learning curve (for example, Figure I-3).

To serve as reference point, the non-sharing curves are also shown. The pool of opponents include the same amount of 1%, 5%, 10%, 20%, 50% and 100% failure rate static entities as in the previous experiments.

The first experiment tackles with a birth-only system. Figure I-8 shows the success curve of a 50.1.1000 society, while Figure I-9 displays the same curve for a 6.1.1000 society. In both cases, knowledge sharing is performed after each round, the system is run for 100 rounds, and one game is played per round against a sufficiently large pool of opponents, with random game assignment.

The most interesting message of the chart is that some of the knowledge sharing strategies work. In Figure I-8, with Age and Opplast the society is able to achieve significantly (25%+) more non-losses, and the winning curve of Age is also clearly above the baseline (No share) winning curve. In terms of non-loss games, the same effect is visible in Figure I-9, but the improvement in wins is not confirmed here. An explanation for that is that 50.1.1000 is an unbalanced society in means of age, and, mostly-ready agents around round 40 have a high chance of importing mature knowledge; while in 6.1.1000 the chance for that is much smaller, and, as there is no fallback mechanism in Age, less imports will happen. Opplast can show off its effectiveness earlier than Age does, because Age needs a stable mature population to work effectively, which is not available around the start-up phase. However, not even Opplast can be significantly effective when there is not enough knowledge to import around startup, as it is the case in 6.1.1000.

The other important message of the charts is that frequent random sharing does not work on a society level. I repeated the experiment with multitudes of combinations of settings, but the results were always the same. Simply, the blind import of knowledge, when the society is flooded with undeveloped agents, leads to delusion, and the inconclusive knowledge of the young agents corrupts the knowledge of the older ones.

---

a There was no significant difference between the implementations.
The next experiment examines the behaviour of a birth-death society. I used a 6.1.40 system because the initial size gives a chance to play against each opponent type (there are 6 of them), and the death rate is chosen to be 40 because previous experiments suggest that the knowledge of an agent starts stabilizing around 40 games when random opponents are used and there is no knowledge sharing.

Figure I-10 shows the success curves of the mechanisms in a 6.1.40 society. The size of the society, after growing to 46, drops to 40 and stays there, so, the numbers after the drop are easy to compare. In this kind of dynamic society, the lack of knowledge sharing is quite unsuccessful, sometimes, the win+tie count of No share is exceeded by the pure Win count of some knowledge sharing mechanisms. Clearly, Random does not work here either. However, both Opplast and Age highly outperform the reference
point, resulting in a very stable behaviour during the 46-100 rounds. Age is better in means of non-lost count, on the other hand, Opplast tends to deliver good win counts more consistently than Age does.

Figure I-10. Success curve of a 6.1.40 society in 100 rounds at various knowledge sharing directions.

Figure I-11 displays the normalized success count of the same system. The chart confirms the previous observations.

Figure I-11. Normalized success curve of a 6.1.40 society in 100 rounds at various knowledge sharing directions

5.4. Knowledge Optimization

The knowledge optimization experiment examines the efficiency of the feature extraction algorithm in means of removing the redundancy so facilitating the learning. First, a new born agent was trained with a perfect opponent in 40 rounds; this agent is
saved under the name K1. (K1 uses the default 54 features.) Then, I used K1’s knowledge for feature extraction, which resulted in 28 features, so, half the size of the original feature set. I trained a new, independent agent using the identified 28 features against the perfect opponent again, in 40 rounds, this is K2 agent. (K2 observes 28 features). Finally, both K1 and K2 got evaluated offline against an opponent of 50% failure rate in 20 games.

Figure I-12 shows the result of the evaluation. K2 with its 28 features was able to achieve more wins and less lost games (0 lost games in fact). This proves that the optimized feature set is more effective for learning.

![Result distribution](image)

**Figure I-12.** Agent success after 40 rounds with the default features and the optimized features, evaluated against a 50% FR opponent.

### 5.5. Adaption to Rule Changes

The last experiment demonstrates how an agent adapts to drastic rule changes. In this experiment I do not use self-revision; with self-revision the transition to the new world rule set would have been faster. In the experiment I trained an agent to learn connect-4 in 50 games with a 0% FR connect-4 playing opponent. Then, without notice, I gave the agent the 0% FR connect-5 playing opponent. Figure I-13 shows the transition. In the first 25 games the agent has serious problems with the game. Then, its new experience replaces the old one (helped by the 1% knowledge deprecation after each step), and, after 101-125 games it is at the same performance level as if it was trained originally to play connect-5.
6. Applications of OLAKO

The OLAKO model is a fairly general approach; its application is not limited to any specific area. Experiments in the extended connect-5 world were intended to demonstrate that the model is feasible and ‘works’ in practice. OLAKO can be utilized in any field where the problem can be modelled as a time series of world states, a finite number of actions, occasional feedback, and, in order to use knowledge sharing, it is good if multitudes of learners face the problem at the same time. One area of application is intelligent gadgets or intelligent services (for human use), such as an adaptive graphical interface, a proactive cache, or a proactive intelligent building system that makes the user’s everyday life easier by learning the general habits of people as well as the personal specialties of each user. Another application area is for adaptive networking, for example, a quality of experience (QoE) based media stream controller that uses the human’s feedback to change the underlying settings of the stream. A third specific application area is in online gaming and augmented reality, where machine agents need to interact with humans in a virtualized environment, similar to the showcase world.

The fact that the model abstracts the raw observations into features makes it also possible to problems with limited or unreliable information, such as information delayed by network conditions, corrupted or non observable raw signals or blocked sensors. Depending on the length and frequency of these signal outages, the agent could either adapt to the lack of information (basically rebuild its knowledge utilizing the still meaningful features only), or incorporate a small constant instability into the knowledge base. Note that the sample problem I used for demonstrating the capabilities of OKALO includes a similar constant noise, i.e. the lack of knowledge about the opponent’s strategy.

7. Conclusion

In this chapter I described and discussed OLAKO, a novel model for a society of online learning, adaptive, knowledge sharing and self-optimizing agents. After elaborating on
the basic learning model, I focused on the effects of a society-wide knowledge share mechanism, and on the details of autonomous knowledge self-revision. A formal description of the model that helps in understanding the system and points out interesting effects was also presented. Simulations proved that the system is able to learn problems in a theoretical world, adapt even to some drastic changes, and, that knowledge sharing and self-optimization facilitate the high performance of the society.

The novelty of OLAKO is that it combines open (online) learning with the foundations of a collective-adaptive system, hence overcomes many burdens present in non-open collective-adaptive systems due to false or insufficient domain knowledge. From the viewpoint of online learning, novelties are the knowledge sharing and self-optimization (dynamic feature selection) ability of agents. The model can be of use in multiple application areas from intelligent human-related services to networking and augmented reality.

An interesting direction for future research in the area is to examine how the openness of collective online learning can be synthesized with a domain knowledge based approach. One possible direction is to use the domain knowledge for identifying new raw features, i.e. to equip the agent with new observation abilities on-the-fly. Another direction is to use the outcome of the learning to clean and validate the domain knowledge.
Chapter II. Improved Self-organization with Shuffling ODC

In this chapter I consider the problem of improving On Demand Clustering (ODC) in self-organizing networks. I propose a novel algorithm, Shuffling ODC that extends the baseline ODC algorithm with the concept of link reconfiguration without direct gain in case a cluster expansion attempt fails. The motivation behind Shuffling ODC was to overcome the too strict locality principle (and the resulting topologic starvation effect) of ODC that hinders the efficient expansion of clusters. I compare Shuffling ODC with two reference algorithms: the baseline ODC and another ODC extension (Spyglass) via a series of simulations on scale-free and random topologies. Spyglass and Shuffling ODC bring similar benefits significant faster cluster creation, significantly larger clusters, and significantly more efficient workload sharing characteristics. However, while Spyglass generates an excessive amount of messaging overhead, Shuffling ODC’s communication cost stays small, at most linear with the baseline. I also show that both Spyglass and Shuffling ODC preserve the principle of locality.
1. Introduction and background

Self-organization in networks is gaining significant interest today in various application fields from 3GPP LTE to pervasive computing and other mobile ad-hoc network scenarios [Dahalman et al. 2006] [Cook, Augusto, Jakkula 2009], [Augusto 2007]. Elements in these networks need to share resources, communicate with each other, perform load balancing or provide services to each other without the help of a central management entity.

While human network managers may possess information about the state of the whole network9, in case of self-organization, nodes need to rely on locally available information, even if it is incomplete and non-objective, when making their autonomous decisions. This is referred to as the locality principle, or principle of locality.

Self-organization often uses simple algorithms with emergent properties, i.e. the multitude of executions result in a complex, “intelligent” high-level behaviour which is of a different quality than the simple building blocks themselves. Emergent algorithms are often inspired by biological, chemical or physical phenomena like the behaviour of swarms, insect colonies, the human brain or immune system. These paradigms were used in numerous ways to solve problems in computer networks. A thoroughly researched application field is load balancing [Di Caro, Ducatelle and Gambardella 2005], [Montresor, Meling and Babaoglu 2002], [Montresor, Meling and Babaoglu 2003], [CASCADAS D3.2 2007]. Self-organization in overlay networks is also widely used in grids and peer-to-peer data networks to realize distributed data or control flows without the need for a supervising entity [Jain, Mahajan and Niswonger 2000], [Apel and Böhm 2005], and [Clarke, Sandberg, Wiley and Hong 2000]. It is notable that due to the low processing power these algorithms demand, self-organization can even be used in wireless sensor networks [Gburzynski, Kaminska and Olesinski 2007], [Krishnan 2004].

Clustering, in the context of self-organizing networks, means that entities of the network search for other entities that meet a certain criterion (similarity in case of normal clustering and complementariness in inverse clustering) and establish connections with them in form of overlay network links. [Abbasi and Younis 2007] [Parkas, Maurer, Ruf, and Plattner 2006] The resulting overlay network can be used for various purposes. Load balancing is one of these use cases; here, entities share their local workload with members of the cluster. While the efficient creation of clusters is a prerequisite for efficient load balancing, it is not a sufficient condition: the load balancing algorithm must also answer the questions when and with which cluster member to share the load, and what information to use (collect, store and update) when making these decisions.

This work focuses on biologically inspired, fully distributed self-organization algorithms for large overlay networks, with an emphasis on clustering and load balancing. I considered the problem of on-demand clustering, where the goal is to expand the cluster when demand for that occurs, for example the cluster has more

---

9 In large networks it is often technically not feasible (or very expensive) to possess detailed information about the detailed state of all entities. However, a human operator is typically able to assess at least mid level state information about the network, and is also aware of the topology and the constitution.
workload than it can process. The demand-driven problem statement has two main differences from the general case: (i) the amount of clustering is proportional to the demand, hence there's no driving force to expand clusters to their theoretical maximum size or to assign each and every node to a cluster, and (ii) the speed of cluster formation is more crucial, because the demand depicts an immediate request, one, that cannot wait.

The novelty in the chapter is the improved clustering ability, achieved with the modification of a well-known baseline clustering algorithm, and demonstrated through a load balancing scenario. The focus of this chapter is Shuffling ODC, a novel self-organization algorithm, defined by us. To serve as comparison, I also use a second algorithm, besides the baseline, in this dissertation, namely Spyglass. Spyglass was created by a member of our research group, and published in [Legény 2010], [Legény and Benkő 2010], and [Benkő and Legény 2011]. Our related work including network dynamics characteristics is further discussed in [Legény and Benkő 2011].

The rest of the chapter is organized as follows. Section II describes the three clustering algorithms: the baseline On Demand Clustering, and its two extensions: Spyglass, and Shuffling ODC. Section III defines the load balancing task that triggers and uses clustering. In Section IV and V, I investigate the algorithms along different network abstractions through simulation, and compare the results. Finally, Section VI concludes the work.

2. Clustering Algorithms

While numerous clustering algorithms are known today, I chose a specific algorithm family, known as On Demand Clustering. This section, after drafting how the base algorithm works, describes the other reference algorithm (Spyglass) and my algorithm (Shuffling ODC).

2.1. On Demand Clustering

On Demand Clustering, or ODC, is an emergent self-organization algorithm with beneficial properties on the node degree, developed at BT Labs [Saffre, Tateson, Halloy, Shackleton and Deneubourg 2008], [Di Nitto, Dubois, Mirandola, Saffre and Tateson 2008], [CASCADAS D3.2].

The basic idea of ODC is that when a node (initiator) is in need for expanding its cluster, it requests one of its randomly selected neighbours (match maker) to provide a link to a possible new member to include in the cluster (match). The match maker looks for a possible match among its own neighbours. If a new match is found, in order to keep the total number of links in the network under control, a process called rewiring begins. This includes the establishment of a new link (between the initiator and the match), and the removal of an existing link (between the match maker and the match). The algorithm can be summarized shown in Algorithm II-1.

Algorithm II-1: ODC

1. The clustering process is initiated on demand, i.e. when a node is in need of expanding its cluster.
2. The node where the demand for clustering raises, called initiator, selects one of its neighbours to serve as match maker.

3. The match maker looks for a matching node, one that meets the initiator’s clustering criterion, among its own neighbours. Returns the match if found.

4. When the match is not already a neighbour for the initiator, a process called rewiring begins. The initiator and the match establish a new link, while, in order to keep the total number of links under control, the match maker removes its own link to the match.

It has been shown that ODC results in an emergent self-organization behaviour, clusters are formed and expanded in accordance with the local demand. However, ODC may not perform equally well in all cases. While the strict locality principle, when searching for a match, guarantees that the communication overhead remains low, it sometimes prevents the fast formation of clusters. This is especially the case in sparse or type-wise highly diverse networks, where often no suitable match is present in the direct neighbourhood of most match makers. This kind of starvation may also be evolved due to the algorithm itself, sooner or later, match makers start running out of further matches. The strict gain-centric operation and the locality principle here hinder the formation of large enough clusters.

ODC uses message based communication: each request, response, link creation and link removal step costs a message.

2.2. Spyglass

The motivation behind Spyglass was to overcome the strict locality principle of ODC by extending the range of the lookup.

In Spyglass [Legény 2010], [Legény and Benkő. 2010], the match maker is able to look one hop farther than in the original ODC, i.e. at its two-hop neighbourhood. Spyglass differs from ODC in the last two steps: in the behaviour of the match maker and in the details of rewiring. The detailed differences are shown in Algorithm II-2.

Algorithm II-2: Spyglass (differences from ODC marked with italic)

1. The clustering process is initiated on demand, i.e. when a node is in need of expanding its cluster.

2. The node where the demand for clustering raises, called initiator, selects one of its neighbours to serve as match maker.

3. The match maker looks for a matching node, one that meets the initiator’s clustering criterion, among its own neighbours. When no match is found there, the match maker continues with checking its two-hop neighbours (the neighbours of its neighbours) for a match. Returns the match if found.
4. When the match is not already a neighbour for the initiator, the rewiring begins. The initiator and the match establish a link, while, in order to keep the total number of links under control, \textit{the connection node to the match removes its own link towards it.} \textit{(The connection node is the match maker if the match is originally its own direct neighbour. Otherwise the connection node is the node between the match maker and the match.)}

The two-hop search horizon, clearly, promises more lookup success. However, looking at two-hop neighbours is an operation of exponential cost, so some kind of optimization is inevitable if we want to avoid the exponential communication overhead. Hence, Spyglass nodes are equipped with a local data structure called Neighbour Cache (NC) storing information about nodes in the vicinity. The NC needs to be established and updated with care; while both the build-up and refresh steps are expensive, having outdated content is even worse as it hinders the self-organization process. In [Benkő and Legény. 2011] various caching strategies were discussed to provide a good trade-off between communication overhead and keeping the cached contents up-to-date. In this chapter these strategies are not discussed in details.

The biggest advantage of Spyglass is its reaction speed to requests. The ability of returning a match fast and with a high probability is a very beneficial property. Clustering, in my model, happens on demand, so a request to the match maker means that the initiator truly needs the match as soon as possible. When the expansion of the cluster is slow the initiator may easily reach critical levels of overload.

Spyglass, just like ODC, uses message based communication. It introduces new lookup messages for detecting the match maker's two-hop neighbours.

2.3. Shuffling ODC

The motivation behind Shuffling ODC was to overcome the strict locality principle of ODC by allowing small changes in the topology without direct gain (i.e. a useful overlay link). The idea is to enable, due to these changes, the gathering of new match makers once the existing ones become of low efficiency.

Shuffling ODC extends the original ODC algorithm with two new ideas: (i) preferring non cluster member nodes as match makers, and (ii) "mingling with the vicinity", i.e. introducing minor topological changes when no match is found. These modifications are achieved with minor changes in steps 2-4 of ODC, as shown in Algorithm II-3. The idea is to modify ODC's behaviour for the case when the match maker is unable to locate a match. While in the baseline algorithm that resulted in no rewiring, in Shuffling ODC the match maker returns a link to one of its randomly picked neighbours. This step, even if does not bring direct gain, tries to increase the diversity of the possible match makers for the initiator.

Algorithm II-3: Shuffling ODC (differences from ODC marked with italic)

1. The clustering process is initiated on demand, i.e. when a node is in need of expanding its cluster.
2. The node where the demand for clustering raises, called initiator, selects one of its non-cluster neighbours to serve as match maker. *When there is no non-cluster neighbour available, a cluster neighbour is selected as match maker.*

3. The match maker looks for a matching node, one that meets the initiator’s clustering criterion, among its own neighbours. If found, returns the match. *If not found, returns a random node from its neighbour list.*

4. When the returned node is not already a neighbour for the initiator, a rewiring process begins. The initiator and the match establish a new link, while, in order to keep the total number of links under control, the match maker removes its own link to the match.

Shuffling ODC tries to overcome the *topologic starvation* caused by ODC’s positive-feedback-only rewiring. What is topologic starvation? I observed that in some cases, with ODC, the set of available match makers for a node is not dynamic enough, because only the node’s original neighbours and the matches found by clustering in the meantime can play this role. After a while, this set may easily run out of matching but not-already-cluster-member neighbours. This is especially the case when the workload is high, hence, cluster members themselves get overloaded and distribute the workload they receive further. In this situation the initiator is unable to expand its cluster via either of its neighbours in a sensible way, because (i) non-cluster members already returned all matches they knew, i.e. are useless now, and (ii) cluster members are already using their matching-colour neighbours for further distributing the excess workload, so even if they return a match, that node is already being used at another point of the distribution chain, hence, it is not really an addition. Shuffling ODC tries to overcome that by introducing potential new match makers when no match is found, and, even better, doing this at nearly no additional cost.

In other words, what Shuffling ODC does is a positive feedback based cluster expansion (just like in ODC) combined with a negative feedback based match maker set expansion (novelty).

The preference for non-cluster (non matching-colour) nodes as match makers comes from the observation already drafted above, namely, that direct cluster members of the initiator often already use their matching neighbours for further distributing the load, i.e. what they could provide is just a movement of certain nodes within the chain. Non cluster members, as themselves do not distribute the initiator’s workload, have a higher potential to find truly new matches, ones not already participating in the chain.

Shuffling ODC uses the same communication messages as ODC.

### 2.3.1. Communication overhead of Shuffling ODC

Let us model the communication overhead of ODC in form of the match maker related communication (lookup request, lookup, response sending) and the cost of rewiring (link removal, link establishment). Supposing that the probability of successful match is $p$, the expected cost of an ODC clustering step is:
Improved Self-organization with Shuffling ODC

\[ ODC\_\text{cost} = \text{matchmaker\_ cost} + p \times \text{rewiring\_cost} \]

The difference between ODC and Shuffling ODC is that in Shuffling ODC rewiring takes place both in case of successful and unsuccessful lookups. Hence, its cost is:

\[ \text{Shuffling\_ODC\_cost} = \text{matchmaker\_cost} + \text{rewiring\_cost} \]

**Statement:** (Linear communication overhead) The cost of Shuffling ODC is at most linear with the cost of ODC.

**Proof:** Let us note that \( p \leq 1 \), so \( 1/p \geq 1 \). Let us multiply the cost of ODC by \( 1/p \).

\[ 1/p \times ODC\_\text{cost} = 1/p \times \text{matchmaker\_cost} + \text{rewiring\_cost} \]

Note that as \( 1/p \geq 1 \), this amount is larger than the cost of Shuffling ODC.

\[ \text{Shuffling\_ODC\_cost} \leq 1/p \times \text{matchmaker\_cost} + \text{rewiring\_cost} \]

\[ \text{Shuffling\_ODC\_cost} \leq 1/p \times \text{ODC\_cost} \]

So, the overhead of Shuffling ODC is at most \( 1/p \) times the cost of ODC.

**Statement:** (Best-case equality of overhead) The overhead of Shuffling ODC and the overhead of ODC are equal when the loop success is 100%.

**Proof:** This is an implication of the previous statement (\( p=1 \)).

### 3. The Load Balancing Task

The load balancing task is a common application area of clustering. I use a model where a load balancing problem generates the demand for the clustering.

The model is as follows.

- The overlay network consists of coloured nodes and links connecting them.
- Nodes are able to process matching-colour jobs.
- Links are not coloured.
- Jobs enter the overlay network via coloured workload generators, each statically attached to a matching-colour node.
- Workload generators generate jobs and put them on the queue of the attached node. The expected value of the generation rate is constant.
- Nodes consume jobs from their local queue.
- When a node feels overloaded, i.e. its local queue length exceeds a limit, it shares the local workload with its cluster neighbours (matching-colour neighbours). Load sharing means to transfer a job from the local queue to each neighbour via the overlay link. The sharing decision may also be bound to specific conditions on the capacity of the link (e.g. not to share over low-capacity links) and on the remote node (e.g. the acceptor is not overloaded).
- When a node feels overloaded and cannot find enough appropriate neighbours to share the load with, a demand for clustering occurs. The node, as an initiator, triggers a clustering step. Clustering is aimed to reorganize the overlay topology and to create a new link to a suitable node on demand.

Figure II-1 shows an example for the model. The local queue of the \( i \) is overloaded, so the node first tries to transfer some of its jobs to the existing cluster neighbours. When that is not enough, \( i \) selects a match maker \( (mm) \), and initiates a clustering step. In this example, the lookup is successful, a new link between \( i \) and the match \( (x) \) will be established, while the link between \( mm \) and \( x \) will be removed. Once \( x \) becomes \( i \)’s cluster member, it will also receive parts of the excess workload.

Figure II-1: Graphical representation of the model: coloured nodes and coloured workload generators. If the local queue length is exceeds a limit, the node shares its workload with cluster neighbours. When that is not enough, the node initiates a self-organization step, i.e. picks a match maker, asks it for a match, and executes a rewiring. In the example above, the link between \( mm \) and \( x \) will be removed during the rewiring, and a new link between \( i \) and \( x \) will be established.

Note that the load balancing task introduces a certain amount of new requirements towards the clustering algorithm. Normally, the resulting cluster size is the most important performance metric of a clustering algorithm. However, when clustering serves load balancing purposes, the need for cluster expansion is limited. We do not need to generate the largest possible clusters, instead, just large enough clusters for the local excess workload (the creation of clusters larger than that would not bring further advantages but would cost communication messages). On the other hand, the clustering speed becomes vital for load balancing: the initiator needs the match urgently. Given that the job generation rate is constant, every unsuccessful search for a match just worsens the initiator’s situation.

4. Evaluation Setting

A simulation environment and a set of evaluation scenarios were created in order to objectively evaluate the clustering algorithms through a load balancing scenario.
4.1. Network Abstraction

For the evaluation I used the scale-free graph model as the abstraction of the network. While the random graph model is also widely used for evaluating networked algorithms, the scale-free model matches many real-life problems more precisely [Barabási and Albert 1999]. In both models, the total number of links between the nodes is fixed; it is possible to generate both random and scale-free graphs for a given node and link count. However, while the node degree distribution in case of a random graph is a Poisson distribution, for scale-free graphs it follows the power law, i.e. there are a few nodes in the network with extremely high node degrees (similar to the presence of hubs and routers on the Internet). This non-homogeneity in the node degree is what makes scale-free graphs a close-to-reality model that can point out bottlenecks of algorithms that would not be visible in random graphs. At the same time, the power-law distribution makes the investigation of scale-free networks with analytical models very hard; instead, typically, simulation based evaluation is used.

The scale-free topologies were generated with Kumar's copy method [Kumar et al 2000], which is an iterative model. In each generation step, one new node is added to the network. The new node gets connected to some of the existing nodes, with a probability proportional to the current node degree of each existing node. Separated (zero-degree) nodes, if any, at the end of the generation phase were connected to randomly selected other nodes in order to guarantee that all nodes have at least one neighbours.

I used networks of 1k nodes with ~2k links, and 10k nodes with ~20k links for the evaluation. The exact numbers are:

- Basic scale-free graph: 1000 nodes, 2029 links.
- Large scale free graph: 10,000 nodes, 20,028 links

4.2. Evaluation Environment

The simulation framework was written in Java 6 SE, and the experiments took place on a desktop PC with 2GHz dual core processor and 2 GByte RAM.

4.3. Evaluation Scenario

Measurements were conducted on the above showcase networks. Workload generators were attached to 30% of the nodes. Nodes belonged to 10 different classes (colours), and load balancing swung into action once the local queue length exceeded the static limit of 5 unprocessed jobs. The choice of network size and workload generation density was motivated by our previous work on random networks [Legény 2010], where this problem size was found to be convenient for demonstrating the scalability of the emergent algorithms as well as for pointing out the differences between directions.

The simulation was executed for 500 rounds. The generation rate of workload generators changed between 1 and 10 jobs per round (selected randomly at startup, and constant during the life time of the generator), and the duration of the generation lasted in some cases for 200 rounds while in others for 500 rounds. When the generation is ceased before the end of the experiment, the shape of the tail curve gets visible too.
I simulated the same scenario, excess workload triggering self-organization, in all cases. Network dynamics, i.e. the appearance of disappearance of nodes, was not considered during simulation.

4.4. Evaluation Criteria

The following metrics were applied for evaluation:

- **Message count** depicts the amount of *communication overhead* generated by the algorithm variant. Small message counts are preferred over high message counts.

- **Number of clustered nodes** during simulation and at the end. Intuitively, the larger this number is the better the self-organization performs. However, the demand for clusters is constrained by the amount of workload, so the cluster size cannot grow indefinitely.

- The **unprocessed job curve** drafts the processing dynamics of the system by plotting the number of injected but unprocessed jobs versus time. The unprocessed job curve displays two very important properties of the clustering algorithm: (i) the *reaction time* is manifested in the shape and in the peak of the curve, smaller peaks and flatter and more rapidly decreasing curves are preferred, and (ii) the area under the unprocessed job curve depicts the total waiting time, which is the smaller the better.

- The **number of overloaded nodes** depicts the dynamics of the demand for clustering. This metrics can be used for two purposes: (i) to understand the characteristics of the demand that triggers clustering; and (ii) to understand how the demand is silenced or excited by the clustering and load balancing algorithm. Note that a node without an attached workload generator may also get overloaded (e.g. receives a new job per round from more than one sources), hence trigger additional clustering. I call this phenomenon *secondary overloading*.

- I also use a visual evaluation of the self-organized network, using on a spring based layout engine that draws a 2D approximation of the topology, considering each link as a spring and each non-cluster node as a magnet with the same pole. Visually evaluated properties of the network include balance between colours, distribution of non-clustered nodes, etc.

5. Evaluation

The evaluation section first discusses the results measured on the Basic Scale-Free Network, then, elaborates on the differences and similarities observed on the Large Scale-Free Network. On both networks, I used two settings: (i) an excitation setting where job generation lasted for 500 rounds (workload generators generate more work than what the network is able to process, so this setting excites self-organization), and (ii) a run-up tail-off setting, where job generation stops after 200 rounds, giving time for the network to consume some of the excess workload.

In the second part of the section I investigate how the change in different parameters affects the results. I analyze what happens when the physical topology is taken into account with significant, distance dependent delays. Afterwards, I provide
and example how the system changes if the task becomes easier or harder, i.e. the number of colours is decreased or increased. I also discuss the effect of using a random network instead of a scale-free one as network abstraction.

5.1. Basic Scale-Free Network

5.1.1. Excitation Shapes

The first experiment tackles with the case when, with generation throughout all 500 rounds of the experiment, workload generators flood their attached node with jobs (for generation rates ≥ 1, the attached nodes must use clustering and load balancing in order to keep track with the load). The three algorithms show different behaviours.

Figure II-2 shows the number of clustered nodes per round. With each algorithm, the curve starts with a rapid rise; the more rapid the rise the better the algorithm's reaction time is. ODC is the slowest from the three, and after the increment phase, it soon reaches a static limit (588 clustered nodes), where match makers do not have matching neighbours any more, hence no further match can be returned. Spyglass reacts to the clustering request very rapidly, in the first few rounds it is by far speedier than the other two algorithms. However, after reaching a peak at 740 clustered nodes, the curve starts to decrease. The explanation for that is the following. When the match maker returns a new indirect match (there is a connecting node between the match maker and the match), then the match maker and the connecting node need to remove their link. This, at the first sight, may not decrease the number of clustered nodes. However, if the initiator and the match are of colour A, and the match maker and the connection are of colour B, then this step disconnects the cluster members of colour B. And, if the match was already clustered before the step (in the cluster of some other node than the initiator), and the match maker and the connection node are one another’s only cluster centre, then, all together, this step decreases the number of clustered nodes by disconnecting one of the colour B nodes from their group. In Shuffling ODC, the same decrement effect can be observed, but, due to the small local rewirings without gain, the algorithm manages to turn back this trend. Shuffling ODC manages to cluster approximately 795 nodes.

The number of processed jobs per round in Figure II-3 shows exactly the same trend. For each algorithm the job processing curves are slightly below the clustered nodes curve, showing that in each round there are a few clustered nodes without load. This is mostly due to the natural clusters coming from the initial topology: some matching-colour nodes are neighbours, i.e. the count as clustered nodes, but never receive a job in their life.
Figure II-2: Number of clustered nodes (Basic Scale-Free network, generation time 500 rounds)

Figure II-3: Number of processed jobs per round (Basic Scale-Free network, generation time 500 rounds)

Figure II-4 depicts the number of shared jobs per round. The curves, again, follow the trend already seen above, however, with smaller numbers (the work done by the nodes with generators is not shown here.)

Figure II-5 visualizes the nature of the overload. The set of darker curves shows the number of overloaded nodes that have workload generators attached (primary overload) per round, while lighter shades display the number of overloaded nodes without workload generators (secondary overload). Primary overload curves follow the same shape, in Shuffling ODC a slight constant rise can be observed, which is due to the fall in the number of clustered nodes (nodes that could bear the load with the help of their cluster get overloaded after losing a cluster member). Secondary overload curves have an initial peak, but with time, when these nodes also find partners for workload sharing, their number decreases rapidly.

Note that while the workload sharing intensity of Shuffling ODC is below the intensity of Spyglass, the overload curve and the job processing curves do not show significant difference. This indicates that Shuffling ODC emerges in more efficient workload sharing chains. The large initial secondary overload also suggests that multi-layer workload sharing chains emerged around the high-intensity generators.
Improved Self-organization with Shuffling ODC

Figure II-4: Number of shared jobs per round (Basic Scale-Free network, generation time 500 rounds)

Figure II-5: Number of overloaded nodes, with primary load (direct from generators) and secondary load (from other nodes) (Basic Scale-Free network, generation time 500 rounds)

The unprocessed job curve in Figure II-6 confirms the previous result. ODC is the weakest in bearing with the overload. Spyglass, after initially being the strongest of the three, falls back to the second place due to the breakup of clusters. Shuffling ODC is similar to Spyglass in the overall performance, but without the performance fallback.

Figure II-6: Number of unprocessed jobs per round (Basic Scale-Free network, generation time 500 rounds)

Figure II-7 displays the messaging overhead of each algorithm. Clearly, Spyglass with its excessive lookup consumes one magnitude more messages than the other two.
ODC is the most economic in this sense. Shuffling ODC produces a linear overhead compared to ODC, with its rewirings without direct gain.

The resulting topologies were also evaluated with the help of a layout engine that produces a contraction and distraction based 2D layout for the overlay network. Without elaborating on the details, large circles display the nodes with workload generators (the diameter is proportional to the count of cluster neighbours), small empty circles are clustered nodes, and small squares are non-clustered nodes. Workload generators, when they are each other's neighbours, are connected by a visual link in the figure. (The layout algorithm concentrates on the non-generator-attached neighbourhood of generator nodes, and sometimes fails to place workload generators near enough, even if they’re connected. The link helps detecting these rendering failures.)

Figure II-8, Figure II-9, and Figure II-10 shows the layout produced by ODC, Spyglass and Shuffling ODC, accordingly. ODC produces the less connected layout: a large number of solitary, un-clustered nodes and only small clusters can be observed. Clusters, if any, tend to stay of strictly local nature. Spyglass and Shuffling ODC produce approximately the same level of organization, a level that is significantly higher than ODC's. The difference in the number of solitary nodes can also be observed: in Shuffling ODC there are clearly less standalone squares. The homogeneity of all algorithms is good: cluster sizes and the distribution of non-clustered nodes are balanced.
Figure II-8: Layout of ODC at the end of the simulation (Basic Scale-Free network, generation time 500 rounds)

Figure II-9: Layout of Spyglass at the end of the simulation (Basic Scale-Free network, generation time 500 rounds)

Figure II-10: Layout of Shuffling ODC at the end of the simulation (Basic Scale-Free network, generation time 500 rounds)
5.1.2. Run-up and Fall-off

The second experiment examines the behavior of the algorithms when the generation of load stops after 200 rounds. I concentrate on the differences here.

In the shape of the clustered node curve in Figure II-11, it can be observed that as the demand for clusters gets weaker, in general, both Shuffling ODC and Spyglass tend to destruct some of the existing groups, and not rebuild them. This is an interesting emergent property: clusters may get weakened when no more demand for them is present (but demand for some other cluster colours is available).

The processed job curve in Figure II-12 emphasizes the practical difference between the processing speed of the algorithms. The rapid drop after 200 rounds illustrates the presence of very successful workload generator-attached nodes: they process and/or share all incoming jobs immediately, so, as the workload stops arriving, they finish their operation within a few rounds. The tail shapes of the curves show that ODC needs more time to finish the task than the two optimized algorithms.

The number of shared jobs in Figure II-13 shows that after step 200, other clusters find and incorporate the former members of the successful clusters (that run out of jobs as the generation is ceased).

![Figure II-11: Number of clustered nodes (Basic Scale-Free network, generation time 200 rounds)](image1)

![Figure II-12: Number of processed jobs per round (Basic Scale-Free network, generation time 200 rounds)](image2)
Figure II-13: Number of shared jobs per round (Basic Scale-Free network, generation time 200 rounds)

Figure II-14 shows the number of overloaded nodes, and Figure II-15 the number of unprocessed jobs. Note that Spyglass produces less overloaded nodes, but with less clustered nodes and the same unprocessed job curve, suggesting that the overloaded nodes in Spyglass have higher amounts of overload than on Shuffling ODC. The original ODC, while following the same overload shape, leaves a significantly higher amount of unprocessed jobs. The two improved algorithms have only a few unprocessed jobs left at the end of the 500th round, compared to the total load, while ODC has significantly more left, and shows much slower processing characteristics.

The messaging overhead, shown in Figure II-16, suggests that the communication follows the demand, and when the demand for clustering decreases, the message count does, too.

Figure II-14: Number of overloaded nodes, with primary load (direct from generators) and secondary load (from other nodes) (Basic Scale-Free network, generation time 200 rounds)
Improved Self-organization with Shuffling ODC

![Graph](image)

Figure II-15: Number of unprocessed jobs per round (Basic Scale-Free network, generation time 200 rounds)

![Graph](image)

Figure II-16: Messaging overhead per round (Basic Scale-Free network, generation time 200 rounds)

As a summary, Shuffling ODC in the previous experiments resulted in average in 25-30% more clustered nodes, 17-23% more processed jobs per round, 20-30% improvement in job sharing, and 30-50% less job processing time, compared to ODC.

5.2. Algorithm Scaling

The scalability of the algorithm was demonstrated with experiments on the Large Scale-Free Network.

Experiments resulted in the same curve shapes as observed on the small network. No significant difference was detected. At some points the details of the mechanisms are clearer on the large network. As two examples, let us have a look at Figure II-17 and Figure II-18, showing the shared and processed job curves for the 200 rounds long workload generation scenario. The change after ceasing job injection can be clearly seen: after a short reaction period, the amount of job sharing reaches the original level. Shuffling ODC here slightly outperforms Spyglass, and both improved algorithms outperform ODC by at least 20-30% in the number of processed jobs, and by 35-50% in means of workload sharing. The relationship between the communication overheads of the algorithms does not change (see Figure II-19).
Improved Self-organization with Shuffling ODC

Figure II-17: Number of shared jobs per round (Large Scale-Free network, generation time 200 rounds)

Figure II-18: Number of processed jobs per round (Large Scale-Free network, generation time 200 rounds)

Figure II-19: Messaging overhead per round (Large Scale-Free network, generation time 200 rounds)

5.3. Effect of the Physical Topology

The next set of experiments tackles with the question how Spyglass’s and Shuffling ODC’s somewhat weakened locality principle affect the real efficiency of workload sharing. One may intuitively assume that these algorithms cause workload sharing between physically more distant nodes than ODC does, and this difference may influence their real-life performance. Another question here is if the use of more
realistic, distance-dependent sharing overheads affect the curve shapes discussed previously. I executed the next set of simulations with a setting where the transfer time of the jobs was visible and proportional to the physical distance between the nodes.

Figure II-20 shows the average workload sharing distance, i.e. the average physical distance to the receiver node, throughout the simulation. Clearly, the average distance in Spyglass and Shuffling ODC is higher than in ODC, but the difference never exceeds the +10% limit.

![Graph showing average workload sharing distance per round](image)

Figure II-20: Average workload sharing share distance per round, with physical distance dependent sharing overhead. (Basic Scale-Free network, generation time 200 rounds)

Table II-1 summarizes the average sharing distance values and average cluster sizes of the two algorithms compared to the values produced by the basic ODC. Clearly, the increment in the sharing distance is significantly lower (+7 and +8%) than the increment in the cluster size (+19 and +18%). In other words, the algorithms do not violate the locality principle by spreading unnecessarily over the network. The workload remains physically more concentrated than the increment in the increment in the cluster size would suggest. Figure II-21 visualizes the difference in a chart.

<table>
<thead>
<tr>
<th></th>
<th>Shuffling ODC</th>
<th>Spyglass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. share distance</td>
<td>107% of ODC</td>
<td>108% of ODC</td>
</tr>
<tr>
<td>Avg. cluster size</td>
<td>119% of ODC</td>
<td>118% of ODC</td>
</tr>
</tbody>
</table>

Table II-1: Average sharing distances and cluster sizes, relatively to ODC
The simulations with visible, distance-proportional sharing overheads showed no significant change in the trends. The higher overhead introduced some delay, but otherwise, the trends in the lines remained unchanged. Figure II-22 and Figure II-23 show two of the charts. The sharing overhead, i.e. that time the job transfer takes, brings a certain amount of delay into the system: the receiver node receives the job later, senses the overload later, and initiates self-organization later.

Figure II-21: Increments in sharing distance and cluster sizes, compared to ODC. (Basic Scale-Free network, generation time 200 rounds, with physical distance dependent sharing overhead)

Figure II-22: Number of unprocessed jobs per round, with physical distance dependent sharing overhead (Basic Scale-Free network, generation time 200 rounds)
Improved Self-organization with Shuffling ODC

Figure II-23: Number of clustered nodes, physical distance dependent sharing overhead (Basic Scale-Free network, generation time 200 rounds)

Simulations were, additionally, also repeated with a sharing delay model that is proportional to the square of the physical distance between the nodes (as often the case in wireless networks is). The trends were also confirmed here. Table II-2 summarizes the increments observed. The sharing increment is still significantly smaller than the increment in the cluster size, hence, the locality principle is not violated. In means of locality preservation, Shuffling ODC outperformed Spyglass: it produced both larger clusters and shorter sharing distances. The explanation is that Spyglass tends to find more remote matches initially, hence it, at once, brings a delay into the system. Not only the actual job sharing but also its further effects are delayed, e.g. the secondary overloads and the triggered further clustering steps. On the other hand, depending on the situation, transferring a job to a more distant node may also have its benefits, for example, it may, so to day, ‘guarantee’ that the receiver is not already overloaded.

<table>
<thead>
<tr>
<th></th>
<th>Shuffling ODC</th>
<th>Spyglass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. share distance</td>
<td>108% of ODC</td>
<td>109.5% of ODC</td>
</tr>
<tr>
<td>Avg. cluster size</td>
<td>118.5% of ODC</td>
<td>116% of ODC</td>
</tr>
</tbody>
</table>

Table II-2: Average sharing distances and cluster sizes with significant, distance square proportional workload sharing delay, relatively to ODC

5.4. Effect of the Problem Complexity

The final set of experiments investigated how the hardness of the problem influences the performance of the algorithms. I consider the problem to be harder if the number of available matches gets lower, i.e. the number of colours is increased or the connectedness of the network is decreased.

Figure II-24 shows the processed job curve of the 200-rounds workload generation setting on the basic scale-free network with 50 different colours. Note that the average rate of workload generation was not changed, so, the same amount of workload now needs to be processed by 5 times less nodes. Spyglass, here, outperforms the other two in speed, it processed sometimes even 50% more jobs than ODC does. Shuffling ODC, even if not in such a high amount as Spyglass, clearly outperforms ODC, too. Figure II-25 shows the cost of the better performance. The messaging overhead generated by
Spyglass is immense, it is by two magnitudes over ODC. Shuffling ODC produces a linear overhead only.

![Graph showing processed jobs per round]

**Figure II-24**: Number of processed jobs per round, with 50 colours (Basic Scale-Free network, generation time 200 rounds)

![Graph showing messages per round]

**Figure II-25**: Messaging overhead per round, with 50 colours (Basic Scale-Free network, generation time 200 rounds)

### 5.5. Effect of the Network Abstraction

The last set of experiments investigate how the network abstraction influences the results. I repeated the experiments with the same node and link counts, but with random networks. Results confirmed the general trends, however, sometimes with a less pronounced difference between Spyglass and Shuffling ODC. Figure II-26 shows an example, the unprocessed job curve which is basically the same as what I saw with the scale-free setting.

With random graph based network abstraction, the differences in the nature of cluster formation are more pronounced (the degree distribution of the nodes is homogeneous, i.e. random picks work better than in the scale-free case). Figure II-27 visualizes the number of clustered nodes along time. The fallback effect, discussed earlier, is less visible here.
6. Application

Shuffling ODC can be applied in areas, without specific restriction, where on-demand clustering is required.

The first significant application area of on-demand clustering is peer-to-peer networks and data grids. Here, nodes need to find cluster members for load sharing, or for data duplication.

Another significant application area is the field of overlay networks, for example a service overlay, where the entity in need for a service can locally look up service providers (inverse clustering), or a service with excess workload is able to locate partners for workload sharing and load balancing.

7. Conclusion

I discussed a novel direction, the Shuffling ODC algorithm, to improve the On Demand Clustering in self-organizing networks. Shuffling ODC allows link reconfiguration without direct gain in case an attempt for cluster expansion fails.
I compared Shuffling ODC via simulation with the original ODC algorithm and with another extension, Spyglass. Experiments confirmed that the two extensions, Spyglass and Shuffling ODC, provide improved clustering and load-balancing capabilities compared to the baseline algorithm. I observed improvement in the number of clustered nodes, clustering speed, the shape of the job processing and unprocessed job curves, primary and secondary overload characteristics, and total job processing times. Visual evaluation confirmed the balanced operation of the algorithms. Scalability was also demonstrated. Spyglass and ODC, while similar in some results, have different strong and weak points. The biggest advantage of Spyglass is its fast reaction time: clusters are extensively and very successfully formed from the arrival of the first trigger. The massive messaging overhead measured during the experiments may be reduced with caching, but not indefinitely. Shuffling ODC on the long term provides the same advantages as Spyglass, without the massive messaging overhead, but with slightly slower initial reaction time. The unique advantage of Shuffling ODC is its emergent self-shuffling effect which enables overloaded initiator nodes to overcome the boundaries of reaching only those new matches that were nearby already in the initial topology (topologic starvation).

Possible future research directions in the field include: experimentation with the density of the network, detailed analysis of the underlying physical routing, and design of new hybrids from the ideas behind Spyglass and Shuffling ODC.
Chapter III.
Vox Populi: Named Entity based Sentiment Mining from Texts

In this chapter I consider the problem of sentiment mining for named entities occurring in various natural language texts retrieved from the web. This task differs from the general opinion mining problem in its focus, which is on the opinion expressed towards a named entity instead of the totality of opinions expressed in the paragraph or document. I describe Vox Populi, a general adaptive approach for named entity based opinion mining. To make sentiment mining adaptive, I propose an automatic self-analysis methodology which is unsupervised in terms of not requiring an annotated reference corpus. Self-analysis includes automatic self-revision (flaw detection) of the term dictionary, and a set of completeness checking algorithms. Measurements on a large corpus showed that Vox Populi produces high precision (68-89%) and recall (67-87%) values.
1. Introduction

Sentiment analysis or opinion mining of various texts such as news articles, product reviews, blogs or the social media has received considerable interest in the last few years. With the expansion of social media, what people think or read is not just an interesting supplement, but an important input for business decisions, both at personal level and for companies.

Sentiment analysis tackles with the semantic orientation expressed in the texts. This can also be considered as a special case of document categorization, where the category of the document is based on its sentiment rather than its topic. Subtasks include: polarity analysis to determine the direction of the opinion (e.g. positive, negative, or neutral attitude), strength analysis to identify the intensity of the opinion, and subjectivity analysis to differentiate between objective and personal expressions. I tackle with the first two tasks in this chapter.

Opinion mining may happen at different levels, for example on the complete document, per blog comment, or per paragraph or sentence.

In means of tools, a large range of methods is known from simple term matching to complex grammatical analysis and machine learning based approaches. Term matching is often mentioned as bag-of-words model. The history of the term based direction starts with [Hatzivassiloglou and McKeown 1997] who used a list of seed words to determine the polarity of opinions in a sentence. [Turney 2002] used simple grammatical patterns in combination with terms. [Yi and Nasukawa 2005] applied natural language processing to identify the approximate relationship between the subject and the surrounding words. A bottleneck of term based approaches is the creation of the term dictionary; it essentially influences the stability and performance of the mining, hence needs to be large, clean and reliable. Machine learning directions try to overcome this burden by using non dictionary features. As an example, [Pang et al 2002, Pang and Lee 2004] used Support Vector Machines with non-dictionary features and a specific distance measure between sentences, primarily for subjectivity analysis. Another possibility to avoid the manual collection of terms is to use a semantic word network, for example WordNet, to propagate the polarity of a handful of manually selected basic seed words. SentiWordNet [Esuli and Sebastiani 2006], [Esuli and Sebastiani 2011] follows this approach. More recent works in the field concentrate on two main directions: (i) to identify the topic structure of the document in order to better understand the relation between the various sentiments expressed, and (ii) to identify sentiments in less formally structured texts such as in the social media. [Hu and Li 2011] presented results in the topic-centric direction. [Lin, He et al 2011], to better identify the topic, applied latent Dirichlet allocation. [Mei at al. 2007] tackle with the problem of sentiment mining from the social media. [Pang and Lee 2008] gave an extensive study about sentiment analysis, including challenges, techniques and the state of the art.

The goal of our research slightly differs from the common task of identifying the overall sentiment of a document or fragment; I focus on named entities, such as persons, brand names, places, and the opinions expressed towards them. I extract
information per named entity occurrence, and collect this information over time and from various sources.

The opinion information our model extracts from the text can be utilized in various ways. A few examples:

- Query of current standings. (What is the opinion about this product now?)
- Analysis over the time axis. (How did the opinion of people change in the last year regarding the product?)
- Analysis by sources or source types. (How is the product rated in the mainstream news and in private blogs?)
- Pair-wise comparison. (Which of the two brands is thought to be more positive? About which of them did people's opinion change more?)

The evaluation of the goodness of opinion mining results is a non-trivial task, as the reference answer, the humanly perceived sentiment, is of highly subjective nature, and may easily differ from person to person. In order to evaluate the goodness of a sentiment mining, a reliable, human-annotated reference corpus is required. Several authors use product rating databases, such as product reviews or movie ratings for this purpose, where the person writing the textual comment also provides a numerical rating (e.g. five stars). For our case, i.e. to identify named entity related sentiments in news/blog sources, this was not applicable. As a solution, I collected real news and blog items from the public media over eight months, and annotated a random sample of it manually, to serve as evaluation reference.

Most of the literature in the field assumes that the language is used by humans in a fixed, static way, i.e. changes in the sentiment attached to the word, as time passes, are not considered. Adaptivity is a basic concept in our point of view, I created an automatic self-analysis mechanism to detect potential flaws in the term database. The self-revision algorithm is unsupervised in terms of not requiring an annotated reference corpus in order to work. This mechanism first of all, enables the adaptive fine-tuning of the term database according to people's ever-changing word usage. Besides that, this algorithm also enables us to turn a rough initial term database, via self-revision rounds, into a clean term dictionary. An initial, rough term database may, for example, even be a machine translated SentiWordNet, if we want to execute sentiment mining for a language other than English. To demonstrate this line, I evaluated the system in a non-English (Hungarian) environment, over a machine translated and self-refined SentiWordNet term database.

Our work extends the state of the art in the following points:

- I propose a model for named entity centric opinion mining. I extend the bag-of-words algorithm with a vicinity and context based weighting to approximate the sentiment expressed towards the named entity.
- I propose an unsupervised self-revision mechanism to make the term dictionary adaptive through identifying flaws in it. The algorithm does not require an annotated reference corpus, and works with an arbitrary opinion mining algorithm that identifies opinions for small text segments.
- I propose an unsupervised completeness checking algorithm to investigate the completeness of the term database and of the test corpus.
I propose a mechanism to approximate the precision and recall of the evidence based extension of the term database. Evidence based extension means that non-term words that consistently occur in a polar environment become terms.

The rest of the chapter is structured as follows. Section 2 describes the general vision of the system, called Vox Populi, and provides an architectural overview. Section 3 tackles with the details of the of named entity centric opinion mining model. Section 4 presents the unsupervised self-analysis methodology, including the self-revision and completeness checking algorithms. In Section 5 I evaluate the opinion mining algorithm on a manually tagged test corpus; and evaluate the outcome of the self-analysis mechanisms manually, too. Finally, Section 6 concludes the work.

2. The Vox Populi Vision

The goal of Vox Populi is to extract the sentiment information expressed in natural language texts (collected from various web sources) towards the occurring named entities, such as persons, brand names, products, places, and so on. The viewpoint is open: there is no list of possible or watched named entities, instead, any entity that occurs in the text should be evaluated. In terms of specification, the followings were targeted.

- Documents are acquired from the web, from various sources including top news portals, other news, topic-specific portals, and sources containing subjective information such as private blogs.
- The goal is to identify the sentiment expressed towards the named entities occurring the text.
- The sentiment information is collected per named entity occurrence. The sentiment of the named entity occurrence is stored in a database along with the article, its source, and time stamp.
- An interactive search interface is provided to the users, over the database, to visualize and compare momentary values, trends over time, over sources, or between different entities.

The main constituents of Vox Populi, as a black box model or external view, are sketched in Figure III-1. The Data Acquisition module provides the system with new source data, i.e. web pages, from time to time. The Opinion Mining module processes the page: extracts the news/blog article from it, identifies the named entities and extracts the sentiment information for each named entity occurrence. The sentiment information is stored in the Opinion Database. Users access the data through an interactive Search Interface. In this chapter I concentrate on the opinion mining module, the others only contribute to the picture as a whole.

This named entity centric opinion mining approach extracts information that is ready to use for decision making. The sentiment information in each case relates to a concrete named entity, a timestamp, and a source. Searches may refer to a concrete standing, timely trends, source distributions, total and relative values, times of occurrences – all of which hold valuable information about the appearance and acceptance of the named entity or entities in question.
Figure III-1. External view of Vox Populi.

The more detailed, internal view of the system, including the self-revision process, is shown in Figure III-2. The flow of the data is two-fold. On the sentiment mining path, the system processes the web page as discussed above, through named entity recognition and term based sentiment calculation (with help of a Sentiment Dictionary), and saves the identified sentiment information to the Opinion Database, which users will search via the web search interface. The other data path realizes the self-analysis. Part of the incoming texts is stored into a database called Self-Evaluation Corpus. When sentiment mining is executed on this corpora, the results are coupled back for further evaluation by the self-revision and completeness checking modules. The Sentiment Self-Revision module tackles with the cleanliness and up-to-dateness of the term database. The Completeness Checker evaluates the solidity of the term database and the reliability of the self-evaluation corpora.

Figure III-2. Internal view of Vox Populi. Sentiment mining flow and self-analysis flow.

A pilot implementation of Vox Populi is in experimental use since September 2010. Data is acquired from public RSS feeds of various news and blog portals.
3. Named Entity Centric Sentiment Mining

This section describes our model for named entity centric sentiment mining. First I elaborate on the general process flow, then on the details of an extended bag-of-words algorithm to consider the environment of the named entity according to its paragraph context.

3.1. Process Flow

The flow of named entity (NE) centric opinion mining, shown in Figure III-3, consists of a named entity detection step, a linguistic preprocessing, and the calculation of the positive, negative and neutral weights of each NE occurrence.

![Diagram of process flow]

Figure III-3. General flow of the named entity based opinion mining.

In the named entity recognition phase named entities occurring in the text get tagged. Tagged words remain intact in the following steps of the process. For named entity recognition we used a dynamic approach which extends its known list of named entities with candidates extracted (learnt) from the texts. The candidate selection mechanism detects those new words that were experienced in various texts in uppercase form (evidence for being a NE) without being often experienced in lowercase form (counterevidence). Other named entity detection mechanisms could also be used.

The second step of the process is the linguistic preprocessing of the non-named-entity words of the text. I applied simple stemming and negation handling, but no deep syntactic or semantic analysis. The goal of stemming is to identify the dictionary form of words, so that the word can be, in the next step, compared to the terms in the term dictionary. Stemming, for example, means to bring words to its singular, present tense, declarative form. For negation handling, I used a simple heuristics: after experiencing a negation word, the next \( n \) words get a polarity inversion marker. I used \( n=2 \) in the tests, the exact value may be language or style specific.

In the third step, the text gets processed per paragraph. Each named entity occurrence in the paragraph is evaluated with help of a (self-revised) SentiWordNet.
Finally, the weights are combined and consolidated in a context-sensitive manner, considering the text length, number of occurrences and number of other occurring entities.

A possible criticism against the above model could be the lack of syntactic and semantic analysis. It was a deliberate decision not to include these steps, and keep the model as simple and generic as possible. Since the level of available deep grammatical processing differs from language to language, and different languages express opinions with different grammatical structures, our preliminary examinations indicated that more structural analysis would undermine the generality and language independence of the model.

Another comment could be the lack of cross-reference handling, i.e. anaphora resolution. It is true that the opinion mining module does not take cross-references into account. However, an advanced named entity detection mechanism could solve this problem by tagging hidden occurrences, anaphoras and cataphoras the same way as the explicit named entities.

3.2. The Sentiment Mining Algorithm

The idea behind our NE-centric sentiment calculation is to use different contextual cutouts, called frames, around the NE, and combine the results of the analysis of these frames. For example, the frame containing the direct vicinity of the NE gets higher weight in the combination than the frame containing the complete paragraph.

In Vox Populi I used two frames, the NE’s direct vicinity, and the whole paragraph (except for other named entities), to approximate the NE’s sentiment value. A context sensitively weighted sum of positive and negative values of the frames adds up the final sentiment of the NE occurrence.

The sketch of the named entity centric opinion mining algorithm is described in Algorithm III-1. First, the parameters, such as the length of the vicinity window and the combination weights of the frames are adjusted with a heuristics based on the named entity’s paragraph context (word count of the paragraph and the presence of other named entities). Then, the vicinity of the NE occurrence, i.e. nearby words, and the totality of words in the paragraph are extracted. A dictionary based matching is executed on both frames, also taking into account the possible negated state of the words. (The algorithm to calculate the frame weights is further detailed in Algorithm III-2. The dictionary for the term match based calculation is a derivative of SentiWordNet.) Then, the positive and negative sentiments extracted from the vicinity frame get combined with the positive and negative sentiments of the paragraph frame with the weights calculated in the first step. A neutrality value is also calculated, so that the positive, negative and neutral directions add up at least to a previously calculated expected amount.

---

Algorithm III-1. Sentiment mining for a NE occurrence.

```java
function evaluate(paragraph, no) {
  // calculate algorithm parameters:
  // pre, post = window length for the vicinity,
  // w1 = the weight for vicinity,
  // w2 = the weight for the total paragraph.
```
// expectedSentiment = expected total sentiments;
calculateParameters();

// extract the two frames
vic = a (pre, post) long window around ne;
p = all non-NE words in the paragraph;

// calculate frame sentiments
calculateFrameSentiments(vic);
calculateFrameSentiments(p);

// combine frame sentiments
ne.pos = w1 * vic.pos + w2 * p.pos;
ne.neg = w1 * vic.neg + w2 * p.neg;
ne.neu = max(0, expectedSentiment - ne.pos - ne.neg);
}

Algorithm III-2. Weight calculation for a frame.

function calculateFrameSentiments(frame) {
  pos = 0;  neg = 0;
  for each word in frame
    if word is a term
      if word is non-negated {
        pos += term.pos;  neg += term.neg;
      } else {
        pos += term.neg;  neg += term.pos;
      }
    frame.pos = pos;
  frame.neg = neg;
}

SentiWordNet, by itself, only contains positive and negative values for terms. I added the neutral dimension to our sentiment model to materialize a new concept, the expected total sentiment amount of a paragraph, which I defined as a value proportional to the length of the text. Each n words are expected to hold at least a given amount of sentiment information; the neutral polarity marks when the target is not reached by positive and negative values solely.

The concept of expected total sentiment also enables us to calculate the normalized positive and negative weights of the occurrence. (Normalized values are calculated by dividing the original values with the sum of positive, negative and neutral weights.) This, when users search the database, helps differentiating between cases where the opinion happens to be less pronounced or less central (e.g. a long paragraph shortly also mentioning the product) and cases when the opinion is extensively present and strongly attributed to the product. I found that these normalized values both theoretically and practically hold more useful information than the pure positive and negative weights themselves.

The calculation of the algorithm’s parameters, such as the frame lengths, the combination weighting of the frames, and the expected total sentiment amount is achieved with a simple heuristics. The heuristics is based on previous offline measurements, where I found that a typical paragraph is 50 words long, and the vicinity of the named entity is typically a \((p-4, p+2)\) window where \(p\) is the position of the NE. These default values are slightly increased or decreased, depending on the
length of the text and the number of other named entities (competitors) in the paragraph. Algorithm III-3 shows the algorithm I used.


```plaintext
function calculateParameters(p) {
    // initial values
    w1 = 1.5; w2 = 1;
    pre = 4; post = 2;
    expectedSentinent = p.words / 20;

    // reference: paragraph: 50 words, 1 NE
    // adjust parameters when longer text or more than one NEs present
    if p contains no other NEs {
        if (p.words > 50) {
            // for long paragraphs, expand the window
            // and decrease the paragraph's weight
            pre = p.words / 11; post = p.words / 20;
            w2 = 0.8;
        }
    } else {
        // ratio means the number of words per NE
        ratio = p.words / p.competitors;
        if (ratio > 50) {
            // if there are considerably more words than
            // NEs, do what the first branch did
            pre = ratio / 11; post = ratio / 20;
            w2 = 0.8;
        } else {
            // if competitors consume too many words
            // adjust the vicinity frame and its weight
            if (competitors > 3) {
                pre++; post--;
                w1 = 2.0;
            }
            if (ratio < 20) {
                pre--; post;
                w1 = 2.0;
            }
        }
    }
}
```

Multiple occurrences of the same named entity in a paragraph are summed up (with re-calculating the neutral strength). Multiple occurrences of the same NE in an article are simply summed.

### 4. Self-Analysis Methodology

In this section I discuss an unsupervised self-analysis methodology for the heart of the system, the term dictionary. First, I propose a self-revision algorithm to identify potential flaws in the dictionary. Afterwards, I discuss how the same basic idea can be used for two other purposes: to check the completeness of the term database and the corpus, and, finally, to estimate the goodness of an evidence-based dictionary extension.
4.1. Unsupervised Term Self-Revision

4.1.1. Goals and Prerequisites

The goal of the self-revision algorithm is to identify terms that behave unexpectedly, without using explicit knowledge, such as annotated corpus, what is expected.

As prerequisite, I assume the presence of a large enough and representative enough textual corpus (non-annotated corpus) to run the measurements on. Besides that, I also assume that the sentiment mining algorithm is of local nature, i.e. identifies opinions expressed in smaller sections (fragments) of the text rather than identifying the totality of opinions in the complete document. Fragments may also be overlapping, such as in the case of Vox Populi.

4.1.2. Algorithm Outline

The idea of the algorithm is to check what happens if we remove a small percentage of words from the term database. If the amount is small enough, the difference on the results shall not be significant. The outline of the algorithm is the following.

- Remove a small percentage of terms \( T \) from the knowledge base, resulting in a nearly-complete knowledge base \( X \).
- Execute sentiment analysis on the self-evaluation corpus with \( X \) as a term dictionary. Members of \( T \) do not take part in the sentiment calculation.
- Register the occurrence context of \( T \) terms, i.e. the polarity of the containing document fragment.
- The encountered contexts are compared with the expectations. The hypothesis is that the direction of the term, normally, should not contradict the typical direction of the containing context. For example, positive words mostly occur in positive or neutral contexts, and not in negative ones. Terms where a contradicting trend is found, i.e. a supposedly-positive term mostly encountered in clearly negative fragments, are marked as suspicious.
- Suspicious terms are selected for automatic or manual cleaning.

4.1.3. Formal Model

The formal model of the algorithm is as follows. The environment consists of documents \( (D) \); documents consists of fragments \( (F) \), and fragments contain words \( (W) \). Fragments may also contain other elements not specified here, e.g. grammatical information, or distances.

\[
D = \{d_i \} \quad \text{Documents} \\
F = \{f_i \} \quad \text{Fragments} \\
W = \{w_i \} \quad \text{Words} \\
d_i = \{f_j \} = \{\{w_k \ldots \}_j \}
\]

The opinion mining algorithm \( (m) \) identifies the sentiments for a fragment. Alternatively, just by expanding the definition of the fragment, the mining algorithm
can also be formulated as a function of words and other information, resulting in
sentiments.

\[ m : F \rightarrow (R+)^3 \]  \hspace{1cm} \text{Mining algorithm}

\[ m : (W^n \ldots) \rightarrow (R+)^3 \]  \hspace{1cm} \text{Mining algorithm 2nd form}

The removal of terms from the term dictionary has the same effect as removing the
matching words from the fragment. The set of words that match with a removed term
is denoted with \( W^* \). I model the removal of terms with the removal of matching words
from the fragment for the self-revision loop's sentiment mining. The modified mining
algorithm \( m^* \) is identical to the original \( m \) except that it executes on the reduced set
of words. The advantage of this model is that we do not need to care about the change
of the term dictionary.

\[ T = \{ t_i \} \]  \hspace{1cm} \text{Terms to neglect}

\[ W^* \in f = \{ w \mid \text{stem}(w) \in T \} \]  \hspace{1cm} \text{Removed words}

\[ m^* : (T, W^n \ldots) \rightarrow (R+)^3 = m(W^n - W^* \ldots) \rightarrow (R+)^3 \]  \hspace{1cm} \text{Mining alg. with removed words}

Let us define the direction of a sentiment annotation as a mapping to an integer,
where the values +2, 0, -2 mean ‘positive’, ‘neutral’ and ‘negative’, and +1, -1 mean
‘rather positive’ and ‘rather negative’.

\[ \text{dir} : (R+)^3 \rightarrow \{-2, -1, 0, +1, +2\} \]  \hspace{1cm} \text{Direction of sentiments}

The first assumption I took is that if \( W^* \) is small enough, the difference between
the direction of \( m \) and \( m^* \) will not be significant. Although, I am unable to formally
prove the truth of this assumption for the general case, heuristic proof is possible.
Human communication is typically of redundant nature. Important elements are not
expressed with single words; instead, people use more than one adjectives to emphasize
the message, give additional details in the next sentence, or repeat the same statement
with different words later. The use of a single message holder word throughout the
fragment, or to excessively repeat a single sentiment holder, is highly non-typical. I
formulate the assumption in a probabilistic form, defining the trend but allowing
exceptions.

\[ \lim_{W^* \rightarrow 0} p(\text{dir}(m) = \text{dir}(m^*)) = 1 \]  \hspace{1cm} \text{Assumption of redundancy}

Our second assumption is that words mostly occur in consistence with the direction
of the containing fragment, i.e. negative words mostly occur in non-positive fragments,
and vice versa. In other words, the number of occurrences in a contradicting
environment is much smaller than the number of occurrences in a non-contradicting environment, where contradiction means that the sign of one party is positive and the sign of the other is negative. (Any word can occur in a neutral environment, and the same way, polar words can also occur in non-polar fragments.) The assumption is formulated as the limes of the relative error, i.e. ratio of contradicting and non-contradicting encounters. Again, I give a probabilistic statement, and not a general rule for all words.

\[
error(w, f) = \begin{cases} 
1 & |dir(w) \cdot dir(f)| < 0 \\
0 & |dir(w) \cdot dir(f)| \geq 0 
\end{cases}
\]

\[
errorRate(w) = \frac{\sum_{f \in D \mid w \in f} error(w, f)}{\sum_{f \in D \mid w \in f} -error(w, f)}
\]

\[
\lim_{|D| \to \infty} p(errorRate \approx 0) = 1
\]

Assumption of consistency

If the two above assumptions are true the algorithm shown in Algorithm III-4. can be used to highlight terms that occur in environments contradicting the expectations.

I propose to register not only the occurrence of the term but also its surface form \(w\) (the word in the original text that matched with the term). This helps in identifying language-specific nuances and unexpected side-effects of stemming. For example, a mis-configured morphologic engine may over-stem the word, i.e. remove important modifiers or affixes, for example by transforming ‘unsuccessful’ to ‘successful’ or ‘faithless’ to ‘faith’.

Algorithm III-4. The self-revision algorithm

```plaintext
// self-revision on documents D using mining method m
// and left-out terms T
function selfRevision(D, m, T) {
    // calculate sentiments for all fragments
    for each fragment f in D {
        dirSeen = dir(m*(T, f));
        for each word w in f {
            if w matches t ∈ T {
                // register the occurrence of the term
                // in the occurrence registry
                registerOccurrence(w, t, dirSeen);
            }
        }
    }
    // summarize the registered occurrences
    for each (w,t) pair in the registry {
        t.dirSeenTypical = calculateTypicalDirectionSeen(t);
    }
    // mark suspicious terms
    for each (w,t) pair in the registry {
        if contradict(dirExpected, dirSeenTypical) {
            markSuspicious(t);
        }
    }
}
```
4.1.4. Matrix View of the Results

The self-revision algorithm categorizes the watched terms based on their expected and encountered occurrence contexts, i.e., the result is an expected and a typically encountered direction for each term.

The summary of different expectations and encounters can be visualized in a matrix, showing the number of terms in each expectation-encounter class. Figure III-4 shows an example for this expectations-encounters (EE) matrix. The cells containing suspicious terms are marked with grey; I use the phrase ‘grey area’ to refer to them. In the example below, most terms occur in harmony with the expectations, i.e., the direction of the typical occurrence is either the same as the term’s direction, or one of them is neutral. The only suspicious encounters in this concrete EE matrix are the terms in the top right cell; these two words were expected to be negative but instead of occurring in negative or neutral contexts, they were mostly encountered in clearly positive fragments.

![Figure III-4. EE (expectations-encounters) matrix. Cells showing contradiction are marked with grey. In this example, there are two suspicious terms (top right cell), both occurred mostly in positive environments while they are supposedly negative terms.](image)

Theoretically, the clearness of the term knowledge base can be characterized by the percentage of entries in the grey area; and the grey area of a clear knowledge base should be empty. However, we must note that the assumptions taken previously may not always be true for each single word and fragment. Hence, the algorithm may falsely put certain words into the grey area (false positives), causing a base error.

4.1.5. Handling the Results

Words marked as suspicious by the algorithm may be processed either manually or automatically. To decide which direction to follow, we must not forget that a word, when in the grey area of the EE matrix, is a potentially but not necessarily mis-annotated term. I found that false positives were typically rare words, that, for some short but intensive time period, occurred in the context of some actual, strongly polar news stream (e.g., reports on natural disasters, such as the tsunami or the red sludge flood), and the strong sentiment of the news itself outweighed, suppressed the polarity of the term. For true positives, the following reasons were found: (i) wrong dictionary form, for example over-stemming, (ii) the actual surface form of the word has a second meaning which differs from the meaning of the dictionary form, (iii) the word is part of a common expression modifying its current sentiment, or (iv) the sentiment is just wrong, i.e., general dictionary or machine translation induced dictionary error.
The automatic handling of the suspicious words means their removal from the term database. The manual handling means that a human supervisor is able to filter out false positives before removing the terms. For concrete cases, a tradeoff between the negative effects of removing false positives and the cost of human work needs to be found. The completeness analysis of the term database, discussed in the next chapter, may facilitate this decision.

4.2. Completeness Analysis

The idea behind the self-revision algorithm, with a slight modification, can also be applied for other objectives. I discuss in this section how it can be utilized for the completeness analysis of the term database and also of the (non-annotated) test corpus.

4.2.1. Completeness of the Corpus

It is essential to execute the measurements on a corpus that is complete enough to bring sensible results. The term ‘complete corpus’ means that it is sufficiently large so that an extension would not bring significant advantages, and sufficiently balanced so that there are no significant bursts of imbalance that could bias the measurement results.

I propose to examine the completeness of the corpus by analyzing the nature of the encountered new terms over time.

The analysis of new terms means that after each \( n \) steps of the self-revision algorithm, statistics about the newly encountered terms get calculated (by newly encountered terms we mean those terms on the watch list that did not occur earlier but were encountered in the last \( n \) steps). For a complete corpus, the followings are expected.

- The number of new terms systematically decreases with time. This also means the utility of processing further documents to bring new terms approaches zero.
- The polarity distribution of new terms stays quasi constant over time, hence the amount of positive, negative, and neutral items is generally balanced (unless some external reason such as a natural disaster or a world-wide sport event is known to be present).

The resulting term distribution chart can be evaluated manually or in an automated way. As a result, local imbalances are identified or closed out, and the utility of extending the corpus with new items is also approximated.

4.2.2. Completeness of the Term Database

The dynamics of the EE matrix, such as its change along time or different parameters, may also give us an impression about the completeness of the term database. By completeness I mean performance stability and descriptive completeness.

I propose two tools for completeness check on the term database: (i) analysis of the EE matrix over time, and (ii) analysis of the EE matrix at different omission rates (\( T \) to \( T+X \) ratios).

The timely analysis of the EE matrix means that the matrix is recalculated after each \( n \) steps. Regarding the trends, the following is expected:

- The trends, i.e. the distribution of the terms in EE cells stabilize with time.
High changes in the distribution are acceptable in the beginning of the measurement, because the number of processed documents is low at this point, and each new document may inject significant amount of new information. However, supposing a complete enough corpus, when the timely dynamics of EE remains high with time, it suggests that a large portion of the watched terms keeps changing its typical occurrence context, meaning that some influencer, i.e. either the term database or the opinion mining algorithm itself, is unstable. Small changes in the EE cells with time suggest that the last few documents did not modify the situation significantly, i.e. the results are stable.

Figure III-5 shows an example for the timely analysis of the EE matrix. I visualized the non-contradictive percentage for positive and negative cases over time (the percentage of terms whose typical occurrence context does not contradict the expected polarity). After an initial phase the ratios become stable, and only minor deviations can be observed.

![Figure III-5. Segments of the EE matrix over time (10 % omission rate). Negative OK denotes the ratio of correctly tagged negative occurrences over all negative occurrences (technically, the sum of the first three cells of the top row over the total top row). Positive OK means the same for the positively attributed NEs. The ratios stabilize with time.](image)

The idea of omission analysis is to compare how an increment in the omission rate ($T_{size}$) influences the EE matrix. For example, how does the EE matrix change with a two times or four times as high term omission rate. The following is expected:

- Higher rates cause the 'precision' and the 'recall' of the EE matrix to decline. (With a higher omission rate other sentiment holders of the context tend to disappear too, so the frame becomes less polar and less reliable.)

The omission analysis may give us an impression how the reduction of the term database would influence the system. If the measured decrement is slow, we may assume that the database is sufficiently complete.

### 4.3. Evidence Based Dictionary Extension

The idea of evidence based (typical occurrence context based) polarity estimation can also be used for extending the term dictionary with additional items. This means that the dictionary gets extended with non-term words that systematically occur in the corpus in a clear-polarity context. However, this evidence based extension is prone to errors of various kinds. In order to work reliably, we would need to ensure that the occurrence of each word is representative in means of topic diversity and actual topic
polarity. An example for the traps is the topical bias, where a word that is not a real sentiment holder occurs in definitely polar environments just because the topic it relates to is mentioned in a polar way for some time. Topically biased words cannot be filtered out by considering the number of occurrences or the diversity of the sources either: different news agencies often repeat the same information, so the same topical bias is present everywhere, even though the number of occurrences is high and source-wise diverse. Topically biased words are typically occupation names and field-specific terminology words.

Precision and recall are two important metrics to describe the success of a dictionary extension. I provide a tool to estimate them.

I estimate the precision and recall of the evidence based extension with the measured quasi precision and ‘quasi recall’ of the self-revision algorithm. This is valid, because the self-revision algorithm, basically, works as an evidence based learner. I call the self revision algorithm’s metrics ‘quasi precision’ and ‘quasi recall’ because they do not depict real human impression (instead, the reference is the original polarity of the word).

5. Experimental Evaluation

This section discusses the evaluation of Vox Populi’s named entity centric sentiment mining algorithm and the self-analysis methodology. First I describe the corpus and the evaluation environment, then continue with the experiments and their outcome.

5.1. Corpus

As the opinion mining algorithm concentrates on real news items in a named-entity centric manner, the well-known sentiment corpora was not applicable (either because they do not contain news or because they are not annotated in a named entity centric manner). Furthermore, I wanted to test the self-revision algorithm on a machine translated dictionary, so decided to execute the experiments in a non-English environment.

We created a news and blog corpora by collecting the entries of public RSS feeds of Hungarian news and blog portals between June 2010 and April 2011. I used a subset of that, the material collected form 8 feeds between September 2 2010 and April 20 2011 for the experiments. (The June-August material was used for the calculation of various statistics; I excluded this part from the actual measurement because they would not count as unseen text.)

Table III-1 shows the overview of the corpus, the sources and the number of collected documents. A spring based content identification model was used for extracting the article text from the complete web page, which typically also contained additional elements, such links to other columns, related articles, recommendations, and adverts.

<table>
<thead>
<tr>
<th>Source</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

78
Table III-1. Overview of the corpus.

A small sample of the corpus, namely 230 randomly selected named entity occurrences (the paragraph containing the NE) underwent manual sentiment annotation, to serve as a reference for measurements. Two independent human annotators were used (a male and a female), they were asked to determine the direction of sentiment expressed towards the highlighted named entity based on the text of the paragraph. (I did not collect information about the sentiment of the further named entities in the paragraph.) The human annotation results correlated with each other. I used the male annotator’s values for this section, because it shows slightly more divergence from the mining algorithm’s answer.

5.2. Environment

I used a Sun Fire X2200 M2, Dual Quad Core AMD Opteron, 8GByte RAM server with 8 MByte RAM and UNIX as operating system for the measurements. Both Vox Populi and the self-revision algorithm were implemented in Java6.

5.3. Evaluation of the Named Entity based Opinion Mining Algorithm

In order to evaluate the goodness of Vox Populi’s named entity centric opinion mining algorithm, I compared the sentiment assigned by the mining algorithm with the sentiment determined by the human annotator on the reference sub-corpus. Table III-2 shows the numeric results of the comparison, while Figure III-6 displays the same information graphically. Most of the occurrences take place on the diagonal of the matrix, meaning that the human-assigned and the algorithm-assigned tags coincide. Deviations from the diagonal mean a discrepancy between the values.
Table III-2. Comparison of human tags and the algorithm’s tags on the annotated reference sub-corpus.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>23</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>16</td>
<td>12</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>7</td>
<td>69</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>10</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>23</td>
<td>38</td>
</tr>
</tbody>
</table>

| Human     | 29 | 34 | 103| 38 | 26 | 230|

Figure III-6. Distribution of human and algorithm assigned tags. High values in the diagonal mean that most of the occurrences were tagged equally by the human and the algorithm.

I used different masks to evaluate the matrix. The first mask, named tolerant (see Table III-3), selects those cells where the difference between the human and machine value is not greater than one. This mask tolerates a slight divergences, i.e. accepts ‘neutral’ as ‘rather positive’ and vice versa. The second mask, called strict, shown in Table III-4, select those cells that clearly match the direction, i.e. only accepts positive pairs for positive values, negative ones for negatives and only neutrals for neutrals. Finally, a user experience maximizing mask (Table III-6) was created to select those cells where the cells show no contradiction to the human’s opinion. Note that this mask is, unlike the first two, asymmetric.

Table III-3. Tolerant mask.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
I calculated the precision and recall of the mining algorithm along these masks. Table III-6 summarizes the values. With the tolerant mask which does not give penalty for unsure values (-1,+1), 84% of the entities are placed correctly. Both the precision and the recall of the model are around 83%. With the strict mask, not tolerating errors in unsure items, all metrics show weaker values: 68% correctly tagged NEs, 67% recall and 68% precision. With the most permissive mask, namely with user experience centric, the 206 occurrences out of the 230, i.e. 89% of the entities, are considered as correct. This means 89% recall and 87% precision.

<table>
<thead>
<tr>
<th>Mask</th>
<th>Accepted NEs</th>
<th>Accepted NE %</th>
<th>Recall</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tolerant</td>
<td>194</td>
<td>84.35%</td>
<td>83.47%</td>
<td>83.50%</td>
</tr>
<tr>
<td>Strict</td>
<td>157</td>
<td>68.26%</td>
<td>67.44%</td>
<td>68.51%</td>
</tr>
<tr>
<td>Non-contradictory</td>
<td>206</td>
<td>89.57%</td>
<td>89.15%</td>
<td>87.38%</td>
</tr>
</tbody>
</table>

Table III-6. Results with different masks. Acceptance of an NE means that its tag is accepted, i.e. considered as correct.

5.4. Evaluation of the Knowledge Self-Analysis Algorithm

I analyzed a machine translated version of the SentiWordNet database on the September-April corpus. The term database contained 13068 terms, out of which 6928 terms had zero sentiment weights (both their positive and negative weights were 0). I used the remaining 6140 sentiment holders for the experiment. The expected direction of the term was modeled with the difference between the positive and negative strength. If the difference is smaller than 0.1 the term was considered to be neutral. Otherwise the expected direction was the larger of the polarities.

Various omission rates were examined from 1% to 20%.

The following factors were measured repeatedly throughout the mining process on the corpus: (i) the current state of the EE matrix, (ii) the number of encountered new terms, and (iii) the complete occurrence statistics of the encountered terms. At the end of the measurement, terms belonging to the grey area of the EE matrix were evaluated manually.
Figure III-7 shows the dynamics of encountering new $t \in \mathbb{T}$ terms while processing the corpus. The ratio of the expected polarities is more or less constant over time, around 50% of the terms are expected to be neutral, and 25-25% to hold positive and negative sentiments, respectively. The constant distribution suggests that the corpus is not affected by local imbalances. The shape of the curve suggests that with time fewer and fewer new terms are encountered, i.e. the corpus is large enough for the measurement (further documents could not bring significantly more additional new terms). Similar curve shapes were observed for other omission rates as well.

![Graph showing the dynamics of encountering new terms while processing the corpus.](image)

Figure III-7. Number of (distinct) terms in the encounter registry over time, per expected polarity, with term omission rate = 20%. With time, fewer and fewer new terms occur in the texts, suggesting that the corpus is large enough for the measurements. The chart shows rounded values with a +/-3 rounding error.

Approximately 50% of the watched terms were encountered in the corpus during the experiment. Manual analysis showed that non-occurring terms are topic-specific or highly informal forms.

Table III-7 through Table III-11 show the final EE matrix at 1%, 2.5%, 5%, 10% and 20% omission rates, respectively. Terms in the grey area are marked suspicious. The algorithm selected consistently less than 9% of the occurring terms for further evaluation, i.e. 4.5% of the watched terms. Checking 4.5% of the terms is 22 times less work than checking the complete term dictionary.

<table>
<thead>
<tr>
<th></th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table III-7. EE matrix, 1% omission rate. 26 terms encountered.
Table III-8. EE matrix, 2.5% omission rate. 63 terms encountered.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>7</th>
<th>0</th>
<th>16</th>
<th>0</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Table III-9. EE matrix, 5% omission rate. 125 terms encountered.

<table>
<thead>
<tr>
<th></th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>13</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
<td>28</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>15</td>
<td>1</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

Table III-10. EE matrix, 10% omission rate. 256 terms encountered.

<table>
<thead>
<tr>
<th></th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24</td>
<td>35</td>
<td>0</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>24</td>
<td>69</td>
<td>0</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>8</td>
<td>45</td>
<td>1</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>

Table III-11. EE matrix, 20% omission rate. 543 terms encountered.

<table>
<thead>
<tr>
<th></th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>49</td>
<td>65</td>
<td>0</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>60</td>
<td>157</td>
<td>6</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>22</td>
<td>76</td>
<td>0</td>
<td>27</td>
<td></td>
</tr>
</tbody>
</table>

Although it is not a goal of the algorithm to truly determine the direction of a term from its typical occurrence context, I measured what would happen if we accepted the typical experienced context as real direction. Figure III-8 visualizes these quasi-precision and quasi-recall values for the different omission rates, using the strict mask as acceptance criterion. Omission rate = 1% is not shown because the total amount of encountered terms is too low there (3 negative and 9 positive, 26 terms total) for making sensible statements. When the amount of omitted terms is increased, the precision and the recall of the model slightly but constantly decreases. This suggests that too high omission amounts are not recommended for the self-revision algorithm, because they produce less reliable results by shifting the outcome towards the 'no-decision' (i.e. neutral) direction.

Figure III-9 shows the same values but with the user experience mask as acceptance criterion. The trends are similar but with significantly higher actual values.

An implication is that the typical encountered direction is not a reliable predictor for the real direction in general (i.e. with the strict mask precision and recall are below 50%), but can be used sensibly for consistency check (i.e. with the user experience mask precision and recall are over 90% for the expected-to-be-positive and negative terms, unless the size of the removed term set is too high). As discussed earlier, the values shown in Figure III-8 can also be used as an estimating the precision and recall of the resulting database if we try to extend the term dictionary with new, unknown words based on their consistent occurrence contexts, without further filtering.
Figure III-8. Precision and recall of the encountered vs. expected directions along different omission rates, with the strict mask. The higher the amount of omitted terms, the worse the precision and recall becomes.

Figure III-9. Precision and recall of the encountered vs. expected directions along different omission rates, with the user experience mask. The higher the amount of omitted terms, the worse the precision and recall becomes.

6. Application

Opinions, with the expansion of social media and the world-wide web, are becoming more important than ever before. The Vox Populi model of named entity centric opinion mining produces results that are ready to utilize for business decisions, both as an individual and on corporate level.

A pilot version of Vox Populi is in experimental use at Magyar Telekom Nyrt. since September 2010.

7. Conclusion

In this chapter, I proposed an adaptive model for named entity centric sentiment mining; with an extended bag-of-words algorithm which considers the named entity’s direct environment as well as its broader context. I proved with experiments on a
Hungarian corpus collected from real RSS feeds over eight months that the model is applicable; I found 68-89% recall and 68-87% precision depending on the evaluation settings.

Above that, I proposed an unsupervised self-analysis methodology for the examination and revision of the term dictionary.

As part of the self-analysis methodology, an algorithm for automatically revising the term database without the need for an annotated corpus was also discussed. I gave a formal definition of the model and elaborated on various aspects of the algorithm. I applied the algorithm for cleaning a translated term dictionary, but other areas of use, such as reviewing automatically generated or extended term databases, are also possible. I proved with experiments that the algorithm is applicable for cleaning the term database.

As a side effect I also showed that guessing a term’s polarity based on the direction of its typical occurrence context gives low precision and recall results, because the occurrence context tends to be more neutral and less consistent than needed. The quasi precision and recall values generated by the self-revision algorithm can be used as an upper bound for approximating the precision and recall of a system where the term dictionary gets extended with new items based on the item’s typical occurrence context direction, without further filtering.
Summary

In my dissertation I discussed three fields for novel, distributed, self-organization and cooperation based machine intelligence models and algorithms. This section summarizes the results achieved.

In the chapter first chapter I discussed a novel, open collective-adaptive multi-agent model. The most important results are the following:

- (OLAKO model) I proposed a complex model, an online learning based collective-adaptive multi-agent society, for the problem of adapting to dynamic environmental requirements. OLAKO synthesizes online learning, an emergent knowledge sharing mechanism between agents, and an autonomous knowledge self-optimization mechanism. I proposed a formal framework to analyze the model, along with a set of statements that the formal description implies.

- (OLAKO Knowledge Self-Optimization) I extended Least-Squares Temporal Difference learning with the concept of using online, dynamically created features. I proposed the model of hierarchical feature unification by grouping. I proposed a knowledge self-optimization algorithm to dynamically create and remove features at runtime. I proposed a knowledge de-optimization algorithm to transform the knowledge back to the original, non-optimal form, containing the initial features. I showed with simulations that the knowledge self-optimization results in 11-18% faster learning on a showcase problem.

- (OLAKO Knowledge Sharing) I proposed a model and an algorithm for importing an externally, independently developed Least-Squares Temporal Difference learning based knowledge block to an agent. I proposed knowledge import strategies, consisting of an agent coupling mechanism, an import decision making mechanism and a knowledge integration mechanism. I showed with simulations on a showcase problem that knowledge import is able to improve the performance of agents (+46-84% non-failure rate, +2% success rate). I described the emergent effect of society-wide knowledge import. I showed with simulations that society-wide random knowledge import has an emergent negative effect, in a showcase problem it impairs the learning ability of agents, their performance does not improve with time. I showed with simulations that society-wide smart knowledge import, where the acceptance criterion depends on an estimated utility, has a positive emergent effect: it results in 21% higher success rates and 30% higher non-failure rates on a showcase problem, compared to the case without knowledge import.

In the second chapter I discussed a novel, improved on demand clustering algorithm for self-organizing mobile networks. In the chapter first chapter I discussed a novel, open collective-adaptive multi-agent model. The most important results are the following:

- (Topologic Starvation) I described the phenomenon of topologic starvation, an emergent negative effect of using only direct-gain link reconfigurations in the self-organization mechanism. I described how the baseline ODC algorithm facilitates the topologic starvation with time.

- (Shuffling ODC) I extended the baseline ODC algorithm with the concept of link reconfiguration without direct gain. The resulting algorithm, Shuffling ODC, injects minor modifications into the adjacent topology when an attempt for cluster expansion is unsuccessful. I showed with simulations that Shuffling ODC,
by avoiding the topologic starvation, results in 25-30% more clustered nodes, 17-23% more processed jobs per round, 20-30% improvement in job sharing, and 30-50% less job processing time, compared to ODC.

- **(Communication Overhead of Shuffling ODC)** I showed that the messaging overhead of Shuffling ODC is linear with the messaging overhead of ODC. I showed that the messaging overhead of Shuffling ODC and ODC are equal when the lookup is always successful.

- **(Locality Preservation in Shuffling ODC based Load Balancing)** I showed with simulations that Shuffling ODC, when used for load balancing, does not break the rule of locality. The resulting workload sharing distance remains 56-63% smaller than the increment in the cluster size.

In the third chapter I discussed a novel, adaptive, named entity centric sentiment mining model. The most important results are the following:

- **(Vox Poluli)** I proposed a model for adaptive, named entity centric opinion mining. I proposed building blocks and the data flow paths for the system. I showed with experiments that the model is applicable for the Hungarian language.

- **(Named Entity based Sentiment Mining Algorithm)** I proposed an algorithm to identify the sentiment expressed towards a named entity that occurs in a text, with a bag-of-words model extended with basic linguistic analysis (negation handling and word stemming) and with the use of weighted contextual frames around the named entity. I proposed an algorithm to dynamically adjust the runtime parameters, such as the frame lengths, of the algorithm. I showed with measurements on a manually tagged reference corpus that the algorithm produces 68-89% precision and 67-87% recall.

- **(Term Self-Revision Mechanism)** I proposed an unsupervised self-revision algorithm for the term dictionary, to identify potential flaws in it, without the need for an annotated reference corpus. I gave a formal definition of the algorithm and the related assumptions. I proved that if the assumptions are true, the algorithm locates the flaws in the dictionary. I proposed a data structure, called Expectations-Encounters (EE) matrix to handle the totality of the results.

- **(Completeness checking)** I proposed an unsupervised method to identify incompleteness in the term database and in the (non-annotated) corpus that is used for the measurements. I showed with measurements that the corpus I used during the evaluation is complete. I showed with measurements that the SentiWordNet based term dictionary I used for Vox Populi is complete.

- **(Precision-Recall Estimation for an Evidence based Dictionary Expansion)** I proposed an algorithm to estimate the precision and the recall of the evidence based extension of the term dictionary. I showed with measurements that the evidence based dictionary extension is not applicable for Hungarian with the Vox Poluli opinion mining model because of its low (43-44%) precision and recall.
References


CASCADAS Deliverable D3.2. 2007. Report on rule-based modules for unit differentiation using cross-inhibition and/or resource competition. IST CASCADAS Project
References


References


References


Mei I., Mi H., Quiao J. 2007. Sentiment Mining and Indexing in Opinion ICWSM’2007 Boulder, Colorado, USA


