Using Markov logic for automation of mobile network management

Juha Niiranen
juha.niiranen@helsinki.fi

April 15, 2016
The demand for mobile services is increasing constantly and mobile network operators need to significantly upgrade their networks to respond to the demand. The increasing complexity of the networks makes it impossible for a human operator to manage them optimally. Currently the network management operations are automated using a pre-defined logic. The future goal is to introduce cognitive network management functions which can adapt to changes in the network context and handle uncertainty in network data.

This thesis discusses using Markov Logic Networks for cognitive management of mobile networks. The method allows uncertain and partial information and makes it possible to consolidate knowledge from multiple sources into a single, compact, representation. The model can be used to infer configuration changes in network parameters and the model parameters can be learned from data.

We test the method in a simulated LTE network and examine the results in terms of improvements in network performance and computational cost.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acronyms</td>
<td>vi</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>2 Mobile Networks and SON</strong></td>
<td>3</td>
</tr>
<tr>
<td>2.1 Mobile networks overview</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Mobile network management</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Measurements</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Self-Organising Networks (SON)</td>
<td>6</td>
</tr>
<tr>
<td>2.5 Cognitive network management</td>
<td>8</td>
</tr>
<tr>
<td><strong>3 Markov Logic Networks</strong></td>
<td>10</td>
</tr>
<tr>
<td>3.1 Definition</td>
<td>10</td>
</tr>
<tr>
<td>3.2 Inference</td>
<td>11</td>
</tr>
<tr>
<td>3.2.1 MAP Inference</td>
<td>11</td>
</tr>
<tr>
<td>3.2.2 Marginal and Conditional Probabilities</td>
<td>12</td>
</tr>
<tr>
<td>3.3 Weight learning</td>
<td>13</td>
</tr>
<tr>
<td><strong>4 MLN model for network management</strong></td>
<td>16</td>
</tr>
<tr>
<td>4.1 Model</td>
<td>17</td>
</tr>
<tr>
<td>4.1.1 Predicates</td>
<td>17</td>
</tr>
<tr>
<td>4.1.2 Formulae</td>
<td>18</td>
</tr>
<tr>
<td>4.1.3 Model composition</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Inference and Learning</td>
<td>20</td>
</tr>
<tr>
<td>4.2.1 Weights</td>
<td>20</td>
</tr>
<tr>
<td>4.2.2 Inference</td>
<td>21</td>
</tr>
<tr>
<td><strong>5 Experimental results</strong></td>
<td>22</td>
</tr>
<tr>
<td>5.1 Setting</td>
<td>22</td>
</tr>
<tr>
<td>5.2 Measured quantities</td>
<td>23</td>
</tr>
<tr>
<td>5.3 Network parameters</td>
<td>25</td>
</tr>
</tbody>
</table>
5.4 Model ................................................................. 25
  5.4.1 Variables and predicates ................................ 25
  5.4.2 Formulas and weights .................................... 27
5.5 Process ............................................................. 28
5.6 Results ............................................................. 29

6 Discussion .......................................................... 32

7 Summary and Conclusions ....................................... 34
Acronyms

3GPP 3rd Generation Partnership Project.
BTS Base Transceiver Station.
CBR Case-Based Reasoning.
CCO Coverage and Capacity Optimisation.
CLL Conditional log-likelihood.
CM Configuration management.
CQI Channel Quality Indicator.
FOL First-Order Logic.
IoT Internet of Things.
KB Knowledge Base.
KPI Key Performance Indicator.
LTE Long-Term Evolution.
MAP Maximum \textit{a Posteriori}.
MCMC Markov Chain Monte Carlo.
MLN Markov Logic Network.
NE Network Element.
NGMN  Next Generation Mobile Networks.

NMS  Network Management System.

OAM  Operation, Administration and Maintenance.

RET  Remote Electrical Tilt.

RLF  Radio Link Failure.

SON  Self-Organising Networks.

TXP  Transmission Power.
Chapter 1

Introduction

The number of mobile subscriptions in the world has increased immensely last decade. At the same time, the amount of wireless traffic continues to grow at a continuously accelerating pace. According to a recent report by Ericsson [2], in 2014 there were 7.1 billion mobile subscriptions globally and by 2020 the number is expected to achieve 9.2 billion, of which mobile broadband subscriptions will account for more than 80 percent. Overall mobile data traffic is expected to increase tenfold from 2014 to 2020. The report notes that the exponential growth in mobile traffic is primarily caused by the increasing demand for social networking and video streaming services. Another contributor is the emerging market of Machine-to-Machine (M2M) communications and Internet of Things (IoT) solutions.

As discussed in the book by Hämäläinen et al. [14], to cope with such massive demand for data traffic, mobile network operators need to significantly upgrade their networks and to use the resources most efficiently. This means that the networks will be very complex and the individual Network Elements (NEs) must be dynamically configured during operation to adapt to changing traffic situations and possible malfunctions. Manual configuration of the NEs is expensive and vulnerable to mistakes. As the amount of traffic grows, the amount of money the network operators earn per transferred bit is shrinking and therefore they must try to keep their operating expenditure at the current level. This means that, in practice, configuring the individual NEs manually is impossible.

To address these challenges, the concept of Self-Organising Networks (SON) [14] has become frequently used. SON aims at increased automation of network operations in order to utilise deployed network resources in an optimised way. Currently the SON tasks are carried out by SON functions in a closed-loop manner with a predefined logic. The future goal is to introduce cognitive network management functions which can learn from
data and adapt. One approach to this is to define the SON functions as knowledge-based systems, which can be automatically updated.

Handling uncertainty and contradictory information is problematic in knowledge-based reasoning. In this thesis we introduce a probabilistic system for cognitive automated network management based on Markov Logic Networks (MLNs) [22], which combines First-Order Logic (FOL) and probabilistic graphical models. The MLN model allows combining hypothetical information with certain domain knowledge in a single, compact, presentation. MLN has been previously used in mobile network management in alarm diagnostics [6].

The structure of this thesis is as follows: in Chapter 2 we discuss mobile networks and SON in general. In Chapter 3 we go through the theory of MLN. In Chapter 4 we will introduce the MLN based network management model and in Chapter 5 we will test the model in practice. In Chapter 6 we will discuss the results and finally, in Chapter 7 we summarise and present conclusions.
Chapter 2

Mobile Networks and SON

In this chapter, we give an overview to mobile networks and their management. We also give an overview to the concept of SON which targets the automation of management of mobile networks. Finally, we discuss the requirements for cognitive network management in future mobile networks.

2.1 Mobile networks overview

This overview to mobile networks has been composed from the books by Cox [7] and Schwartz [24] and the thesis by Hätönen [15]:

A mobile network provides mobile telecommunications services, such as telephone and the Internet, for a large geographical area such as a state or a country. The current state-of-the-art mobile networks are fourth generation networks, called Long-Term Evolution (LTE) networks [7]. A mobile network is run by a network operator and the network is operated via an Operation, Administration and Maintenance (OAM) system. A mobile network user can connect his or her mobile (terminal), such as a mobile phone or a tablet, to the network and move within the service area without losing the connection to the network. The continuous coverage is provided by placing base stations at various locations throughout the geographical area of the network. Each base station has one or more sets of antennae which are oriented to cover a sector called a cell and they are used to communicate with the mobiles in the sectors. A typical base station uses three sets of antennae to control three sectors, each of which spans an arc of 120°. Figure 2.1 shows an example layout of base stations covering a part of a mobile network.

Remark. In the remaining of this thesis we will use the word cell to refer both to a sector and to the set of antennae and other equipment in a base station that provide the coverage of the sector.
The size of each cell is limited by the maximum range at which the receiver can hear the transmitter. Each cell also has a limited capacity, which is determined by the maximum combined data rate of all mobiles in the cell. The required size and capacity of a cell depend on its location. There exist several types of cells with different sizes and capacities, which co-operate in a multi-layered network topology. *Macrocells* provide wide-range coverage in rural areas or suburbs and have a size of a few kilometres. *Microcells* have a size of a few hundred metres and provide a greater collective capacity suitable for densely populated urban areas. *Picocells* are used in large indoor environments, such as shopping centres and offices, with a coverage area of a few tens of metres across. Finally, subscribers can buy and install *home base stations* in their own homes or small offices. They control *femtocells*, which cover an area of a few metres across.

### 2.2 Mobile network management

According to [11], a mobile network is managed by a Network Management System (NMS). The role of network management is to optimise the operational capabilities of the network, which includes keeping the network operating at peak performance. The tasks of an NMS can be divided into five categories, of which the first three are in the focus of this thesis:
• *Fault management* provides information on the status of the network. This includes information on failures in the network but also information on degraded performance. Fault management should be able to isolate the cause of the problem and to fix network performance.

• *Configuration management* is responsible for the resources to be managed, such as the timely deployment of network resources to satisfy expected service demand. It collects data from and delivers data for the network for the purpose of preparing for, initializing, starting, and providing for the operation and termination of services.

• *Performance management* is responsible for monitoring network performance to assure it is meeting specific goals defined by the operator. Such goals might include that sufficient capacity exists to support end-user communication requirements, but also that there is no unnecessary excess capacity and therefore no wasted resources.

• *Security management* controls access to the network and the NMS and protects them from abuse, unauthorised access and communication loss.

• *Accounting management* processes and records service and utilization information. It generates customer billing reports based on services rendered, identifies costs and establishes charges for the use of services and resources in the network.

### 2.3 Measurements

Network management is based on the data that the NEs, such as BTSs, generate. The NMS collects the data from the NEs for further analysis and use in different network management functions.

The data can be divided into three categories:

• *System configuration data* contain information about how the network has been constructed and organised. These data include, among others, the network topology. These types of data are highly persistent and are not usually modified after the deployment of the network.

• *System parameter data* define how different system elements are implemented and how they operate in the network. Examples of such data include antenna transmitter output power or the threshold for the maximum number of simultaneous connections of a BTS. These parameter data can be modified during network operation by the network operator.
• *Dynamic data* consist of measurement values and statistical time series, alarms and other events, call records, system logs, etcetera. These data describe the dynamic use of the network and operation of processes implementing network functions.

As discussed in [18], system parameter data is handled by Configuration management (CM). Parameter changes are defined in the NMS and then deployed to NEs. For performance management, the NEs count and record all elementary events occurring during the operation of the network, forming raw low level counter value data. The time frame used in the counting depends on the management purpose and can vary from minutes to days. These counter values are analysed in order to gain information about network performance. In a typical network there exist thousands of different counters for different events, and therefore they are usually aggregated to higher level variables called *Key Performance Indicators (KPIs)*. KPIs are formed to present understandable and easily interpretable functional factors and they are often given descriptive names.

**Example 2.3.1.** Here is an example of two commonly used KPIs:

1. A *Radio Link Failure (RLF)* [19] occurs when a user terminal incidentally loses the connection to the network. When the connection is re-established, the terminal can send an RLF report, indicating the last connected cell, to the network. The number of RLF events in a given time frame can then be counted for each cell.

2. An LTE terminal repeatedly sends a *Channel Quality Indicator (CQI)* report [7] to the BTS. The CQI report is an integer value between 0-15 and indicates maximum coding rate that the terminal can handle. The number of CQI reports in each class in a given time frame can be counted for each cell.

### 2.4 Self-Organising Networks (SON)

As discussed in the previous chapter, the number of adjustable settings and parameters in today’s mobile networks is too large to be addressed manually, due to the complex nature of the network, without increasing the operational cost remarkably. The SON concept [14] aims at increased automation of network operations in order to utilise deployed network resources in an optimised way. Several SON use cases have been defined by the Next Generation Mobile Networks (NGMN) and 3rd Generation Partnership Project.
The SON use cases can be categorised into three functional areas:

- **Self-configuration** consists of tasks related to bringing new NEs or NE parts to service with minimal human operator intervention. Such tasks include automatic connectivity setup between the NE and the OAM system and automatic configuration of BTS radio parameters.

- **Self-optimisation** consists of optimisation tasks during the operation of the network. Such tasks are needed, for example, due to changes in the network environment and are based on measurements from the network.

- **Self-healing** consists of actions that keep the network operational or prevent disruptive problems from arising. Such tasks include detecting faulty or degraded performance, identifying the cause for and automatic recovery from such situations.

The SON tasks are carried out by SON functions in a closed-loop manner without human intervention. Typically each SON function is designed to take care of one SON use case.

**Example 2.4.1** (Coverage and Capacity Optimisation, derived from [19]). The coverage and capacity of a cell may vary due to changes in the environment. The changes can be due to season, such as trees in full leaf during summer or heavy snowfall during winter, or man-made changes in the environment. The requirements in coverage and capacity may also change due to variations in traffic distribution, for example in city centre during busy hour, if compared with quiet hours in the night. Because manual optimisation of network coverage and capacity is expensive and time consuming, the Coverage and Capacity Optimisation (CCO) use case has been identified for automatic optimisation of network resources. One way to optimise cell capacity and coverage is to automatically adjust the Remote Electrical Tilt (RET) i.e. horizontal tilt, or the Transmission Power (TXP) of the cell antenna.

As described in [8], SON functions can operate on different levels of the network architecture. They can be located in the NMS where they operate to meet centralized policies, reconfiguring NE parameters based on network information gathered from the NEs. They can also be located in the NEs where they operate distributed, based on local policies received from network management.
In a self-organising system there are many SON functions which perform several tasks related to the configuration, optimisation and healing of the network. As noted by Räisänen and Tang [21], it is likely that some SON functions are applied to the same geographical area of the network at the same time. Therefore the operations of SON functions must be coordinated, so that several functions do not try to perform conflicting operations to the same area at the same time. The SON functions can be coordinated by a centralised SON coordinator, or the functions can coordinate by themselves in distributed manner [21].

Besides coordinating the operation of several SON functions, it would be beneficial that a SON system would be able to verify the activity of a SON function. After the execution of configuration change actions by a SON function, the system needs to monitor for changes in network performance. In case of undesired network behaviour caused by the actions, the system should be able to undo these configuration changes. SON verification is still under research, and not in active use [28, 29].

2.5 Cognitive network management

In a SON system the behaviour of SON functions is configured through high-level parameters and policies which are determined by a human operator according to the operational context, that is, operator goals, network environment and the technical properties of the network [12]. If the context changes frequently, adapting the SON configuration can be challenging for the human operator. Because of this, the automated network management mechanisms need to be able to adapt to changes in the operational context.

A report of the COMMUNE project [3] discusses the requirements and techniques for cognitive reasoning and learning in automated network management. One of the requirements discussed in the report is handling uncertainty, i.e., coping with partial or unreliable data in the network. Another requirement discussed is the co-operation of cognitive functions deployed in different parts of the network.

One of the techniques introduced in the report is Case-Based Reasoning (CBR) [1]. In CBR experience from previous problem situations is stored into a case base and a new problem is solved by finding a similar past case and reusing the solution to the problem. In mobile network management the use of CBR has been studied especially for automatic fault detection and diagnostics [27], where a problem is expressed as a set of KPI values and a solution is either a root cause for the problem or a corrective configuration action. If there is no matching case for a problem, then a solution is acquired.
from a human operator and the problem and the solution are included as a new case in the case base.
Chapter 3

Markov Logic Networks

Markov logic \[22\] is a language that combines first-order logic and Markov networks to define probability distributions in relational domains. In Markov logic, unlike in first-order logic, if a world violates a formula, it is not invalid, but less probable. An \textit{MLN} is a first-order knowledge base with a weight attached to each formula. Other things being equal, a world that satisfies a positive (negative) weighted formula is more (less) likely than a world that does not satisfy the formula. The magnitude of the weight determines the difference of the likelihoods. An \textit{MLN} can also contain hard constraints, i.e. formulas that must be satisfied for a world to be valid. Inference in Markov logic allows us to perform probabilistic reasoning over structured data.

3.1 Definition

\textbf{Definition 3.1.1.} A Markov logic network \(L\) is a set of pairs \((F_i, w_i)\), where \(F_i\) is a first-order formula and \(w_i\) is a real number, the weight of formula \(F_i\). Together with a finite set of constants \(C\) (the constant terms to which the predicates appearing in the formulae in \(L\) are applied to) it defines a ground Markov network \(M_{L,C}\) as follows:

1. \(M_{L,C}\) contains a binary node for each possible grounding of each predicate appearing in the formulae of \(L\).

2. \(M_{L,C}\) contains one feature for each possible grounding of each formula \(F_i\) in \(L\). The value of the feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the \(w_i\) associated with \(F_i\) in \(L\).

\textit{Remark.} All features corresponding to the groundings of the same formula will have the same weight.
Remark. Hard constraints in an MLN model can be understood as formulas with infinite weights.

The set of variables $X = \{X_1, \ldots, X_n\}$ of the ground Markov network $M_{L,C}$ is thus the set of ground atoms implicitly defined by the predicates in $L$ and the set of constants $C$. The ground Markov network $M_{L,C}$ defines a probability distribution over possible worlds $x \in \mathcal{X}$, i.e. the set of possible assignments of truth values to each of the ground atoms in $L$, as follows,

$$P_{M_{L,C}}(X = x) = \frac{1}{Z} \exp \left( \sum_i w_i n_i(x) \right)$$

(3.1)

where $Z := \sum_{x' \in \mathcal{X}} \exp(\sum_i w_i n_i(x'))$ is the partition function and $n_i(x)$ is the number of the true groundings of the $i$th formula in possible world $x \in \mathcal{X}$. This definition follows the log-linear representation of a Markov network.

If two possible worlds differ only on a single formula $F$, then the weight of the formula is the log-odds between the two worlds. For example, if $x, x' \in \mathcal{X}$, $n_i(x) - n_i(x') = 1$ for formula $F_i$ and $n_j(x) = n_j(x')$ for all other formulae $F_j$, then

$$\frac{P_{M_{L,C}}(X = x)}{P_{M_{L,C}}(X = x')} = \frac{\frac{1}{Z} \exp(\sum_k w_k n_k(x))}{\frac{1}{Z} \exp(\sum_k w_k n_k(x'))}$$

$$= \exp(w_i(n_i(x) - n_i(x')))$$

$$= \exp(w_i)$$

However, usually a formula $F$ has common variables with other formulae, and it might not be possible to reverse the truth assignment of formula $F$ and at the same time keep the truth assignment of those other formulae unchanged. In this situation, there is no one-to-one relation between the weights and the probabilities of the formulae.

### 3.2 Inference

#### 3.2.1 MAP Inference

A basic inference task is to find the most probable state of the world given some evidence. This is known as Maximum a Posteriori (MAP) inference. From eq. (3.1) we can see that in Markov logic this reduces to finding a truth assignment that maximises the sum of the weights of the satisfied clauses. This problem can be solved using any weighted satisfiability solver. [9] efficiently use MaxWalkSAT [16], a weighted variant of the WalkSAT stochastic search satisfiability solver [25].
Algorithm 1 Network construction for inference in MLNs adapted from [22]

**Input:** $F_1$, a set of ground atoms with unknown truth value
$F_2$, a set of ground atoms with known truth values
$L$, a Markov logic network
$C$, a set of constants

**Output:** $M$, a ground Markov network

**function** `ConstructNetwork($F_1, F_2, L, C$)

```plaintext
G ← $F_1$
while $F_1 \neq \emptyset$
  do
    for all $q \in F_1$
      do
        if $q \notin F_2$
          then
            // MB($q$) is the Markov blanket of $q$ in $M_{L,C}$
            $F_1 \leftarrow F_1 \cup (\text{MB}(q) \setminus G)$
            $G \leftarrow G \cup \text{MB}(q)$
            $F_1 \leftarrow F_1 \setminus \{q\}$

  return $M$, the ground Markov network composed of the nodes in $G$, all arcs between them in $M_{L,C}$ and the weights and features on the corresponding cliques
```

3.2.2 Marginal and Conditional Probabilities

Another important inference task is computing the probability that a formula holds given an MLN and a set of constants, and possibly other formulae as evidence. If $F_1$ and $F_2$ are two formulae in FOL, $C$ is a finite set of constants that includes any constants that appear in $F_1$ or $F_2$, and $L$ is an MLN, then

$$P(F_1|F_2, L, C) = P(F_1|F_2, M_{L,C})$$

$$= \frac{P(F_1, F_2|M_{L,C})}{P(F_2|M_{L,C})}$$

$$= \frac{\sum_{x \in X_{F_1} \cap X_{F_2}} P_{M_{L,C}}(X = x)}{\sum_{x \in X_{F_2}} P_{M_{L,C}}(X = x)}, \quad (3.2)$$

where $X_{F_i}$ is the set of the worlds where $F_i$ holds.

As summing over the possible worlds requires time exponential in the number of possible ground literals, computing eq. (3.2) directly is feasible only in the smallest domains. [22] propose an efficient inference algorithm for the case where $F_1$ and $F_2$ are conjunctions of ground literals, which, in practice, is the most common type of query. The proposed algorithm proceeds in two phases. The first phase computes the minimal subset $M$ of the ground Markov network required to compute $P(F_1|F_2, L, C)$. Pseudocode
for the first phase is shown in algorithm 1. The network is constructed as follows: First, the atoms in the query formula are added. Then, repeatedly, those atoms, that the atoms in the network depend on, and that are not in the evidence, are added, until all relevant atoms have been retrieved.

The second phase uses Gibbs sampling \([5]\), a widespread Markov Chain Monte Carlo (MCMC) sampling technique \([13]\), to perform inference on network \(M\). The basic step consists of sampling one ground atom given its Markov blanket. The probability of a ground atom \(X_l\) when its Markov blanket \(B_l\) is in state \(b_l\) is

\[
P(X_l = x_l | B_l = b_l) = \frac{\exp \left( \sum_{f_i \in F_l} w_i f_i(X_l = x_l, B_l = b_l) \right)}{\sum_{j \in \{0, 1\}} \exp \left( \sum_{f_i \in F_l} w_i f_i(X_l = j, B_l = b_l) \right)}
\]

where \(F_l\) is the set of the formulae that \(X_l\) appears in, and \(f_i(X_l = x_l, B_l = b_l)\) is the value of the feature corresponding to the \(i\)th ground formula when \(X_l = x_l\) and \(B_l = b_l\). The estimated probability of a conjunction of ground literals is the fraction of samples where the ground literals are true, after the Markov chain has converged. Because the distribution is likely to have many modes, the Markov chain needs to be run many times. When the \(\text{MLN}\) is in clausal form, we can minimize the burn-in period by starting each round from a mode found by MaxWalkSAT.

As \([10]\) point out, the MCMC method will not work well if the \(\text{MLN}\) contains deterministic or near-deterministic dependencies. Deterministic dependencies break up the space of possible worlds into regions that are not reachable from each other, which violates a basic requirement of MCMC. Near-deterministic dependencies create regions of low probabilities that are very difficult to traverse and greatly slow down inference. To solve this problem, the authors in \([20]\) introduce MC-SAT, a slice sampling algorithm, which greatly outperforms standard MCMC methods in domains with deterministic or near-deterministic dependencies. The algorithm uses SampleSAT \([33]\) to uniformly sample a new state.

3.3 Weight learning

The weights of the formulae in an \(\text{MLN}\) can be learned from data. Given a set of formulae and a training database of atoms, we wish to find the MAP weights of the formulae, i.e., the mode of the posterior density of the weights:

\[
f(w | X = x) = \frac{P_w(X = x) f(w)}{\int P_w(X = x) f(w') d w'},
\]
where $P_w(X = x)$ is the likelihood of database $x$ given the weights $w$ as defined in eq. (3.1) and $f(w)$ is the prior density of the weights. Since optimization is usually formulated as function minimization, we will equivalently minimize the negative posterior. The denominator of the posterior does not depend on $w$ and can be ignored in the optimization. Thus the task of maximizing the posterior is equivalent to minimizing the sum of the negative log-likelihood and the negative log-prior:

$$Q(w) = -\log P_w(X = x) - \log f(w).$$

A well-known method for function minimization is the gradient descent method, where we update the weight vector by taking steps proportional to the negative gradient of the function at the current point,

$$w^{(t+1)} = w^{(t)} - \eta g,$$

where $w^{(t-1)}$ is the weight vector of the current model, $g$ is the gradient of function $Q$ with respect to the current weights and $\eta$ is a learning rate parameter.

The partial derivative of the log-likelihood with respect to weight $w_i$ is

$$\frac{\partial}{\partial w_i} \log P_w(X = x) = \frac{\partial}{\partial w_i} \sum_j w_j n_j(x) - \frac{\partial}{\partial w_i} \log Z$$

$$= n_i(x) - \frac{1}{Z} \frac{\partial}{\partial w_i} Z$$

$$= n_i(x) - \frac{1}{Z} \sum_{x' \in \mathcal{X}} \frac{\partial}{\partial w_i} \exp \left( \sum_j w_j n_j(x') \right)$$

$$= n_i(x) - \frac{1}{Z} \sum_{x' \in \mathcal{X}} \exp \left( \sum_j w_j n_j(x') \right) n_i(x')$$

$$= n_i(x) - \sum_{x' \in \mathcal{X}} P_w(X = x') n_i(x'),$$

which is the difference between the observed number of the true groundings of formula $i$ in the data and the expected number according to the current model. As [22] point out, calculating the expected number of true groundings is intractable. Instead they propose optimizing the pseudo-likelihood of the data according to the model [4]:

$$P_w^*(X = x) = \prod_{l=1}^n P_w(X_l = x_l|MB_{x}(X_l)),$$

where $MB_{x}(X_l)$ is the Markov blanket of variable $X_l$ in the data.
In many applications, the goal is to correctly predict the value of some query predicates given the value of a set of evidence predicates, and we know a priori which predicates will be queried and which ones will be evidence. If we partition the ground atoms in the domain into a set of evidence atoms $X$ and a set of query atoms $Y$, the Conditional log-likelihood (CLL) of $Y$ given $X$ is

$$P_w(Y = y|X = x) = \frac{1}{Z_x} \exp \left( \sum_{i \in F_Y} w_i n_i(x, y) \right),$$

where $F_Y$ is the set of all MLN formulae with at least one grounding involving a query atom, $n_i(x, y)$ is the number of true groundings of the $i$th formula involving query atoms and $Z_x$ is the partition function. For simplicity, all non-evidence predicates are treated as query predicates. \cite{26} show that the pseudo-likelihood method is consistently outperformed by discriminative training, which minimizes the CLL of the query predicates given the evidence ones. In this thesis, we will focus on discriminative training.

If we assume that the weights are independent and use a Gaussian prior for each weight $w_i$ with mean $\mu_i$ and variance $\sigma^2$, the partial derivative of the log-prior with respect to $w_i$ is

$$\frac{\partial}{\partial w_i} \log f(w) = \frac{\partial}{\partial w_i} \log \prod_j \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{1}{2} \frac{(w_j - \mu_j)^2}{\sigma^2} \right)$$

$$= \sum_j \frac{\partial}{\partial w_i} \left( -\frac{1}{2} \frac{(w_j - \mu_j)^2}{\sigma^2} \right)$$

$$= -\frac{w_i - \mu_i}{\sigma^2}.$$

A natural source for the prior is to use the initial formula weights as the prior means. The prior variance can then be adjusted to control the influence of the prior to the posterior distribution.
Chapter 4

MLN model for network management

In Section 2.5 we discussed the requirements and goals for network management in future mobile networks. The main goal discussed is the ability for the network management functions to adapt to changes in the operational context. This means that, instead of operating on a static set of rules, the functions could be able to build up knowledge of the effects of the configuration actions. One such approach mentioned was the use of CBR in fault detection and diagnosis studied in [27]. One assumption in the article was that to each problem there is only one correct solution. In general this assumption is not true. As discussed in Section 2.5 the cognitive management functions need to cope with uncertainty. One source of uncertainty could be that, because it is impossible to model the network behaviour in every detail, a solution to an observable problem might work in some occurrences of the problem, but not in the others. There could also be several potential solutions to the problem.

Another requirement discussed in Section 2.5 was the co-operation of distributed cognitive functions. We would like to be able to consolidate the knowledge built up independently in the functions. If one function, for example, discovers an action that is effective in a very specific situation, we would like to make this knowledge available for other functions, perhaps operating in other parts of the network so that they immediately know how to act when facing this situation for the first time. Therefore we need a method of combining the Knowledge Bases (KBs) of several functions into a single model. A further motive for creating a combined model is to provide a perceivable interface for the human operator to explore and to verify the discovered knowledge and possibly input and test their own hypotheses. When combining KBs of several functions (and possibly a human operator),
the resulting model may contain contradictory information.

In this chapter, we will introduce a method for cognitive network management based on \( \text{MLN} \). Using an \( \text{MLN} \) model we are able to handle uncertain and contradictory knowledge. The model allows probabilistic inference of network configuration actions based on measured network data. We will focus on inference and learning the model parameters and assume that the (hypothetical) knowledge is acquired using external mechanisms, for example a CBR approach or from a human operator.

### 4.1 Model

A network management function makes changes to \( \text{CM} \) parameters based on the current status of the network, i.e. \( \text{KPI} \) values, network topology, etc. The changes are made to achieve some performance objectives, i.e. target \( \text{KPI} \) values. We will define an \( \text{MLN} \) model using these same kinds of information elements and connections between them.

#### 4.1.1 Predicates

Our model contains three kinds of predicates:

- **Context predicates** indicate the current status of the network and the environment. A context predicate can indicate, for example, that some \( \text{KPI} \) value for a cell is currently below the acceptable level, that two cells are neighbours in the network topology, or that there are a lot of terminals connected to a cell. A context predicate can also indicate some properties of the current \( \text{CM} \) parameter values, for example, that the value of a parameter is at the minimum or maximum level.

- **Objective predicates** indicate required changes to \( \text{KPI} \) values to achieve target values defined by the operator, e.g., a particular \( \text{KPI} \) value for some cell is too low and needs to be increased.

- **Action predicates** indicate changes, increases or decreases, to \( \text{CM} \) parameter values.

Each predicate represents an attribute of a cell in the network or a relation among the cells. The domain of a predicate can be either the set of cells \( X \) or an \( n \)-ary Cartesian product of \( X \). For example, if predicate \( P \) represents an increase in a certain configuration parameter of a cell and predicate \( Q \) represents a neighbourhood relation among a pair of cells, then the domain of \( P \) is \( X \) and the domain of \( Q \) is \( X \times X \).
To determine the values of the context predicates representing the current performance of the network, the numeric KPI values need to be categorised into ordinal classes. Defining the classification intervals of KPIs is not a straightforward task. To say that the value of a KPI is above or below normal level, one must first determine the normal level for that KPI. This is closely related to anomaly detection in mobile networks [18] and is beyond the scope of this thesis.

We assume that the acceptable region for each KPI is connected, i.e. an interval. Then we define the value of an objective predicate representing a required change to the value of a KPI as an indicator of a step towards the acceptable region of that KPI. We do not require that the acceptable region is achieved due to the step, but that the change is large enough step, i.e. larger than some predefined threshold value.

The value of an action predicate representing a change to the value of a CM parameter is defined as an indicator of a (predefined) fixed-size increase or decrease of the parameter value.

4.1.2 Formulae

We consider the knowledge in cognitive functions as a set of rules which can be expressed as a KB in FOL where each rule is a formula of the form

\[ C_1(x) \land C_2(x) \land \cdots \land C_k(x) \Rightarrow A_1(x) \land A_2(x) \land \cdots \land A_l(x) \]

which means that when the conditions indicated by the context predicates \( C_1, C_2 \ldots C_k \) hold, then CM parameter changes indicated by action predicates \( A_1, A_2, \ldots, A_l \) are executed. This formulation is analogous to the formulation of cases in the CBR approach discussed in Section 2.5. We assume that there exists, for each rule, a set of objective predicates \( O_1, O_2, \ldots, O_m \), determined by the current performance of the network (which is indicated by the context predicates) and the KPI target values of the function. Therefore, the function rule, in effect, expresses a formula

\[ C(x) \Rightarrow (A(x) \Rightarrow O(x)) \],

which is intuitively correct, it is not useful for our purposes: We want to use the MLN model to query for CM actions, i.e. values for the grounded action predicates. The values of the context and objective predicates are determined by the current network status and are given as evidence for the query. If the conjunction of the objective predicates is satisfied
by the evidence, then the formula is satisfied regardless of the values of the action predicates, and therefore the formula has no effect on the result of the query. A better alternative would be to change the direction of the right side implication of formula 4.1:

\[ C(x) \Rightarrow (O(x) \Rightarrow A(x)). \] (4.2)

Now, given that \( C(x) \) and \( O(x) \) are satisfied, formula 4.2 is satisfied only if also \( A(x) \) is satisfied, and the formula does affect the result of the query.

While formula 4.2 can be used for querying, it now poses another problem: If we want to learn the MLN rule weights using previous network status data, CM parameter changes and realized KPI value changes as training data, formula 4.2 is always satisfied if \( A(x) \) is satisfied, and we lose the information, whether the executed actions resulted in the wanted objectives or not. The final alternative is to use an equivalence between the object predicates and the action predicates:

\[ C(x) \Rightarrow (O(x) \leftrightarrow A(x)). \] (4.3)

Now formula 4.3 is useful for both querying and for weight learning. For each function rule we include one such formula in the MLN model.

In addition to the formulae corresponding to the function rules, we can also include certain relational knowledge in the MLN model as hard constraints. Such knowledge can include ontological knowledge in the telecommunications domain or consistency constraints, such as, that the cell neighbourhood relation is irreflexive and symmetric, that a KPI value of a cell can not both at low level and at normal level, or that a CM parameter can not be increased, if it is already at the maximum level.

**Example 4.1.1.** Let us look at a simple example MLN model. Let us consider a small network consisting of two cells: \( c = C_1, C_2 \), one KPI \( K \), and one CM parameter: \( P \). Let us assume we know how to categorize the values of \( K \) to low, moderate or high and let us define the following context predicates: \( IsLow(cell, K) \), \( IsModerate(cell, K) \) and \( IsHigh(cell, K) \). Let us also define the following objective and action predicates: \( ObjIncrease(cell, K) \), \( ObjDecrease(cell, K) \), \( ActIncrease(cell, P) \) and \( ActDecrease(cell, P) \). If our goal now is to keep the value of KPI \( K \) at a moderate level, we can
define an $\text{MLN}$ consisting of the following formulas:

\begin{align*}
\infty & : \text{IsLow}(c, K) \lor \text{IsModerate}(c, K) \lor \text{IsHigh}(c, K) \\
\infty & : (\neg \text{IsLow}(c, K) \lor \neg \text{IsModerate}(c, K)) \land \\
& \quad (\neg \text{IsModerate}(c, K) \lor \neg \text{IsHigh}(c, K)) \land \\
& \quad (\neg \text{IsLow}(c, K) \lor \neg \text{IsHigh}(c, K)) \\
\infty & : \neg \text{ObjDecrease}(c, K) \lor \neg \text{ObjIncrease}(c, K) \\
\infty & : \neg \text{ActDecrease}(c, P) \lor \neg \text{ActIncrease}(c, P) \\
w_1 & : \text{IsLow}(c, K) \Rightarrow (\text{ObjIncrease}(c, K) \Leftrightarrow \text{ActIncrease}(c, P)) \\
w_2 & : \text{IsLow}(c, K) \Rightarrow (\text{ObjIncrease}(c, K) \Leftrightarrow \text{ActDecrease}(c, P)) \\
w_3 & : \text{IsHigh}(c, K) \Rightarrow (\text{ObjIncrease}(c, K) \Leftrightarrow \text{ActIncrease}(c, P)) \\
w_4 & : \text{IsHigh}(c, K) \Rightarrow (\text{ObjIncrease}(c, K) \Leftrightarrow \text{ActDecrease}(c, P))
\end{align*}

First we have four hard constraints that guarantee that the value of $K$ is in exactly one category and that neither the Objective nor Action can not be both decrease and increase. Then we have four formulas, each of which has a real weight parameter. Now we can see that the formulas are contradictory, and the formula weights describe our belief in each formula.

### 4.1.3 Model composition

A function knowledge base, defined as a set of rules, can be transformed into an $\text{MLN}$ following the procedure described in Section 4.1.2. Similarly, several functions can be transformed into a single $\text{MLN}$ model if they share the same set of predicates. If the functions use different predicates for describing the same facts, then we need to include a mapping of the predicates. It is not reasonable to combine the knowledge bases of every function into a single model, but only of such functions that carry out related use cases.

### 4.2 Inference and Learning

#### 4.2.1 Weights

The weights of the formulae in the $\text{MLN}$ model can be learned using the method described in Section 3.3. The training data consist of observations of previous context predicate values, executed CM actions and realized objectives. The initial weights given to the formulas can be used to describe our prior preferences of the rules. If we do not have any such preferences, we can initialize each weight to zero.
4.2.2 Inference

The MLN model can be used to infer CM configuration actions based on evidence. The evidence consists of context and objective predicate values which are determined by network measurements and operator targets, as discussed in Section 4.1.1. As discussed in Section 3.2, the result of the inference can be either an absolute truth value for each action (MAP state of the world) or a conditional marginal probability distribution for each action. In the latter case, the actual configuration actions can be drawn from the marginal distributions. This way we get more variance to historical data which is used as training data in the weight learning process.
Chapter 5

Experimental results

In this chapter, we study the applicability of the method introduced in the previous chapter. We will apply the MLN method to the CCO use case introduced in Example 2.4.1 and examine the results in terms of improvements in network performance and computational cost. We will analyse the qualitative change in network performance as our MLN model is used to infer CM parameter changes and the model is iteratively updated based on historical data.

5.1 Setting

We will carry out the experiments in a simulated LTE network environment. The simulated network consists of 12 LTE macro cell base stations and 32 cells in total, covering an urban area with a size of a few kilometres across. 1000 mobile terminals move individually at random, at a walking speed, around the simulation area, so that the density of terminals is approximately the same across the whole network during the simulation. Figure 5.1 contains a screen capture of the simulation software, showing the network layout. The implementation of the simulator is based on [30].

We will try to optimise the performance of the network by adjusting the parameters of each cell simultaneously based on a single MLN model and performance data measured from the simulated network. The data is measured every 15 minutes and the parameter changes are made after each measurement round. Besides performance data, we will also use cell neighbourship information as input for the MLN model.
5.2 Measured quantities

We will measure the following data from the network: for cell $c = 1, \ldots, 32$, terminal $u = 1, \ldots, 1000$ and time $t > 0$ in minutes, let

- $a_{u,t} \in \{0, 1, \ldots, 32\}$ be the cell that terminal $u$ is connected to. If a terminal is not connected to any cell ($a_{u,t} = 0$), it is outside of the network coverage area.

- $r_{c,t} \geq 0$ be the number of occurred RLFs (see Example 2.3.1 for details) in cell $c$ during the measurement interval $[t - 15, t]$.

- $d_{c,t} = (d_{c,t}^{(0)}, d_{c,t}^{(1)}, \ldots, d_{c,t}^{(15)}), d_{c,t}^{(j)} \geq 0, j = 0, \ldots, 15$, be the histogram of CQI reports (see Example 2.3.1 for details) in each CQI class for cell $c$ during the measurement interval $[t - 15, t]$.

We will then transform these data to the following three cell-specific scalar KPIs that are used to compute the evidence for MLN inference:

- $A_{c,t} = \sum_{u=1}^{1000} 1_c(a_{u,t})$ – Number of terminals connected to a cell. We will also compute the number of terminals that are not connected to any cell, $A_{0,t}$. 

• $R_{c,t} = \begin{cases} \frac{r_{c,t}}{A_{c,t}} & \text{if } A_{c,t} = 0 \\ 0 & \text{otherwise} \end{cases}$ – Ratio of occurred RLFs to the number of terminals connected to a cell.

• $Q_{c,t} = \frac{\sum_{j=0}^{15} b_j d^{(j)}_c}{\sum_{j=0}^{15} d^{(j)}_c}$ – The average channel efficiency. Here $b_j$ is the number of information bits per transferred symbol when using the coding rate indicated by CQI class $j$, shown in Table 5.1.

<table>
<thead>
<tr>
<th>CQI</th>
<th>Information bits per symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>1</td>
<td>0.15</td>
</tr>
<tr>
<td>2</td>
<td>0.23</td>
</tr>
<tr>
<td>3</td>
<td>0.38</td>
</tr>
<tr>
<td>4</td>
<td>0.60</td>
</tr>
<tr>
<td>5</td>
<td>0.88</td>
</tr>
<tr>
<td>6</td>
<td>1.18</td>
</tr>
<tr>
<td>7</td>
<td>1.48</td>
</tr>
<tr>
<td>8</td>
<td>1.91</td>
</tr>
<tr>
<td>9</td>
<td>2.41</td>
</tr>
<tr>
<td>10</td>
<td>2.73</td>
</tr>
<tr>
<td>11</td>
<td>3.32</td>
</tr>
<tr>
<td>12</td>
<td>3.90</td>
</tr>
<tr>
<td>13</td>
<td>4.52</td>
</tr>
<tr>
<td>14</td>
<td>5.12</td>
</tr>
<tr>
<td>15</td>
<td>5.55</td>
</tr>
</tbody>
</table>

Table 5.1: Information bits per transferred symbol for each CQI class. Adapted from [7].

Finally, the KPI values are categorised to low, moderate or high category. As discussed in the previous chapter, deciding the category limits is not a straightforward task, and the required sophisticated analysis is beyond the scope of this thesis. Instead, we will define the limits by hand. Table 5.2 shows the defined category limits. We will define our operator goals so that we wish to keep the RLF ratio $R_{c,t}$ at a low level and the average channel efficiency $Q_{c,t}$ at a moderate level for each cell. We will not set a target value for the number of terminals in a cell, $A_{c,t}$, but will only use it to describe current network status.
5.3 Network parameters

We will adjust the antenna TXP and RET of the network cells. The parameters have an initial value, a minimum and maximum allowed value, and after each measurement round they are either increased or decreased by a fixed step size, or stay unchanged. These values are shown in Table 5.3.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>init</th>
<th>min</th>
<th>max</th>
<th>step size</th>
</tr>
</thead>
<tbody>
<tr>
<td>TXP</td>
<td>20.0</td>
<td>9.0</td>
<td>40.0</td>
<td>1.0</td>
</tr>
<tr>
<td>RET</td>
<td>4.0</td>
<td>0.0</td>
<td>11.0</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 5.3: Minimum and maximum values and step size of parameters for each cell.

5.4 Model

We will define an MLN model following the "Context-Objective-Action" format defined in the previous chapter.

5.4.1 Variables and predicates

We will define context predicates indicating the category of each KPI value of a cell. We will also define a context predicate indicating a neighbourship relation of a pair of cells. Cells in the same BTS are always neighbours (Intra-BTS), but cells in different BTSs can also be neighbours if the cells are adjacent in the network (Inter-BTS). Finally, we will define context predicates indicating that a cell parameter is currently at minimum or maximum value.

We will define objective predicates indicating a desired increase or decrease in cell KPI value for the RLF ratio $R_{c,t}$ and for the average channel efficiency $Q_{c,t}$ and action predicates indicating required increase or decrease in cell parameters for TXP and RET.

The variables in the predicates are defined in Table 5.4 and the predicates are defined in Table 5.5.
### Table 5.4: Variables used in the model predicates and their possible values

<table>
<thead>
<tr>
<th>Name</th>
<th>Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell</td>
<td>1,...,32</td>
<td>Cell id</td>
</tr>
<tr>
<td>kpi</td>
<td>A, R, Q</td>
<td>Name of the cell KPI</td>
</tr>
<tr>
<td>param</td>
<td>RET, TXP</td>
<td>Name of the cell parameter</td>
</tr>
<tr>
<td>nship</td>
<td>INTER, INTRA</td>
<td>Type of neighbourship relation</td>
</tr>
</tbody>
</table>

### Table 5.5: Definition of the context, objective and action predicates used in the model.

<table>
<thead>
<tr>
<th>Name</th>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IsLow</td>
<td>cell, kpi</td>
<td>value of kpi of cell is at a low level</td>
</tr>
<tr>
<td>IsMod</td>
<td>cell, kpi</td>
<td>value of kpi of cell is at a moderate level</td>
</tr>
<tr>
<td>IsHigh</td>
<td>cell, kpi</td>
<td>value of kpi of cell is at a high level</td>
</tr>
<tr>
<td>IsAtMax</td>
<td>cell, param</td>
<td>value of param of cell is at the maximum level</td>
</tr>
<tr>
<td>IsAtMin</td>
<td>cell, param</td>
<td>value of param of cell is at the minimum level</td>
</tr>
<tr>
<td>IsNeighbour</td>
<td>cell, cell, nship</td>
<td>cell is a type nship neighbour of cell</td>
</tr>
<tr>
<td>ObjInc</td>
<td>cell, kpi</td>
<td>increase value of kpi of cell</td>
</tr>
<tr>
<td>ObjDec</td>
<td>cell, kpi</td>
<td>decrease value of kpi of cell</td>
</tr>
<tr>
<td>ActInc</td>
<td>cell, param</td>
<td>increase value of param of cell</td>
</tr>
<tr>
<td>ActDec</td>
<td>cell, param</td>
<td>decrease value of param of cell</td>
</tr>
</tbody>
</table>
5.4.2 Formulas and weights

As discussed in the previous chapter, we want to be able to combine knowledge from different sources, for example correlations discovered by cognitive network management functions and hypotheses made by a human operator. The resulting knowledge base could be large and some of the rules erroneous. Therefore we will not try to define a set of effective rules by hand. Instead, we will generate the set of formulas combinatorically from the defined context, objective and action predicates. This way we get a varied set of formulas, of which some are poor and some are good. We will generate two types of formulas as follows:

- All possible combinations of cell-specific formulas for cell $c$ with context predicates of one, two or three KPIs, objective predicates of one or two KPIs and action predicates of one or two parameters.

- All possible combinations of cell-pair-specific formulas for cell pair $(c_1, c_2)$ with a neighbourship predicate, context predicates of one, two or three KPIs for $c_1$, objective predicates of one or two KPIs for $c_1$ and action predicates of one or two parameters for $c_2$.

For both formula types we filter out those outcomes in which the objective predicates do not match the context predicates, i.e., where

- there is no context predicate for some of the KPIs of the objective predicates, or

- a context predicate indicates that an operator goal is not currently achieved, but there is no related objective predicate.

In total 1920 formulas resulted from the application of the above criteria.

To make our model sound, we need to define hard constraints that guarantee the following properties:

- A KPI value must belong to exactly one category:

  \[
  (\text{IsLow}(c,k) \lor \text{IsMod}(c,k) \lor \text{IsHigh}(c,k)) \land \\
  (\neg \text{IsLow}(c,k) \lor \neg \text{IsMod}(c,k)) \land \\
  (\neg \text{IsMod}(c,k) \lor \neg \text{IsHigh}(c,k)) \land \\
  (\neg \text{IsLow}(c,k) \lor \neg \text{IsHigh}(c,k))
  \]

- A KPI or a parameter value can not be both increased and decreased at the same time:

  \[
  (\neg \text{ObjInc}(c,k) \lor \neg \text{ObjDec}(c,k)) \land \\
  (\neg \text{ActInc}(c,p) \lor \neg \text{ActDec}(c,p))
  \]
• If a parameter is currently at the maximum/minimum possible value, it can not be increased/decreased any further:

\((\neg \text{IsAtMax}(c,p) \lor \neg \text{ActInc}(c,p)) \land (\neg \text{IsAtMin}(c,p) \lor \neg \text{ActDec}(c,p))\)

• The neighbourship relation is irreflexive and symmetric and two cells can not be both Inter-BTS and Intra-BTS neighbours:

\((\text{IsNeighbour}(c_1,c_2,n) \lor \neg \text{IsNeighbour}(c_2,c_1,n)) \land \\
(\neg \text{IsNeighbour}(c,c,n)) \land \\
(\neg \text{IsNeighbour}(c_1,c_2,\text{INTRA}) \lor \text{IsNeighbour}(c_1,c_2,\text{INTER}))\)

The initial weights of the uncertain formulas are set to zero. The weights are updated after every 48 measurement rounds, i.e. every 12 hours, using the measurements from previous \(h\) rounds as training data.

5.5 Process

We use the Alchemy 2.0 software package \cite{Alchemy2} for MLN inference and weight learning. Alchemy 2.0 implements the marginal inference and discriminative weight learning algorithms discussed in Chapter \ref{chapter:algorithms}.

For inference, first the performance data is read from the simulator. Then the data is pre-processed together with cell neighbourship information to get truth values for all ground context and objective predicates. These values are written to a database file, which is input as evidence to infer marginal distributions for the action predicates based on the MLN model. Finally, the parameter changes are drawn from the marginal distributions and the changes are written to the simulator. This cycle is repeated every 15 minutes. Figure \ref{fig:flowchart} shows a diagram of the inference work flow.

Remark. In the pre-processing step the objective predicate truth values are derived from the context predicate truth values based on the defined operator goals. For example, if the RLF ratio for cell 1 is high, i.e. \text{IsHigh}(1,R) is true, then also \text{ObjDec}(1,R) is true.

Remark. We draw the parameter change actions from the marginal distributions, so that we get variance to the data that is later used to learn the model weights, even though this could lead to suboptimal network performance. Later, when we have acquired enough data and the weights have converged, we could use the most likely action predicate values to determine corresponding network parameter changes.
Figure 5.2: Workflow of the parameter change inference process. A: Data is read from the simulator for pre-processing. B: The pre-processed data is input to Alchemy for inference. C: Marginal distributions for action predicates, i.e. the output of the inference process, are used to draw parameter changes. D: The new parameter values are written to the simulator.

On each round the numerical KPI values and cell parameter values are also logged. After every 48 rounds, the model weights are updated using these data as training data. The historical sample data from the previous \( h \) rounds are pre-processed to get the realised objectives and actions. These data are then used to get a set of \( h \) database files, each of which contain the truth values of ground context, objective and action predicates for one round. These database files are input to the weight learning algorithm to get the MAP estimate weights using the current weights as prior means. Finally, the model weights are replaced by the learned MAP estimates and the new model is used for inference in the future.

5.6 Results

We will examine the changes in network performance with different history sizes \( h \). We will try using historical data from the last 24, 48 and 96 rounds. With each history size, we run the trial for 72 hours (288 rounds), during which the model weights are updated 6 times. For \( h = 96 \), in the first weight learning iteration we will use the data from the first 48 rounds, because there is no more data available yet. With each history size, the trial is run 3 times
and the results are averaged.

Because there are so many formulas in the model, the weight learning process would take very long to converge. Therefore we set the maximum number of gradient descent steps to 100. The algorithm always failed to converge during the maximum number of steps. Table 5.6 shows the average running times of the weight learning process. We can see that the running time is directly proportional to the history size.

<table>
<thead>
<tr>
<th>$h$</th>
<th>Running time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>68</td>
</tr>
<tr>
<td>48</td>
<td>134</td>
</tr>
<tr>
<td>96</td>
<td>286</td>
</tr>
</tbody>
</table>

Table 5.6: Average running time of the weight learning process with different history size $h$

Figure 5.3 shows how the total number of RLFs in the network changes as the model weights are updated with different history sizes. We can see that with all history sizes the number of RLFs begins to drop after the first update. With history size $h = 24$ the number starts to fluctuate after a couple of iterations, suggesting that 24 rounds of historical data are not sufficient for robust performance of the method. With $h = 48$ and $h = 96$ the number continues to drop after the first couple of iterations and stays stable during the rest of the iterations. The number is also substantially lower than in the beginning, where the cell parameters were set to their initial values. With $h = 48$ the number is most of the time slightly lower than with $h = 96$, so it seems that 48 rounds of historical data are enough, especially as the running time of the weight learning process doubles as the number of rounds is doubled.

Because terminals which are not connected to any cell do not generate any RLFs we need to also examine the number of terminals which are outside the network coverage area. Figure 5.4 shows how this number changes as the model weights are updated with different history sizes. We can see that with $h = 24$ the situation is again unstable, but with $h = 48$ and $h = 96$ the number drops rapidly to near zero and stays low.

---

1Alchemy 2.0 was run via Cygwin on a 64-bit Windows 7 computer with a 2.70 GHz CPU and 32 GB memory
Figure 5.3: Total number of radio link failures in the network with different history levels

Figure 5.4: Total number of terminals not connected to a cell in the network with different history levels
Chapter 6
Discussion

In Chapter 4 we defined a method for consolidating knowledge from cognitive network management functions using an MLN model. In Chapter 5 we used a single MLN model for optimising an entire simulated mobile network. In a real mobile network, this would not be possible if the network management was performed in a distributed way in the NEs. However, after the MLN model has been composed and the model weights have been fitted, the model could be disseminated back to the concerning distributed management functions. This approach would require that the internal knowledge presentation of the functions is implemented as an MLN.

The results in the previous chapter showed that the overall performance of the MLN model is good in the simulated network environment. One thing to consider is that the weight learning process is quite slow because of the large size of the generated model. Fortunately, the performance of the model stabilises after a few iterations, so the model weights do not have to be repeatedly updated, unless the environment changes radically or the set of formulas in the model is changed. One solution would be to try to prune the model by removing irrelevant formulas from the model. Identifying irrelevant formulas in the model requires future research.

The model that we used for experimentation is relatively simple. It only contains a couple of KPIs and parameters. In a real mobile network, the number of different measurements and parameters is much larger and all such quantities are infeasible to be included in a single model. Otherwise the size of the MLN model and the ground Markov network becomes too large very fast. Therefore the relevant measurements and parameters to be used for a particular use case need to be carefully selected. Another thing to consider is the scope of the formulas. In the previous chapter, the scope was either a single cell or a pair of cells. If the model has formulas with large scopes, the ground Markov network, again, becomes very large. Finally, in
a real environment the whole network should not be optimised together, but
different parts of the network should be analysed separately.

One limitation of our MLN method is that the predicates take only
boolean values. One consequence is that the KPI values need to be cate-
gorised and we lose information of the magnitude of the values. Determining
suitable category limits also requires sophisticated analysis, as discussed in
Section 4.1.1. Another consequence is that the objectives can not be priori-
tised. Each objective has equal influence on the inference even though some
of them could be more critical than the others. One possible solution to this
problem could be Hybrid Markov Logic Networks [32], where the real-valued
functions can be used in the model together with boolean predicates.
Chapter 7

Summary and Conclusions

We presented a method for cognitive management of mobile networks using MLN. We showed that the method made it possible to combine certain and uncertain knowledge from different sources into a single, compact, representation. We also showed how the model could be used for inference and how the model parameters can be learned from data. We experimented how the model performs in a simulated network environment. We also discussed the limitations of the model when applied in a real network.

The experimental results show that the model performs well in the simulated environment. They show that increasing the size of historical sample data does not necessarily lead to better results, but that we get good results already with a moderate sample size. They also show that because of the large size of the generated model, the weight learning process is quite slow. In a real network setting the size of the model needs to be considered.

Possibilities for future research include reducing the model size by pruning the set of formulas and prioritising the objectives based on, for example, operator goals and the distance of the KPIs from the target values.
Bibliography


37


