Steffen Hölldobler
Andrey Malikov
Christoph Wernhard
(Eds.)

Young Scientists' International Workshop on Trends in Information Processing (YSIP)

Proceedings

North-Caucasus Federal University
Stavropol, Russian Federation
April 22–25, 2014

Co-located with the Sixth International Conference on Infocommunicational Technologies in Science, Production and Education (INFOCOM-6)

Volume Editors

Steffen Hölldobler
International Center for Computational Logic
Technische Universität Dresden
01062 Dresden, Germany
email: sh@iccl.tu-dresden.de

Andrey Malikov
Institute for Information Processing and Telecommunication
North-Caucasus Federal University
Stavropol
Russian Federation
email: Malikov@ncstu.ru

Christoph Wernhard
International Center for Computational Logic
Technische Universität Dresden
01062 Dresden, Germany
email: christoph.wernhard@tu-dresden.de

Preface

The idea to organize an international workshop for young researchers was born during a discussion within the organization committee of the International Conference on Infocommunicational Technologies in Science, Production and Education (INFOCOM-6). In particular, we wanted to bring together master and PhD students from Russia and Europe to present and to discuss their new scientific results in Information Processing.

We have accepted eight technical papers and a tutorial for presentation at the workshop and publication in these proceedings. The papers were reviewed by an international program committee and we would like to thank Viktorija Drozdova, Ulrich Furbach, Joáo Leite, Igor Mandriza, Sergei Obiedkov, Josef Schneeberger, and Sergio Tessaris for providing the reviews.

We would also like to thank the North-Caucasus Federal University and, in particular, Oxana Mesentseva, Tatyana Kortchagina and her team of the International Department as well as Julia Komarova for their help and support in organizing this event.

Steffen Hölldobler
Andrey Malikov
Christoph Wernhard

Dresden and Stavropol
April 2014

Chairs

Steffen Hölldobler	Technische Universität Dresden, Dresden, Germany
Andrey Malikov	North-Caucasus Federal University, Stavropol, Russian Federation
Christoph Wernhard	Technische Universität Dresden, Dresden, Germany

Program Committee

Viktorija Drozdova	North-Caucasus Federal University, Stavropol, Russian Federation
Ulrich Furbach	Universität Koblenz-Landau, Koblenz, Germany
Steffen Hölldobler	Technische Universität Dresden, Dresden, Germany
Joáo Leite	Universidade Nova de Lisboa, Lisbon, Portugal
Andrey Malikov	North-Caucasus Federal University, Stavropol, Russian Federation
Igor Mandriza	North-Caucasus Federal University, Stavropol, Russian Federation
Sergei Obiedkov	Higher School of Economics, Moscow, Russian Federation
Josef Schneeberger	University of Applied Science Deggendorf, Deggendorf, Germany
Sergio Tessaris	Free University of Bozen-Bolzano, Bolzano, Italy
Christoph Wernhard	Technische Universität Dresden, Dresden, Germany

Table of Contents

Review of Modern Techniques of Qualitative Data Clustering

Sergey Cherevko and Andrey Malikov

The North Caucasus Federal University, Institute of Information Technology and Telecommunications

cherevkosa92@gmail.com, malikov@ncstu.ru

Abstract: This article made a brief comparative survey of modern clustering algorithms quantitative and qualitative data. As a practical component to the process of analysis of algorithms used in the task of analyzing the consumer basket. The examples of commercial use of clustering algorithms, described the current problems of using cluster analysis.

Keywords: cluster analysis, qualitative data, algorithm Clope, Kohonen‘s maps, overview of clustering algorithms

1 Introduction

In general, the concept of clustering and cluster analysis can be described as segmentation of a set of objects into different groups, called clusters. Affinity between objects from a same cluster should be higher than affinity between objects from different clusters. In the process of applying the techniques of clustering in the bulk of the tasks the number of clusters is unknown in advance - this characteristic is defined in the algorithm, but it's worth noting that there are algorithms that require initial quantification of finite groups.

The possibility of applying different clustering algorithms and their performance is determined by the following set of indicators:

1. Type of data to be clustered. Conditionally data types are divided into numerical continuous (values of a certain period: adult height, projectile range), discrete numerical (values from a list of some specific numbers: the number of children in the family, the number of clients seeking day) and categorical (descriptive characteristic feature of any object).

2. Dimensionality of the data set to be clustered. In accordance with this feature, you can select the algorithms that work only with sets of small sizes (for large samples, the effectiveness of these techniques falls due to low performance of the algorithm), and algorithms designed to handle large and very large data sets (in this case, however, quality of created clusters may suffer) [1].

3. Dimensionality of the data set being evaluated. If the set contains a large amount of emissions, some algorithms can give incorrect results misrepresent the nature and composition of the clusters.

2 Descriptive Comparison of Clustering Algorithms

If we talk about the current level of clustering techniques, it should be noticed that the range of techniques regarded to the analysis of quantitative data is wide enough and allows you to perform the tasks of clustering in accordance with all requirements to processing speed and quality of the final result. The most well-known representatives of numeric data clustering algorithms are:

1. Algorithm Clustering Using Representatives.
2. Algorithm Balanced Iterative Reducing and Clustering using Hierarchies.
3. Algorithm HCM (Hard C - Means).

Usage of the aforementioned methods for clustering categorical data is inefficient, and often impossible. The main difficulties are associated with high dimensionality and huge volumes of data that often characterizes such databases, because the pairwise comparison of the characteristics of objects from multi-million records database tables can take quite a long period of time. Clustering algorithms of aforementioned types are currently having a number of special requirements - recommendations that will optimize performance when working with large amounts of data:

1. Ability to work in a limited amount of RAM.
2. The algorithm should work under the condition that the information from the database can be gathered only in the forward-only cursor.
3. Ability to abort the algorithm with preservation of intermediate results with the possibility of resuming the process of data processing.
4. Minimizing the number of requests for the full database table scan [2].

Currently, the most promising and popular clustering algorithms qualitative data include:

Currently, the most promising and popular algorithms for clustering qualitative data include:

1. Algorithm Clope. The main advantages of this algorithm are speed and quality of clustering which are achieved by using the estimated global optimization criterion based on the maximization of the gradient histogram cluster height. During its operation, the algorithm stores a small amount of information on each of the clusters in memory, and requires a minimum number of data sets scans. Number of clusters is automatically selected by the algorithm based on the coefficient of repulsion, which was originally set by the user (the more the level of the coefficient is, the lower the level of similarity will be, and more clusters will be generated). [3]

2. Algorithm LargeItem, which was developed in 1999 as an optimized algorithm for clustering data sets based on the appraised, absolute function that uses the support parameter.

3. Kohonen maps (this method is applicable to both qualitative and quantitative indicators). Allows you to identify useful and non-trivial patterns, consider the influence of hundreds of factors, visualize complex multidimensional clusters in the form of clear and accessible maps.

4. Hierarchical clustering, as an example of MST (Algorithm based on Minimum Spanning Trees). MST algorithm first builds minimum spanning tree on the graph, and then sequentially removes edge with the largest weight. The disadvantage is the high degree of dependence on emissions contained in the data sets [4].

3 Comparison of Algorithms Clope and Kohonen‘s Maps for the Problem of the Consumer Basket

For a more practical comparison of algorithms we will describe a typical clustering problem: the consumer's basket analysis. The aim is to define a set of products that are most often found in one check (or order).

3.1 Algorithm Clope

"Clope" algorithm functionality is based on the idea of maximizing the global cost function, which increases the degree of similarity of transactions in the clusters by increasing the parameter of cluster histogram. Consider a simple example, we have 5 consumer checks:

1. {[bread, milk]}.
2. {[bread, milk, sour cream]}.
3. {[bread, sour cream, cheese]}.
4. {[cheese, sausage]}.
5. {[cheese, sausage, peppers]}.

Suppose that we already have two partitions into clusters:

1. {[bread, milk, bread, milk, sour cream, bread, sour cream, cheese] [cheese, sausage, cheese, sausage, peppers]}.
2. {[bread, milk, bread, milk, sour cream}, {[bread, sour cream, cheese, cheese, sausage, cheese, sausage, peppers]}.

Let's calculate the height (H) and width of the cluster (W) for both clusters.

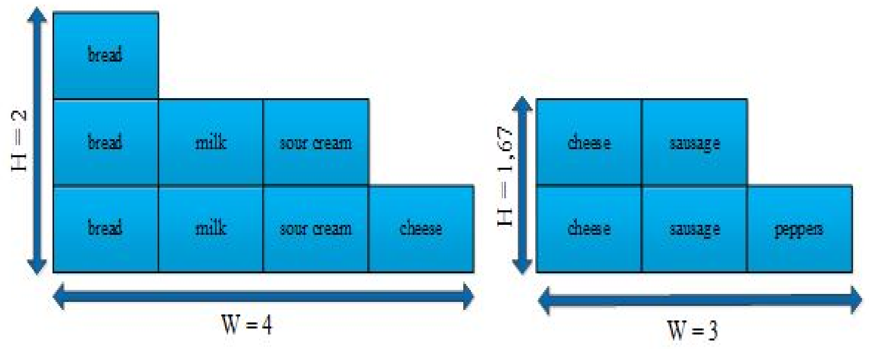


Figure 1 - distribution of the first cluster

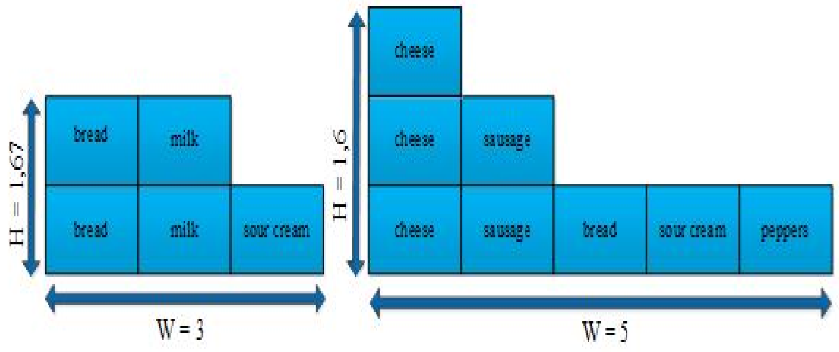


Figure 2 - distribution of the second cluster

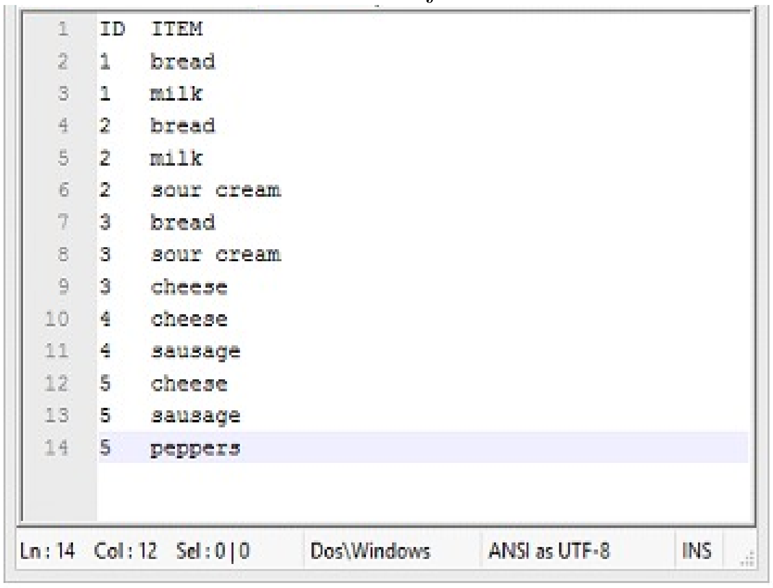
H - height of the chart, is calculated as the average height (density) of all the columns in the chart. For example, for the first cluster we got H = 2 as a result of the following arithmetic operation:

(3 +2 +2 +1) / 4 = 2.

W - the width of the cluster, which is equal to the number of columns in the cluster. To determine the quality of the partition clusters by Clope it is necessary to calculate cluster diagram parameter (H / W) for both distributions. For the first cluster it will be (2 + 1.67) / (3 + 4) = 0.52. For the second cluster, respectively (1.67 + 1.67) / (3 + 5) = 0.41. Obviously, the first partition is better option because cluster diagram parameter is higher, which indicates that the transaction have a large overlap with each other.

3.2 Kohonen's Maps

Let's try to perform this distribution using Kohonen maps. Compose a text file containing information about the checks. Column "Id" corresponds to the check box and ”Item” describes the subject contained in the check.



```
ID  ITEM
1   bread
1   milk
2   bread
2   milk
2   sour cream
3   bread
3   sour cream
3   cheese
4   cheese
4   sausage
5   cheese
5   sausage
5   peppers
```

Figure 3 - The composition of a text file for processing

Processing options in the input file define that we want to get 2 clusters as a result of processing, composed on the basis of field ”Item”. Neurons odds left by default, without changing the coefficients. The result of the processing is following (see Figure 4, 5).

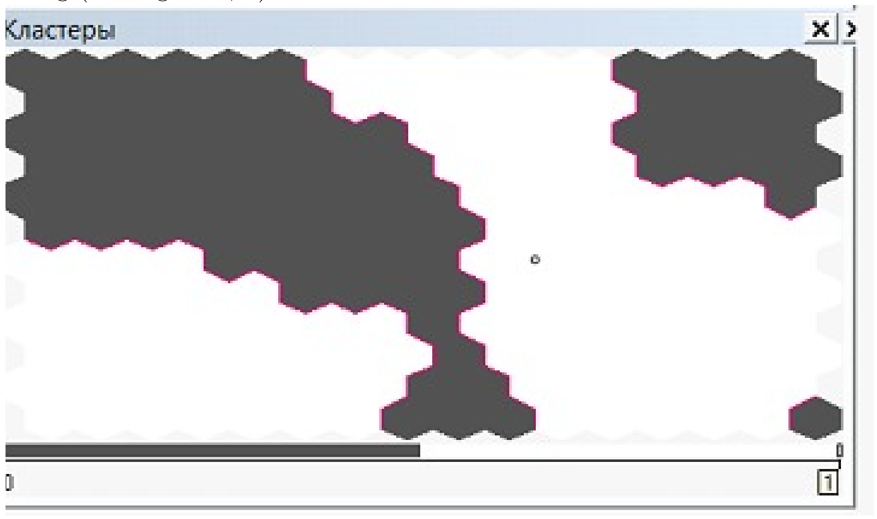


Figure 4 - a graphic representation of the clusters

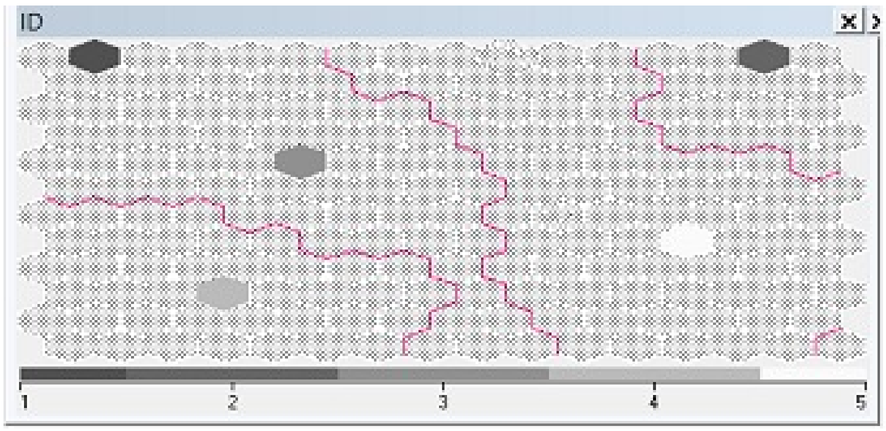


Figure 5 - Distribution of the consumer basket items in clusters

According to the results of distribution two clusters were allocated, the first checks were 1,2,3, the second 4,5. Distribution is obtained similarly to distribution provided by Clope algorithm. Basic formula, in which the distribution is a function of the neighborhood, allows us to define a "measure neighborhood" nodes compared:

$h_{ci}(t) = \alpha(t) \times exp(-\frac{||r_c - r_i||^2}{2\sigma^2(t)})$, where $0 < \alpha(t) < 1$ is learner and decreasing factor, r_i and r_j are coordinates of nodes $M_i(t)$ and $M_c(t)$. The last step of the algorithm is to change the vectors of weights on approaching the observation under consideration:

$m_i(t) = m_i(t-1) + h_{ci}(t) \times (x(t) - m_i(t-1))$.

As the results of the algorithm, the user has a visual map, where similar indicators will be grouped into visually isolated clusters.

4 Conclusion

Nowadays there is enough of active practical usage of clustering methods, as it makes work with homogeneous data sets much easier and allows the usage of specific methods of treatment which are not suitable for the total sample. In fact, the bigger and bigger funds are conducted by different companies around the globe to provide scientific segmentation and clustering of customers and end-products. The most famous events in this direction are: "Svyaznoi" company with their program "Svyaznoi Club", Bank of Moscow with the targeted marketing automation system SAS Marketing Automation, "PSB", "Tinkoff" bank and many others. For example, the company "Svyaznoi" in its loyalty program "Svyaznoi Club" implemented deep customer segmentation - cardholders of "Svyaznoi Club". Specialists from BaseGroup Labs and analytical platform Deductor were involved to solve this problem. They helped to process the entire array of raw data, and built more than 120 classification models for client's response per share. Despite the fairly extensive use of qualitative data clustering methods there are a number of topical issues that hinder the work and further developments: 1. Lack of comparable treatment for describing objects with attributes of different nature, measured in different units. 2. The complexity of making a formalized description of objects having both qualitative and quantitative traits. 3. The problem of detecting and displaying the artifacts in the source data. The need to develop such clustering method that would identify artifacts in the source data and either exclude them from the final result, or carry specific processing to visually represent the fact of "break out".

References

[1] Article Ke Wang, ChuXu, Bing Liu Clustering Transactions Using Large Items 2003

[2] Yang, Y., Guan, H., You. J. CLOPE: A fast and Effective Clustering Algorithm for Transactional Data In Proc. of SIGKDD'02, July 23-26, 2002, Edmonton, Alberta, Canada.

[3] Classification and comparison of clustering methods / IM transmission line protected [electronic resource].

[4] F. Hooper, F.Klawonn, R.Kruse, T.Runkler, Fuzzy Clustering. Chichester, United Kingdom: Wiley, 1999

Advanced Petri Nets and the Fluent Calculus

Steffen Hölldobler and Ferdian Jovan

International Center for Computational Logic
Technische Universität Dresden, 01062 Dresden, Germany
sh@iccl.tu-dresden.de ferdian.jovan@gmail.com

Abstract. In this paper we discuss conjunctive planning problems in the context of the fluent calculus and Petri nets. We show that both formalisms are equivalent in solving these problems. Thereafter, we extend actions to contain preconditions as well as obstacles. This requires to extend the fluent calculus as well as Petri nets. Again, we show that both extended formalisms are equivalent.

1 Introduction

It is widely believed that humans generate models and reason with respect to these models [14]. It is less widely believed that logics can be used to adequately model human reasoning [3]. Based on ideas first presented in [18], Hölldobler and Kencana Ramli [10] have developed a logic based on weakly completed logic programs and interpreted under the three-valued Łukasiewicz semantics; this logic was shown to adequately model human reasoning scenarios like the suppression and the selection task by generating a least model of an appropriate logic program and reasoning with respect to this least model [6,7]. Moreover, it was shown that there is a connectionist realization of this approach based on the core method [11,1].

However, human reasoning is much more complex than the above mentioned scenarios and involves – among others – reasoning about actions and causality including compositionality, concurrency, quick reactions, and resilience in the face of unexpected events. An architecture for such actions was developed in [2] based on extended Petri nets. Unfortunately, there is a huge gap between Petri nets and the logic developed by Hölldobler and Kencana Ramli and it is not at all obvious how the two approaches can be combined. Moreover, a close inspection of [2] revealed that some concepts are only specified procedurally.

A central notion in Petri nets are tokens which are consumed and produced when executing an action. Likewise, in the equational logic programming approach to actions and causality presented in [12] resources are used. The approach was later called *fluent calculus* in [19]. The logic programs in the fluent calculus admit least models and reasoning is performed with respect to these models. Hence, the fluent calculus seems to be a promising candidate to add reasoning about actions and causality to the human reasoning approach of Hölldobler and Kencana Ramli.

The goal of the research presented in this paper is to understand the relation between the fluent calculus and the extended Petri networks used in [2]. To this end, we will rigorously define various classes of planning problems, we will map these problems into Petri nets and into the fluent calculus, and we formally prove that there is a one-to-one correspondence between the two approaches in solving such problems.

The paper is structured as follows: Following the introduction in Section 1 we will present main notions and notations in Section 2. Conjunctive and advanced planning problems are discussed in Sections 3 and 4. In the final Section 5 we will discuss our results and point to future work. Due to lack of space we cannot include proofs; they are worked out in detail in [15] if not stated otherwise.

2 Preliminaries

Multisets Multisets are generalizations of sets, where members may occur more than once. In this paper, multisets are depicted with the help of the parenthesis $\dot{\{}$ and $\dot{\}}$. $\dot{\emptyset}$ denotes the empty multiset and $\dot{\cup}$, $\dot{\cap}$, $\dot{\subseteq}$, $\dot{\setminus}$, $\doteq$, and $\dot{\neq}$ denote the usual operations and relations on multisets, viz. multiset-union, -intersection, -subset, -difference, -equality, -inequality, respectively. Moreover, $x \in_k \mathcal{M}$ holds if and if x occurs exactly k times in the multiset $\mathcal{M}$, where $k \in \mathbb{N}$.

Petri Nets A *Petri net* is a tuple $(\mathcal{P}, \mathcal{T}, \mathcal{F})$, where $\mathcal{P}$ and $\mathcal{T}$ are finite sets called *places* and *transitions*, respectively, $\mathcal{P} \cap \mathcal{T} = \emptyset$, and $\mathcal{F} \dot{\subseteq} (\mathcal{P} \times \mathcal{T}) \dot{\cup} (\mathcal{T} \times \mathcal{P})$. A *marking* is a finite multiset $\mathcal{M}$ over $\mathcal{P}$; its elements are called *tokens*. The *pre-set* $\bullet t$ of $t \in \mathcal{T}$ is a finite multiset with $p \in_k \bullet t$ iff $p \in \mathcal{P} \wedge (p, t) \in_k \mathcal{F}$. The *post-set* $t\bullet$ of $t \in \mathcal{T}$ is a finite multiset with $p \in_k t\bullet$ iff $p \in \mathcal{P} \wedge (t, p) \in_k \mathcal{F}$.

Let $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F})$ be a Petri net and $\mathcal{M}$, $\mathcal{M}'$, and $\mathcal{M}''$ be markings. $t \in \mathcal{T}$ is *enabled* at $\mathcal{M}$ in $\mathcal{N}$ iff $\bullet t \dot{\subseteq} \mathcal{M}$; an enabled transition t can *fire* leading to $\mathcal{M}'$, denoted by $\mathcal{M} \xrightarrow{[t]} \mathcal{M}'$, where $\mathcal{M}' \doteq (\mathcal{M} \dot{\setminus} \bullet t) \dot{\cup} t\bullet$. *Firing sequences* are inductively defined as follows: $\mathcal{M} \xrightarrow{[]} \mathcal{M}$; if $\mathcal{M} \xrightarrow{[t]} \mathcal{M}'$ and $\mathcal{M}' \xrightarrow{w} \mathcal{M}''$ then $\mathcal{M} \xrightarrow{[t|w]} \mathcal{M}''$, where w is a list of transitions. A *firing sequence from $\mathcal{M}$ to $\mathcal{M}'$ of $\mathcal{N}$* is a firing sequence which starts from $\mathcal{M}$ and yields $\mathcal{M}'$.

Equational Logic Programming We assume the reader to be familiar with first-order predicate logic with equality and, in particular, with equational logic programming as, for example, presented in [9,17,13].

Fluents and Fluent Terms In planning, the notion of a fluent is often used to describe an item which may be present in one state but not in the next state. In the fluent calculus, *fluents* are non-variable terms built over some alphabet like a, $f(a)$, or $f(X)$, where a is a constant, f a function symbol, and X a variable; this alphabet must not contain the binary function symbol $\circ$ and the constant 1 as these symbols are used to represent multisets of fluents; *ground fluents* are fluents which do not contain an occurrence of a variable (e.g., a and

$f(a)$); *simple fluents* are fluents which are constants (e.g., a). The set of *fluent terms* is the smallest set satisfying the following conditions: 1 is a fluent term; each fluent is a fluent term; if s and t are fluent terms, then so is $s \circ t$. As the sequences of fluents occurring in a fluent term is not important, we consider $\circ$ to be associative and commutative, and 1 to be a unit element with respect to $\circ$; let $\mathcal{K}_{AC1}$ be the corresponding equational axioms plus the axioms of equality.

There is a one-to-one correspondence between equivalence classes of fluent terms with respect to $\mathcal{K}_{AC1}$ and multisets of fluents as follows: Let t be a fluent term and $\mathcal{M}$ a multiset of fluents in:

$$t^I = \begin{cases} \dot{\emptyset} & \text{if } t = 1, \\ \dot{\{}t\dot{\}} & \text{if } t \text{ is a fluent}, \\ u^I \dot{\cup} v^I & \text{if } t = u \circ v, \end{cases} \quad \text{and} \quad \mathcal{M}^{-I} = \begin{cases} 1 & \text{if } \mathcal{M} \doteq \dot{\emptyset}, \\ s \circ \mathcal{M}'^{-I} & \text{if } \mathcal{M} \doteq \dot{\{}s\dot{\}} \dot{\cup} \mathcal{M}'. \end{cases}$$

3 Conjunctive Planning Problems

Conjunctive planning problems were considered in [8] to relate the fluent calculus to the linear connection method and to linear logic. Here, we consider a slightly simplified version in that we restrict fluents to simple fluents.

A *conjunctive planning problem (CPP)* is a tuple $(\mathcal{I}, \mathcal{G}, \mathcal{A})$, where $\mathcal{I}$ and $\mathcal{G}$ are finite multisets of simple fluents called *initial* and *goal state*, respectively, and $\mathcal{A}$ is a finite set of actions; an *action* is an expression of the form $a : \mathcal{C} \Rightarrow \mathcal{E}$, where a is the *name* of the action and $\mathcal{C}$ and $\mathcal{E}$ are finite multisets of simple fluents called *conditions* and *(immediate) effects*, respectively.

Let $\mathcal{S}$ be a finite multiset of simple fluents; such multisets are called *states* in the sequel. An action $a : \mathcal{C} \Rightarrow \mathcal{E}$ is *applicable* to $\mathcal{S}$ iff $\mathcal{C} \dot{\subseteq} \mathcal{S}$; its *application* leads to the state $(\mathcal{S} \setminus \mathcal{C}) \dot{\cup} \mathcal{E}$. A *plan* is a sequence or list of actions; it transforms state $\mathcal{S}$ into $\mathcal{S}'$ if and only if S' is the result of successively applying the actions occurring in the plan to $\mathcal{S}$. A plan is a *solution* to a CPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$ if and only if it transforms $\mathcal{I}$ into $\mathcal{G}$.

To illustrate CPPs, consider a situation where a man living in an apartment becomes severely ill and calls the ambulance. The ambulance men decide that he needs to undergo treatment in a hospital and carry him on a stretcher to the ambulance. Finally, the ambulance car is driven to the hospital. This problem is considered as a CPP with fluent ill (denoting the ill man), apt (denoting that he is in his apartment), amb (denoting that the patient is in the ambulance car), and hos (denoting that the patient is in the hospital). The action names are c (the ambulance men are carrying the patient to the ambulance car) and d (driving to the hospital). Alltogether we obtain a CPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$*, where* $\mathcal{I} \doteq \dot{\{}ill, apt\dot{\}}$, $\mathcal{G} \doteq \dot{\{}ill, hos\dot{\}}$*, and*

$$\mathcal{A} \doteq \{c : \dot{\{}ill, apt\dot{\}} \Rightarrow \dot{\{}ill, amb\dot{\}},\ d : \dot{\{}ill,\ amb\dot{\}} \Rightarrow \dot{\{}ill, hos\dot{\}}\}.$$

The goal state $\mathcal{G}$ *is reached from the initial state* $\mathcal{I}$ *by applying first c yielding the intermediate state* $\dot{\{}ill, amb\dot{\}}$ *and, thereafter, applying d.*

3.1 Conjunctive Planning Problems in the Fluent Calculus

This subsection is based on [12], where an equational logic programming solution to CPPs was presented. For each action $a : \mathcal{C} \Rightarrow \mathcal{E}$ in a given CPP a fact

$$\mathit{action}(\mathcal{C}^{-I}, a, \mathcal{E}^{-I})$$

is specified; let $\mathcal{K}_A$ be the set of all facts of this form for a given CPP. For the running example we obtain

$$\mathcal{K}_A = \{\mathit{action}(\mathit{ill} \circ \mathit{apt}, c, \mathit{ill} \circ \mathit{amb}), \mathit{action}(\mathit{ill} \circ \mathit{amb}, d, \mathit{ill} \circ \mathit{hos})\}$$

The applicability of an action is specified by

$$\mathit{applicable}(C \circ Z, A, E \circ Z) \leftarrow \mathit{action}(C, A, E),$$

where Z is a variable which will be used to collect all fluents of a state which are not affected by the application of an action. Causality is specified with the help of a ternary predicate symbol *causes*. Intuitively, $\mathit{causes}(X, P, Y)$ is used to represent that the execution of plan P in state X transforms X into state Y. *causes* is specified inductively on the structure of plans, i.e., lists of actions:

$$\begin{array}{l} \mathit{causes}(X, [\,], X) \\ \mathit{causes}(X, [A|P], Y) \leftarrow \mathit{applicable}(X, A, U) \wedge \mathit{causes}(U, P, Y) \end{array}$$

Let K_C be the set containing the clauses for *applicable* and *causes*. A CPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$ can now be represented in the fluent calculus by the question of whether

$$\mathcal{K}_A \cup \mathcal{K}_C \cup \mathcal{K}_{AC1} \models (\exists P)\, \mathit{causes}(\mathcal{I}^{-I}, P, \mathcal{G}^{-I}),$$

and SLDE-resolution can be applied to compute an answer substitution for P encoding a solution for the CPP if it is solvable.

Let $\mathcal{FC}_Q$ denote the *fluent calculus representation* of a CPP Q. The following theorem is proven in [8]:

Theorem 1. *Let Q be a CPP. The following statements are equivalent for a plan p:*

1. *p is a solution for Q.*
2. *p is generated by SLDE-resolution for $\mathcal{FC}_Q$.*

3.2 Conjunctive Planning Problems in Petri Nets

Let $Q = (\mathcal{I}, \mathcal{G}, \mathcal{A})$ be a CPP. The Petri net $\mathcal{N}_Q = (\mathcal{P}, \mathcal{T}, \mathcal{F})$ together with the markings $\mathcal{I}$ and $\mathcal{G}$ is the *Petri net representation* of Q, where $\mathcal{P}$ is the set of all simple fluents occurring in Q, $\mathcal{T}$ is the set of all action names occurring in Q, $(p, t) \in_k \mathcal{F}$ if and only if $t : \mathcal{C} \Rightarrow \mathcal{E} \in \mathcal{A}$ such that $p \in_k \mathcal{C}$, and $(t, p) \in_k \mathcal{F}$ if and only if $t : \mathcal{C} \Rightarrow \mathcal{E} \in \mathcal{A}$ such that $p \in_k \mathcal{E}$.

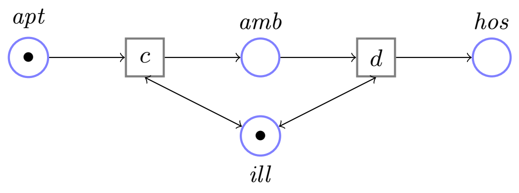


Fig. 1. A Petri net for the ill man problem with initial marking $\dot{\{}apt, ill\dot{\}}$.

One should observe that for each action $a : \mathcal{C} \Rightarrow \mathcal{E}$ in $\mathcal{A}$ we find a transition $a \in \mathcal{T}$ with $\bullet a \doteq \mathcal{C}$ and $a\bullet \doteq \mathcal{E}$. Conversely, whenever a transition t is enabled in $\mathcal{N}_Q$ given a marking $\mathcal{M}$, then there exists an action with name t in $\mathcal{A}$ which is applicable in $\mathcal{M}$.

The question of whether there exists a plan p solving a CPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$ is the question of whether there exists a firing sequence from $\mathcal{I}$ to $\mathcal{G}$ in $\mathcal{N}_Q$. The Petri net for the running example is depicted in Figure 1. One should observe that $[c, d]$ is a firing sequence leading from the $\dot{\{}apt, ill\dot{\}}$ to $\dot{\{}ill, hos\dot{\}}$.

3.3 Petri Net versus Fluent Calculus Representations

As a first result we extend Theorem 1. Throughout this subsection, let Q be the CPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$ and $\mathcal{FC}_Q$ and $\mathcal{N}_Q$ be its representations in the fluent calculus and in Petri nets, respectively.

Theorem 2. *The following statements are equivalent for a plan p:*

1. *p is a solution for Q.*
2. *p is generated by SLDE-resolution for $\mathcal{FC}_Q$.*
3. *p is a firing sequence from $\mathcal{I}$ to $\mathcal{G}$ in $\mathcal{N}_Q$.*

Proof. Because of Theorem 1 it suffices to show that the *2.* and *3.* are equivalent. By induction on the number of transitions occurring in p in can be shown that *3.* implies *2.*. The converse can be shown to hold by induction on the number of actions occurring in p.

4 Advanced Planning Problems

In conjunctive planning problems, all conditions of an action are consumed when the action is executed. In this section we extend planning problems to allow preconditions which are only tested when an action is executed but are not consumed and to allow obstacles which prevent an action from being executed even if its conditions are satisfied.

An *advanced conjunctive planning problem (ACPP)* is a tuple $(\mathcal{I}, \mathcal{G}, \mathcal{A})$, where $\mathcal{I}$ and $\mathcal{G}$ are finite multisets of simple fluents called *initial* and *goal state*, respectively, and $\mathcal{A}$ is a finite set of *advanced actions*; an advanced action is an expression of the form $a : \mathcal{C} \overset{\mathcal{R},\mathcal{O}}{\Longrightarrow} \mathcal{E}$, where a is the *name* of the action, $\mathcal{C}$, $\mathcal{R}$, $\mathcal{O}$ and $\mathcal{E}$ are multisets of simple fluents called *conditions*, *preconditions*, *obstacles*, and *effects*, respectively, and $\mathcal{C} \dot{\cap} \mathcal{E} \doteq \dot{\emptyset}$.

Let $\mathcal{S}$ be a state. An extended action $a : \mathcal{C} \overset{\mathcal{R},\mathcal{O}}{\Longrightarrow} \mathcal{E}$ is *applicable* to $\mathcal{S}$ if and only if $\mathcal{C} \dot{\subseteq} \mathcal{S}$, $\mathcal{R} \dot{\subseteq} \mathcal{S}$, and $\forall e \in_k \mathcal{O}(e \in_j \mathcal{S} \rightarrow j < k)$. Its application yields the state $(\mathcal{S} \dot{\setminus} \mathcal{C}) \dot{\cup} \mathcal{E}$. If the last condition in the definition of applicability to $\mathcal{S}$ is violated, i.e., if there is an extended action with name a, obstacles $\mathcal{O}$ and $e \in_k \mathcal{O}$, and $e \in_j \mathcal{S}$ such that $j \geq k$, then a is *hindered* in $\mathcal{S}$. *Plans* and *solutions* are defined as before.

To illustrate ACPPs we modify the running example by assuming that the patient was so fat that he did not fit through the appartment door. Hence, the ambulance men cannot carry him to the ambulance car. We introduce an additional fluent fat and obtain the ACPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$, where $\mathcal{I} \doteq \dot{\{}ill, fat, apt\dot{\}}$, $\mathcal{G} \doteq \dot{\{}ill, fat, hos\dot{\}}$, and

$$\mathcal{A} \doteq \{c : \dot{\{}apt\dot{\}} \xrightarrow{\dot{\{}ill\dot{\}}, \dot{\{}fat\dot{\}}} \dot{\{}amb\dot{\}},\ d : \dot{\{}amb\dot{\}} \xrightarrow{\dot{\{}ill\dot{\}}, \dot{\emptyset}} \dot{\{}hos\dot{\}}\}.$$

Obviously, this ACPP cannot be solved as the obstacle fat hinders the application of the action c. By the way, the ill man was later rescued with a help of a mobile cran, which carried him out of his apartment through a window.

4.1 Advanced Planning Problems in Petri Nets

Petri nets were extended by so-called inhibitory arcs, which may by weighted and which increase the modeling power of Petri nets to the level of Turing machines [16,5,4]. We combine inhibitor arcs with so-called test arcs, which were introduced in [2] to allow for places, which may contain real-valued or discrete tokens in order to enable an action, but which are not consumed.

An *advanced Petri net* is a tuple $(\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{H}, \mathcal{L})$, where $(\mathcal{P}, \mathcal{T}, \mathcal{F})$ is a Petri net, $\mathcal{H} \dot{\subseteq} \mathcal{P} \times \mathcal{T}$, and $\mathcal{L} \dot{\subseteq} \mathcal{P} \times \mathcal{T}$; $\mathcal{H}$ and $\mathcal{L}$ are the multiset of *inhibitory* and *test arcs*, respectively. The multiset $\blacktriangledown t$ of *inhibitory places of transition* $t \in \mathcal{T}$ is defined by $p \in_k \blacktriangledown t$ if and only if $p \in \mathcal{P} \wedge (p,t) \in_k \mathcal{H}$. Likewise, the multiset $\blacktriangle t$ of *test places of transition* t is defined as $p \in_k \blacktriangle t$ if and only if $p \in \mathcal{P} \wedge (p,t) \in_k \mathcal{L}$.

Let $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{H}, \mathcal{L})$ be an advanced Petri net and $\mathcal{M}$ a marking. $t \in \mathcal{T}$ is *enabled* at $\mathcal{M}$ in $\mathcal{N}$ if and only if $\bullet t \dot{\subseteq} \mathcal{M}$, $\forall p \in \mathcal{P}((p,t) \in_k \mathcal{L} \wedge p \in_j \mathcal{M} \rightarrow k \leq j)$, and $\forall p \in \mathcal{P}((p,t) \in_m \mathcal{H} \wedge p \in_n \mathcal{M} \rightarrow m > n)$. The notions *fire* and *firing sequence* are defined as before.

Let $Q = (\mathcal{I}, \mathcal{G}, \mathcal{A})$ be an ACPP. The Petri net $\mathcal{N}_Q^{\mathcal{A}} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathcal{H}, \mathcal{L})$ together with the markings $\mathcal{I}$ and $\mathcal{G}$ is the *Petri net representation* of Q, where $\mathcal{P}$, $\mathcal{T}$, and $\mathcal{F}$ are defined as in Subsection 3.2, $(p,t) \in_k \mathcal{H}$ if and only if $\exists(t : \mathcal{C} \overset{\mathcal{R},\mathcal{O}}{\Longrightarrow} \mathcal{E}) \in \mathcal{A}$ such that $p \in_k \mathcal{O}$, and $(p,t) \in_k \mathcal{L}$ if and only if $\exists(t : \mathcal{C} \overset{\mathcal{R},\mathcal{O}}{\Longrightarrow} \mathcal{E}) \in \mathcal{A}$ such that $p \in_k \mathcal{R}$. The Petri net for the modified running example is shown in Figure 2.

From this definition we learn that for every action $t : \mathcal{C} \overset{\mathcal{R},\mathcal{O}}{\Longrightarrow} \mathcal{E}$ in $\mathcal{A}$ we find a transition $t \in \mathcal{N}_Q^{\mathcal{A}}$ with $\blacktriangledown t \doteq \mathcal{O}$, $\blacktriangle t \doteq \mathcal{R}$, $\bullet t \doteq \mathcal{C}$, and $t\bullet \doteq \mathcal{E}$. One should observe

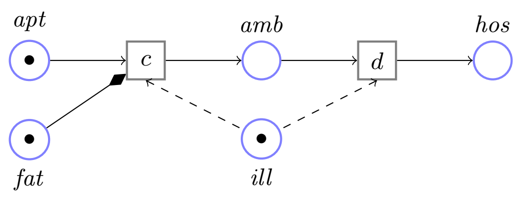


Fig. 2. The advanced Petri net for the modified running example with initial marking. Test arcs are depicted by dashed arrows, inhibitory arcs by arrows with a diamond head.

that the requirements for enabling a transition in $\mathcal{N}_{\mathcal{Q}}^{\mathcal{A}}$ are the requirements for the applicability of an action in $\mathcal{Q}$. Hence, a transition t is enabled at marking $\mathcal{M}$ in $\mathcal{N}_{\mathcal{Q}}^{\mathcal{A}}$ if and only if there exists an action t in $\mathcal{Q}$ with t being applicable in $\mathcal{M}$.

4.2 Advanced Planning Problems in the Fluent Calculus

To maintain the additional features of ACPPs several new facts and rules are added to the fluent calculus representation introduced in Subsection 3.1. For each advanced action $a : \mathcal{C} \xRightarrow{\mathcal{R},\mathcal{O}} \mathcal{E}$ and each obstacle $o \in_k \mathcal{O}$ the fact

$$\mathit{inhib}(\overbrace{o \circ \ldots \circ o}^{k \text{ times}}, a)$$

is added to $\mathcal{K}_A$. In addition, for each advanced action $a : \mathcal{C} \xRightarrow{\mathcal{R},\mathcal{O}} \mathcal{E}$ the fact

$$\mathit{precon}(\mathcal{R}^{-I}, A)$$

is added to $\mathcal{K}_A$; it is used in the (modified) definition of *applicable* to test whether all preconditions are met. For our modified running example we obtain

$$\begin{aligned}\mathcal{K}_A = \{\ & \mathit{action}(\mathit{apt}, c, \mathit{amb}),\ \ \mathit{action}(\mathit{amb}, d, \mathit{hos}),\\ & \mathit{inhib}(\mathit{fat}, c),\ \ \mathit{precon}(\mathit{ill}, c),\ \ \mathit{precon}(\mathit{ill}, d)\ \}.\end{aligned}$$

The rule

$$\mathit{hinder}(X \circ Z, A) \leftarrow \mathit{inhib}(X, A)$$

is added to $\mathcal{K}_C$ to prohibit the application of action A whenever sufficiently many obstacles X are present in a state $X \circ Z$. The definition of *applicable* in $\mathcal{K}_C$ is modified to

$$\begin{aligned}\mathit{applicable}(C \circ Z, A, E \circ Z) \leftarrow\ & \mathit{action}(C, A, E)\ \wedge\\ & \mathit{precon}(R, A)\ \wedge\ R \circ Y \approx C \circ Z\ \wedge\\ & \neg \mathit{hinder}(C \circ Z, A)\end{aligned}$$

where the subgoal $R \circ Y \approx C \circ Z$ is used to check whether the preconditions R of action A are satisfied in state $C \circ Z$. As the equality predicate $\approx$ is now used explicitly as a subgoal, the axiom of reflexivity

$$X \approx X$$

must be added to $\mathcal{K}_C$, effectively forcing the AC1-unification of the left-hand and the right-hand side of the subgoal $R \circ Y \approx C \circ Z$.

Let $\mathcal{K}_A^{\mathcal{A}}$ and $\mathcal{K}_C^{\mathcal{A}}$ be the modified sets of clauses for a given ACPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$. The ACPP can now be presented in the fluent calculus by the question of whether

$$\mathcal{K}_A^{\mathcal{A}} \cup \mathcal{K}_C^{\mathcal{A}} \cup \mathcal{K}_{AC1} \models (\exists P)\, \mathit{causes}(\mathcal{I}^{-I}, P, \mathcal{G}^{-I}),$$

and SLDENF-resolution can be applied to compute an answer substitution for P, if existing [13]. SLDENF-resolution is sound [17], but it is only shown to be complete if the completion of $\mathcal{K}_A^{\mathcal{A}} \cup \mathcal{K}_C^{\mathcal{A}} \cup \mathcal{K}_{AC1}$ is satisfiable and SLDENF-derivations neither flounder nor are infinite [13].

Lemma 3. *The completion of $\mathcal{K}_A^{\mathcal{A}} \cup \mathcal{K}_C^{\mathcal{A}} \cup \mathcal{K}_{AC1}$ is satisfiable.*

Proof. By construction of a model for the completion $\mathcal{K}_A^{\mathcal{A}} \cup \mathcal{K}_C^{\mathcal{A}} \cup \mathcal{K}_{AC1}$.

Regarding the question of whether SLDENF-derivations flounder or are infinite we observe that the definition of *causes* is recursive in the second argument, which is a list. If the length of this list is known in advance and the first argument of *causes* is a ground fluent term (which holds by the definition of a planning problem), then SLDENF-derivations neither flounder nor are infinite.

Proposition 4. *Let s be a ground fluent term and a the name of an action. Then, each SLDENF-derivation of $\leftarrow \mathit{hinder}(s, a)$ is finite.*

Proof. Follows immediately from the definition of *hinder* and *inhib*.

Proposition 5. *No SLDENF-derivation of $\leftarrow \mathit{causes}(\mathcal{I}^{-I}, [A_1, \ldots, A_n], \mathcal{G}^{-I})$ flounders or is infinite.*

Proof. By induction on n.

Based on this result we must refine the fluent calculus representation of an ACPP to the question of whether

$$\mathcal{K}_A^{\mathcal{A}} \cup \mathcal{K}_C^{\mathcal{A}} \cup \mathcal{K}_{AC1} \models (\exists A_1, \ldots, A_n)\, \mathit{causes}(\mathcal{I}^{-I}, [A_1, \ldots, A_n], \mathcal{G}^{-I}).$$

and iteratively increase n in the search for a solution of the planning problem.

Finally, we show that *hinder* prevents actions from being applicable:

Proposition 6. *There are enough obstacles in a state $\mathcal{S}$ to hinder an advanced action a if and only if there is an SLDENF-resolution proof of $\leftarrow \mathit{hinder}(\mathcal{S}^{-I}, a)$.*

Proof. Follows from the definitions of *hinder* and *inhib*.

4.3 Petri Nets versus Fluent Calculus Representations

Throughout this subsection, let Q be the ACPP $(\mathcal{I}, \mathcal{G}, \mathcal{A})$, $\mathcal{FC}_Q^{\mathcal{A}}$ and $\mathcal{N}_Q^{\mathcal{A}}$ be its representations in the advanced fluent calculus and the advanced Petri nets, respectively, and p be the plan $[a_1, \ldots, a_n]$.

Theorem 7. *The following statements are equivalent for a plan p:*

1. *p is a solution for Q.*
2. *p is generated by SLDENF-resolution for* $\mathcal{FC}_Q^{\mathcal{A}}$*.*
3. *p is a firing sequence from* $\mathcal{I}$ *to* $\mathcal{G}$ *in* $\mathcal{N}_Q^{\mathcal{A}}$*.*

Proof. The theorem is obtained if we can prove that *1.* implies *2.*, *2.* implies *3.*, and *3.* implies *1.* These implications can be shown by inductions on the length of the plan p, on the length of the SLDENF-resolution refutation, and on the length of the firing sequence, respectively.

5 Discussion

In this paper we have shown that there is a close correspondence between Petri nets and the fluent calculus for conjunctive planning problems. This correspondence is preserved if we extended Petri nets and the fluent calculus by test and inhibitory arcs. The correspondence can be even further extended to planning problems with fluents containing real values as investigated in [15]. In [2], it was shown that Petri nets can be combined with Bayes nets via real-valued fluents, and, hence, it should now be possible to combine the fluent calculus and Bayes nets. However, this needs to be rigourously investigated in the near future.

Whereas in this paper we were computing answer substitutions by SLDE- and SLDENF-resolution in the fluent calculus, we also like to invesigate the corresponding fixpoint characterization of the fluent calculus. This is the obvious next step in order to combine the human reasoning approach mentioned in the introduction with reasoning about actions and causality in the fluent calculus. Finally, the ultimate goal is a connectionist realization of the combined approach within the core method [11,1].

Acknowledgements We would like to thank Bertram Fronhöfer and Christoph Wernhard for many fruitful discussions and the anonymous referees for there comments.

References

1. S. Bader and S. Hölldobler. The core method: Connectionist model generation. In S. Kollias et. al., editor, *Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN)*, volume 4132 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2006.
2. L. Barrett. *An Architecture for Structured, Concurrent, Real-time Action.* PhD thesis, Computer Science Division, University of California at Berkeley, 2010.

3. R. M. J. Byrne. Suppressing valid inferences with conditionals. *Cognition*, 31:61–83, 1989.
4. R. David and H. Alla. On hybrid Petri nets. *Discrete Event Dynamic Systems*, 11(1-2):9–40, 2001.
5. J. Desel and W. Reisig. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, chapter Place/Transition Petri Nets, pages 122–173. Springer, 1998.
6. E.-A. Dietz, S. Hölldobler, and M. Ragni. A computational logic approach to the suppression task. In N. Miyake, D. Peebles, and R. P. Cooper, editors, *Proceedings of the 34th Annual Conference of the Cognitive Science Society*, pages 1500–1505. Cognitive Science Society, 2012.
7. E.-A. Dietz, S. Hölldobler, and M. Ragni. A computational logic approach to the abstract and the social case of the selection task. In *Proceedings Eleventh International Symposium on Logical Formalizations of Commonsense Reasoning*, 2013.
8. G. Große, S. Hölldobler, and J. Schneeberger. Linear deductive planning. *Journal of Logic and Computation*, 6(2):233–262, 1996.
9. S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence.* Springer, Berlin, 1989.
10. S. Hölldobler and C. D. P. Kencana Ramli. Logic programs under three-valued Łukasiewicz's semantics. In P. M. Hill and D. S. Warren, editors, *Logic Programming*, volume 5649 of *Lecture Notes in Computer Science*, pages 464–478. Springer Berlin Heidelberg, 2009.
11. S. Hölldobler and C. D. P. Kencana Ramli. Logics and networks for human reasoning. In C. Alippi, Marios M. Polycarpou, Christos G. Panayiotou, and Georgios Ellinasetal, editors, *Artificial Neural Networks – ICANN*, volume 5769 of *Lecture Notes in Computer Science*, pages 85–94. Springer Berlin Heidelberg, 2009.
12. S. Hölldobler and J. Schneeberger. A new deductive approach to planning. *New Generation Computing*, 8:225–244, 1990.
13. S. Hölldobler and M. Thielscher. Computing change and specificity with equational logic programs. *Annals of Mathematics and Artificial Intelligence*, 14:99–133, 1995.
14. P.N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness.* Cambridge University Press, Cambridge, 1983.
15. F. Jovan. Planning problems in Petri nets and fluent calculus. Master's thesis, TU Dresden, Faculty of Computer Science, 2014.
16. T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, 1989.
17. J. C. Shepherdson. SLDNF–resolution with equality. *Journal of Automated Reasoning*, 8:297–306, 1992.
18. K. Stenning and M. van Lambalgen. *Human Reasoning and Cognitive Science.* MIT Press, 2008.
19. M. Thielscher. Introduction to the fluent calculus. *Electronic Transactions on Artificial Intelligence*, 2:179–192, 1998.

Generic CDCL – A Formalization of Modern Propositional Satisfiability Solvers

Steffen Hölldobler, Norbert Manthey, Tobias Philipp and Peter Steinke

International Center for Computational Logic
Technische Universität Dresden

Abstract. Modern propositional satisfiability (or SAT) solvers are very powerful due to recent developments on the underlying data structures, the used heuristics to guide the search, the deduction techniques to infer knowledge, and the formula simplification techniques that are used during pre- and inprocessing. However, when all these techniques are put together, the soundness of the combined algorithm is not guaranteed any more. In this paper we present a small set of rules that allows to model modern SAT solvers in terms of a state transition system. With these rules all techniques which are applied in modern SAT solvers can be adequately modeled. Finally, we compare Generic CDCL with related systems.

1 Introduction

Many practical problems of computer science are in the complexity class NP. There are many well studied formalisms that can handle problems of this class, among them are constraint satisfaction [20], answer set programming [7], or satisfiability checking [3]. Although the two former formalisms admit a richer language, the latter approach is still very competitive even if the expressivity of its language is comparatively low.

The *propositional satisfiability problem* (SAT) consists of a propositional formula and asks whether there is a satisfying assignment for the Boolean variables occurring in the formula. From a complexity theory point of view SAT is NP-complete [4] and, thus, intractable. Still, there are many industrial and academic applications that can be solved nicely with modern SAT solvers. For instance a SAT-based railway scheduling software outperformed the native version [9]. Likewise, haplotype matching [13] can be solved nicely with modern SAT solvers.

The success of the SAT approach lies in the strength of today's SAT solvers. SAT solvers do not operate on testing all possible variable assignments, but on constructing an assignment by successively interleaving two processes, viz., guessing and propagating the assignment of literals. The main inference rule is *unit propagation*, an efficient form of *resolution*. Combined with a *decision* rule it is the core of the basic algorithm known as the *DPLL algorithm* [5]. In the case that a contradiction is found in the formula with respect to the current variable assignment, advanced SAT solvers backtrack and learn a conflict clause which prevents the current and similar conflicts. With the addition of so-called *learned clauses* the basic algorithm is known as *CDCL algorithm* [16].

Modern systematic SAT solvers are highly tuned and complex proof procedures employing many advanced techniques like *clause learning* [16], *non-chronological backtracking*, *restarts* [8], *clause removal* [2, 6], *decision heuristics* [2,17], and formula simplification techniques [12]. Specialized, cache-conscious data structures [10] further improve the performance. This way, today's solvers like `Riss`,[1] `MiniSAT` or `Lingeling` can handle formulas with millions of variables and millions of clauses.

However, the success of modern solvers carries a price tag: *increased code complexity*. Successful SAT solvers like the above mentioned ones consist of multiple thousand lines of code and are written in programming languages with side effects like C or C++. Due to the code complexity, the behavior of SAT solvers is hard to understand and state-of-the-art SAT solver internals are hard to teach. Moreover, finding additional techniques and integrating them into a SAT solver is getting more complex, as we have to consider the interplay with all the remaining techniques. Consequently, abstracting from specific algorithms, data structures, and heuristics is extremely important in order to discover and prove properties of a modern SAT solver as well as to understand the principles of SAT solving.

This problem was tackled by different formalizations, notably LINEARIZED DPLL [1], RULE-BASED SAT SOLVER DESCRIPTIONS [15], and ABSTRACT DPLL [18]. However, these systems do not appropriately model modern SAT solvers anymore. In particular, preprocessing and applying preprocessing techniques interleaved with search, known as *inprocessing*, became a crucial part in SAT solving. Applying formula simplification techniques also during search is an attractive idea since it allows to use valuable formula simplifications while taking learned clauses into account. For example, the SAT solver `Lingeling` benefits considerably from this approach.

The contribution of this paper is the formalism *Generic CDCL* that models the computation of modern SAT solvers. Equipped with a small set of simple state transition rules, we can model all well-established techniques like preprocessing, inprocessing, restarts, clause sharing, as well as clause learning and forgetting. This formalism allows us to reason about the behavior of SAT solvers independently of the specific implementation. Additionally, the framework is a first step to explain how modern SAT solvers are working in a compact and easy way. Besides the presentation of Generic CDCL, the main result of this paper is the proof that Generic CDCL and, consequently, all its instances are sound.

The paper is structured as follows: In Section 2 we describe basic concepts of satisfiability testing. We present Generic CDCL in Section 3, where we also prove that Generic CDCL correctly solves the satisfiability problems. Afterwards, we compare Generic CDCL with related formalism in Section 4 and we conclude the paper in Section 5.

[1] The SAT solver `Riss` is freely available at `tools.computational-logic.org`.

2 Preliminaries

2.1 The Satisfiability Problem

We assume a fixed infinite set $\mathcal{V}$ of Boolean *variables.* A *literal* is a variable v (*positive literal*) or a negated variable $\overline{v}$ (*negative literal*). The *complement* $\overline{x}$ of a positive (negative, resp.) literal x is the negative (positive, resp.) literal with the same variable as x. The complement of a set S of literals, denoted with $\overline{S}$, is defined as $\overline{S} = \{\overline{x} \mid x \in S\}$. Finite sets of clauses are called *formulas*, where a *clause* is a finite set of literals. Sometimes, we write a clause $\{x_1, \ldots, x_n\}$ also as the disjunction $(x_1 \vee \ldots \vee x_n)$ and a formula $\{C_1, \ldots, C_n\}$ as the conjunction $(C_1 \wedge \ldots \wedge C_n)$. The empty clause is denoted by $\bot$, the empty formula by $\top$. The formula obtained from F by replacing all occurrences of the variable v by the variable w is denoted by $F[v \mapsto w]$. The set of all variables occurring in a formula F (in positive or negative literals) is denoted by $\mathsf{vars}(F)$; the set of all literals occurring in F by $\mathsf{lits}(F)$. For instance, if $x, y \in \mathcal{V}$, then $F = \{\{\overline{x}, y\}, \{y\}\}$ is a formula, its alternative representation using logical connectives is $(\overline{x} \vee y) \wedge y$, $\mathsf{vars}(F) = \{x, y\}$, and $\mathsf{lits}(F) = \{\overline{x}, y\}$.

The semantics of formulas is based on the notion of an interpretation. An *interpretation* I is a set of literals which does not contain a complementary pair $x, \overline{x}$ of literals. An interpretation I is *total* iff for each $v \in \mathcal{V}$ either $v \in I$ or $\overline{v} \in I$. The satisfaction relation $\models$ is defined as follows: Let I be an interpretation, then $I \models \top$, $I \not\models \bot$, $I \models (x_1 \vee \ldots \vee x_n)$ iff $I \models x_i$ for some $i \in \{1, \ldots, n\}$, and $I \models (C_1 \wedge \ldots \wedge C_n)$ iff $I \models C_i$ for all $i \in \{1, \ldots, n\}$. Interpretation I is a *model* for the formula F iff $I \models F$. In the case that a formula F has a model, then F is *satisfiable*, otherwise it is *unsatisfiable.*

We relate formulas by three relations: the *entailment*, the *equivalence* and the *equisatisfiability* relation: Formula F *entails* formula F' iff every total model of F is a model of F'. Two formulas F and F' are *equivalent*, in symbols $F \equiv F'$, iff F entails F' and F' entails F. Two formulas F and F' are *equisatisfiable*, in symbols $F \equiv_{\mathsf{sat}} F'$, iff either both are satisfiable or both are unsatisfiable.

For instance, the interpretation $I = \{x, \ \neg z\}$ is a model of the formula $F_1 = (x \vee y) \wedge (\overline{x} \vee \overline{z})$ and, therefore, F is satisfiable. The formula $F_2 = x \wedge \overline{x}$ has no model and, therefore, is unsatisfiable. The formula $F_3 = x \wedge \overline{z}$ is satisfiable and, therefore, the formulas F_1 and F_3 are equisatisfiable, but the formulas F_1 and F_2 are not equisatisfiable. In fact, $F_3 \models F_1$ since every total model I of the formula F_3 must contain x and $\overline{z}$ and, hence, the two clauses of F_1 are satisfied by I. Finally, we find for all clauses C and formulas F that $C \vee \top \equiv \top \vee C \equiv \top$, $C \vee \bot \equiv \bot \vee C \equiv C$, $F \wedge \top \equiv \top \wedge F \equiv F$, and $F \wedge \bot \equiv \bot \wedge F \equiv \bot$.

Let x be a literal, $C = (x \vee C')$ and $D = (\overline{x} \vee D')$ be two clauses. Then the clause $(C' \vee D')$ is the *resolvent of the clauses C and D upon the literal x.* A *linear resolution derivation from the clause C to the clause D in the formula F* is a finite sequence of clauses $(C_i \mid 1 \leq i \leq n)$ such that $C_1 = C$, $C_n = D$ and C_i is a resolvent of the clause C_{i-1} and some clause in the formula F for all $i \in \{2, \ldots, n-1\}$; one should observe that F entails D and that the addition of entailed clauses to a formula preserves equivalence.

2.2 Variable Assignments and the Reduct Operator

Let J be a finite sequence of literals. In J each literal may be marked as a *decision literal* by placing a dot on top like in $\dot{x}$; if a literal x is not marked, then it is a *propagation literal.* Let J be a sequence of literals of length n. We say that literal $x \in J$ iff there is a $k \in \{1, \ldots, n\}$ such that $x = x_k$. Let $J_1 = (x_1, \ldots, x_n)$ and $J_2 = (y_1, \ldots, y_m)$ be two sequences of literals; their concatenation $J_1 J_2$ is the sequence $(x_1, \ldots, x_n, y_1, \ldots, y_m)$. If a finite sequence J of literals does not contain a complementary pair of literals, then J represents an interpretation. As this condition is always met in this paper, we identify sequences of literals with interpretations whenever appropriate.

The *reduct of a formula F w.r.t. an interpretation J*, in symbols $F|_J$, is defined as $F|_J := \{C|_J \mid C \in F$ and for every literal $x \in C$ we find that $x \notin J\}$, where $C|_J = C \setminus \{\overline{x} \mid x \in J\}$. Intuitively, the reduct operator expresses the state of a SAT solver, where the formula F is the *working formula* and J is the *working assignment.* For instance, let $F = \{\{\overline{x}, y\}, \{z\}\}$, then $F|_x = \{\{y\}, \{z\}\}$, $F|_{\overline{z}} = \{\{\overline{x}, y\}, \bot\}$ and $F|_{y\,z} = \top$, where the interpretations are written as sequences of literals. One should observe that the reduct operator does not distinguish between propagation and decision literals.

Lemma 1 below summarizes the properties of the reduct operator: (i) The reduct is monotone. (ii) A formula F is satisfiable iff there exists an interpretation J such that the reduct of a formula w.r.t. J is the empty formula. (iii) If the reduct of a formula F w.r.t. the interpretation $\{x\}$ is unsatisfiable, and the formula F entails the literal x, then the formula F is unsatisfiable. (iv) The reduct operator is a *semantic operator* in the sense that it cannot distinguish equivalent formulas.

Lemma 1 (Reduct Operator). *Let F, F' be formulas and x a literal.*
(i) $F|_J \subseteq (F \cup F')|_J$ *for every interpretation* J.
(ii) F *is satisfiable iff there exists a* J *such that* $F|_J = \top$.
(iii) *If* $F|_x$ *is unsatisfiable and* $F \models x$*, then* F *is unsatisfiable.*
(iv) *If* $F \equiv F'$*, then* $F|_J \equiv F'|_J$ *for every interpretation* J.

Proof. See [19, pp.10–12]. □

3 Generic CDCL

Modern SAT solvers are based on the linearized `DPLL` [5] algorithm and consists of the following components: *termination criteria*, a *decision component* that picks the branching literals, an *inference component* that adds propagation literals to the working sequence of literals, a *backtracking component* that rolls back wrong decisions and a *formula management component* that simplifies the working formula. We maintain two data structures during the execution of modern SAT solvers: the *working formula*, and the *working set of literals.* Together they define the *state.* The components are modelled as a transition relation over the set of states; the union of the rules in Fig. 1 is then the transition relation

SAT-rule:	$F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{SAT}} \mathsf{SAT}$ iff $F\|_J = \top$.
UNSAT-rule:	$F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{UNSAT}} \mathsf{UNSAT}$ iff $\bot \in F\|_J$ and J contains only propagation literals.
DEC-rule:	$F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{DEC}} F \mathbin{/\!/} J\,\dot{x}$ iff $x \in \mathsf{vars}(F) \cup \overline{\mathsf{vars}(F)}$ and $\{x, \overline{x}\} \cap J = \emptyset$.
INF-rule:	$F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{INF}} F \mathbin{/\!/} J\,x$ iff $F\|_J \equiv_{\mathsf{sat}} F\|_{J\,x}$ and $\{x, \overline{x}\} \cap J = \emptyset$.
LEARN-rule:	$F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{LEARN}} F \cup \{C\} \mathbin{/\!/} J$ iff $F \models C$.
FORGET-rule:	$F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{FORGET}} F \setminus \{C\} \mathbin{/\!/} J$ iff $F \setminus \{C\} \models C$.
BACK-rule:	$F \mathbin{/\!/} J\,J' \rightsquigarrow_{\mathsf{BACK}} F \mathbin{/\!/} J$.
INP-rule:	$F \mathbin{/\!/} \varepsilon \rightsquigarrow_{\mathsf{INP}} F' \mathbin{/\!/} \varepsilon$ iff $F \equiv_{\mathsf{sat}} F'$.

Fig. 1: Transition relations of Generic CDCL. These relations apply to all formulas F and F', clauses C, literals x and lists of literals J and J'. ε denotes the empty sequence of literals.

of Generic CDCL: Formally, we model the computation of modern SAT solvers by means of state transition systems as follows:

Definition 2 (Generic CDCL). Generic CDCL *is a state transition system whose sets of states is*

$$\{F \mathbin{/\!/} J \mid F \textit{ is a formula and } J \textit{ is a sequence of literals}\} \cup \{\mathsf{SAT}, \mathsf{UNSAT}\},$$

whose initial state for the input formula F *is* $\mathsf{init}(F) = F \mathbin{/\!/} \varepsilon$*, whose set of terminal states is* $\{\mathsf{SAT}, \mathsf{UNSAT}\}$*, and whose transition relation* $\rightsquigarrow$ *is defined as:*

$$\rightsquigarrow \; := \{\rightsquigarrow_{\mathsf{SAT}}, \rightsquigarrow_{\mathsf{UNSAT}}, \rightsquigarrow_{\mathsf{DEC}}, \rightsquigarrow_{\mathsf{INF}}, \rightsquigarrow_{\mathsf{LEARN}}, \rightsquigarrow_{\mathsf{FORGET}}, \rightsquigarrow_{\mathsf{BACK}}, \rightsquigarrow_{\mathsf{INP}}\}.$$

The SAT*-rule* terminates the computation with the output SAT, if the reduct of the working formula w.r.t. the working set of literals is the empty formula. This condition can be decided in linear time w.r.t. the size of the working formula F. By Lemma 1 (ii) the working formula is then satisfiable.

The UNSAT*-rule* terminates the computation with the output UNSAT, if no model of the working formula exists. This is the case when a conflict occurs in the top level, i.e. $\bot \in F|_J$ and the sequence J of literals contains only propagation literals. These conditions can be decided in polynomial time.

The DEC*-rule* extends the working sequence of literals by an unassigned literal $\dot{x}$ as a decision literal.

The INF*-rule* extends the working list of literals by a propagation literal x, if the reducts of the working formula w.r.t. the working sequence of literals and its extension are equisatisfiable.

The BACK*-rule* models backtracking, as well as backjumping and restarts, by deleting outermost right literals in the working sequence of literals.

The LEARN*-rule* adds a clause C to the working formula, if it is entailed by the working formula F. Deciding whether $F \models C$ holds, is coNP-complete. Similarly to the INF-rule, SAT solvers avoid this check by using techniques for creating the clause C that ensure this property, as for example resolution.

The FORGET*-rule* deletes a clause C of the working formula F, if $F \setminus \{C\} \models C$. The question whether $F \setminus \{C\} \models C$ holds is coNP-complete. Typically, we use tractable algorithms to identify redundant clauses. For instance, clauses that were introduced by the LEARN-rule but have turned out to be useless and did not participate in the elimination of other clauses in the formula can be removed. For more details on the deletion of clauses see [12].

The INP*-rule* models formula simplifications that are used in pre- and inprocessing. It replaces the working formula with an equisatisfiable formula when the working sequence of literals is empty.

Let $\overset{*}{\rightsquigarrow}$ be the reflexive and transitive closure of $\rightsquigarrow$. Let $x \overset{0}{\rightsquigarrow} x$ for all states x, and $x \overset{n}{\rightsquigarrow} z$ for all natural numbers $n \in \mathbb{N}$ if and only if there exists a state y such that $x \overset{n-1}{\rightsquigarrow} y \rightsquigarrow z$. In the next subsection we investigate the question whether Generic CDCL correctly solves the SAT problem. Formally, we define Generic CDCL to be *sound* iff for all formulas F_0 we have that $\mathsf{init}(F_0) \overset{*}{\rightsquigarrow} \mathsf{SAT}$ implies that F_0 is satisfiable and $\mathsf{init}(F_0) \overset{*}{\rightsquigarrow} \mathsf{UNSAT}$ implies that F_0 is unsatisfiable.

3.1 Generic CDCL is Sound

Before proceeding to the soundness proof of Generic CDCL, it will be necessary to study two invariants of Generic CDCL that are presented in the proposition below: (i) states that the rules of Generic CDCL do not change the satisfiability of the working formula, and (ii) states that whenever the working sequence of literals is of the form $J_1\, x\, J_2$, where x is a propagation literal, the reducts of the working formula w.r.t. J_1 and $J_1\, x$ are equisatisfiable.

Proposition 3 (Invariants). *Let F_0, F be formulas, J be a sequence of literals, and $n \in \mathbb{N}$. If $\mathsf{init}(F_0) \overset{n}{\rightsquigarrow} F \mathbin{/\!/} J$, then*

1. *$F_0 \equiv_{\mathsf{sat}} F$, and*
2. *$F|_{J_1} \equiv_{\mathsf{sat}} F|_{J_1\, x}$, for all sequences of literals J_1, J_2 and propagation literals x with $J = J_1\, x\, J_2$.*

Proof. The claims are proven by induction on n. For the base case $n = 0$, *1.* follows from $F_0 = F$ and *2.* holds since the J is empty. For the induction step, assume that the claim holds for the state $F \mathbin{/\!/} J$ and suppose that $F \mathbin{/\!/} J \rightsquigarrow_{\mathsf{R}} F' \mathbin{/\!/} J'$ where $\mathsf{R} \in \{\mathsf{DEC}, \mathsf{INF}, \mathsf{LEARN}, \mathsf{FORGET}, \mathsf{BACK}, \mathsf{INP}\}$:

- DEC-rule: In this case, $F' = F$ and $J' = J\,\dot{x}$ for some decision literal $\dot{x}$ with $\{x, \overline{x}\} \cap J = \emptyset$. *1.* follows since $F_0 \equiv_{\mathsf{sat}} F$ holds by induction. *2.* holds because the appended literal is a decision literal. Formally, let J'_1, J'_2 be literal sequences, y be a propagation literal such that $J' = J'_1\, y\, J'_2 \dot{x}$. By induction, we conclude that $F|_{J'_1} \equiv_{\mathsf{sat}} F|_{J'_1\, y}$. Hence, $F'|_{J'_1} \equiv_{\mathsf{sat}} F'|_{J_1\, y}$.
- BACK-rule: In this case, $F' = F$ and $J = J'\, J''$. *1.* follows since $F_0 \equiv_{\mathsf{sat}} F$ by induction. *2.* holds because the literal sequence J' is a prefix of J. Formally, let J'_1, J'_2 be literal sequences and y be a propagation literal such that $J' = J'_1\, y\, J'_2$. By induction, we conclude that $F|_{J'_1} \equiv_{\mathsf{sat}} F|_{J'_1\, y}$, and consequently we know that $F'|_{J'_1} \equiv_{\mathsf{sat}} F'|_{J'_1\, y}$.
- LEARN-rule: In this case, $F' = F \cup \{C\}$ where $F \models C$ and $J' = J$. *1.* follows since $F \equiv F'$ and the addition of the entailed clause C preserves equivalence of the working formula. *2.* follows from the reduct operator being a semantic operator by Lemma 1.iv and therefore $F'|_{J'_1} \equiv_{\mathsf{sat}} F'|_{J'_1\, y}$ holds by induction for every literal sequences J'_1, J'_2 and propagation literals y with $J' = J'_1\, y\, J'_2$.
- FORGET-rule: This case can be treated as in the LEARN-rule.
- INF-rule: In this case, $F' = F$ and $J' = J\,x$ for some propagation literal x with $\{x, \overline{x}\} \cap J = \emptyset$. *1.* follows since $F_0 \equiv_{\mathsf{sat}} F$ holds by induction. *2.* follows from the definition of the INF-rule: Consider the literal sequences J'_1, J'_2 and a propagation literal y such that $J' = J'_1\, y\, J'_2$. In the case that $y = x$, we know that J'_2 is the empty sequence of literals and consequently $F|_{J'_1} \equiv_{\mathsf{sat}} F|_{J'_1\, y}$ holds by the definition of the INF-rule. In the case of $y \neq x$, we can conclude the claim by induction.
- INP-rule: In this case, $F' \equiv_{\mathsf{sat}} F$ and J' is the empty sequence. Consequently, *1.* holds by the definition of the INP-rule, and *2.* is satisfied as $J'' = \varepsilon$. □

We can now show the main theorem in this paper.

Theorem 4 (Soundness). *Generic CDCL is sound.*

Proof. We divide the proof in two parts, first proving that the output SAT is correct, and then proving that the output UNSAT is correct. Let F_0, F be formulas, J be a sequence of literals and suppose that

$$\mathsf{init}(F_0) \overset{*}{\rightsquigarrow} F \,/\!\!/\, J \rightsquigarrow \mathsf{SAT}(\mathsf{UNSAT}, \text{resp.}).$$

SAT By the definition of the SAT-rule, we know that $F|_J = \top$. By Lemma 1.ii, we know that the formula F is satisfiable. From the result that the formula F is satisfiable together with the property that the formulas F_0 and F are equisatisfiable given in Prop. 3(*1.*), we conclude that the input formula F_0 is satisfiable.

UNSAT By the definition of the UNSAT-rule, we know that $\bot \in F|_J$ and the working sequence of literals $J = (x_1 \ldots x_n)$ contains only propagation literals. Since a conflict occurs, $F|_J$ is unsatisfiable. From the result that the formula $F|_J$ is unsatisfiable and the literal sequence J contains only propagation literals we can repeatably apply Prop. 3(*2.*) and we obtain that the formula F is unsatisfiable. Since the formula F is unsatisfiable and the formulas F and F_0 are equisatisfiable by Prop. 3(*1.*), we conclude that F_0 is unsatisfiable. □

3.2 Generic CDCL Subsumes Important SAT Solving Techniques

We now describe some important SAT solving techniques, and demonstrate that Generic CDCL can adequately model these techniques.

Subsumption For a formula F, the clause $C \in F$ *subsumes* the clause $D \in F$ iff $C \subset D$. In this case, D can be deleted because $F \setminus \{D\} \models D$. Consequently, $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{FORGET}} F \setminus \{D\} \mathbin{/\!\!/} J$ holds for every literal sequence J. Removing subsumed clauses is done as a preprocessing step in SAT solvers and during clause learning.

Tautologies A clause C is a *tautology* if it contains a complementary pair of literals. Every formula F entails a tautology and the LEARN-rule in Generic CDCL subsumes this techniques. Tautologies are eliminated during preprocessing.

Conflict-Directed Backtracking and Learning [21] is an improvement of naive backtracking that takes the reason of the conflict into account. Consider the state $F \mathbin{/\!\!/} J \dot{x} J'$ and a clause $C \in F$ where $C|_{J \dot{x} J'} = \bot$. The clause C is the *conflict clause*. If there is a linear resolution derivation from the conflict clause C to a clause D in the formula F such that $D|_J$ is the unit clause y, the technique rewrites the state $F \mathbin{/\!\!/} J \dot{x} J'$ into the state $F \cup \{C\} \mathbin{/\!\!/} J y$. Conflict-directed backtracking and learning can be simulated by the following transition steps: $F \mathbin{/\!\!/} J \dot{x} J' \rightsquigarrow_{\mathsf{BACK}} F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{LEARN}} F \cup \{D\} \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{INF}} F \cup \{D\} \mathbin{/\!\!/} J y$.

Blocked Clause Elimination [11] A clause C is *blocked* in the formula F if it contains a literal x such that all resolvents of the clause C and clauses $D \in F$ upon x are tautological. Blocked clauses are removed from a formula during pre- and inprocessing. If C is blocked in F, then $F \equiv_{\mathsf{sat}} F \setminus \{C\}$ and, therefore, the INP-rule subsumes the blocked clause elimination technique.

Unit Propagation A clause that contains a single literal is a *unit clause*. Unit propagation adds the propagation literal x to the literal sequence J, whenever the reduct of the working formula w.r.t. J contains the unit clause (x). Since $F|_J \models x$, we know that $F|_J \equiv_{\mathsf{sat}} F|_{J\,x}$ and consequently the INF-rule subsumes unit resolution.

Pure Literal A literal x is *pure* in the formula F, if $\overline{x} \notin \mathsf{lits}(F)$. For pure literals, it holds that $F \equiv_{\mathsf{sat}} F|_x$ and, therefore, whenever a literal x is pure in the formula $F|_J$ for some literal sequence J, Generic CDCL can add the pure literal to the working literal sequence: $F \mathbin{/\!\!/} J \rightsquigarrow_{\mathsf{INF}} F \mathbin{/\!\!/} J x$.

4 Related Work

Several attempts have been made to formalize sequential SAT solvers in terms of transition systems or proofs calculi: Abstract DPLL [18], Linearized DPLL [1], and Rule-based SAT solver description [15]. In general, these formalization are based on a notion of state like Generic CDCL.

However, these formalizations cannot adequately model recent SAT solvers: For instance, Linearized DPLL does not model the SAT solver `MiniSAT`, because

Linearized DPLL restricts decision literals to occur in the working formula, but the solver `MiniSAT` can also select the complements of such literals. Additionally, Linearized DPLL does not model formula simplification techniques such as blocked clause elimination, or probing-based inference techniques. Similarly, Abstract DPLL and the Rule-based SAT solver description [15] do not model formula simplifications that changes the semantics of formulas like blocked clause elimination. Maric highlighted the implementation of clause learning techniques in his Rule-based SAT solver description [15], but it does not include recent developments such as *clause strengthening.* All these formalizations consider DPLL-based SAT solvers, but the ancient *pure literal rule* is not subsumed in these systems. In contrast, Generic CDCL subsumes all recent SAT techniques to the best of our knowledge.

In [12] Jarvisalo et. al. developed a formal system to model clause learning, forgetting and formula simplification techniques to understand the side-effects of the combination of different rules. They drew our attention to the interplay of the learned clause database with inprocessing techniques. The interplay of clause sharing and formula simplification techniques in *parallel* SAT solvers was analyzed in [14], where the state of a sequential SAT solver was modelled just as the working formula. We believe that Generic CDCL is an important fragment to understand sequential SAT solvers with inprocessing and their cooperation in the parallel-portfolio setting with clause sharing.

5 Conclusion

The propositional satisfiability problem is of great practical interest and can be efficiently answered by modern SAT solvers like `Riss`, `Lingeling` or `MiniSAT`. Today, modern SAT solvers are highly tuned proof procedures with many advaned techniques. Therefore, it is desireable to investigate SAT solving techniques in combination with each other and to abstract from implementations.

In this paper, we developed Generic CDCL, a formalism that models the computation of modern SAT solvers in terms of a state transition system, where each transition rule abstracts a component in a SAT solver. In particular, the transition rules INF and INP model formula simplification techniques like blocked clause elimination and inference techniques such as the pure literal rule. We have examined invariants in Generic CDCL and have shown that Generic CDCL is sound. In contrast to previous work on formalizations of SAT solvers, we can model all recent techniques. The findings add to our understanding of the interplay of inprocessing techniques with the other components of SAT solvers.

A limitation of Generic CDCL is its lack of details in the learning and inference component. As future work, we plan to investigate properties such as *completeness*, *confluence* and *termination.*

References

1. Arnold, H.: A linearized DPLL calculus with clause learning. Tech. rep., Universität Potsdam. Institut für Informatik (2009)

2. Audemard, G., Simon, L.: Predicting learnt clauses quality in modern SAT solvers. In: Boutilier, C. (ed.) IJCAI 2009. pp. 399–404. Morgan Kaufmann Publishers Inc., Pasadena (2009)
3. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press, Amsterdam (2009)
4. Cook, S.A.: The complexity of theorem-proving procedures. In: Harrison, M.A., Banerji, R.B., Ullman, J.D. (eds.) STOC 1991. pp. 151–158. ACM (1971)
5. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (1962)
6. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
7. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) ICLP 1988. pp. 1070–1080. MIT Press (1988)
8. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. Autom. Reason. 24(1–2), 67–100 (2000)
9. Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., Steinke, P.: Solving periodic event scheduling problems with SAT. In: Jiang, H., Ding, W., Ali, M., Wu, X. (eds.) IEA / AIE 2012. LNCS, vol. 7345, pp. 166–175. Springer, Heidelberg (2012)
10. Hölldobler, S., Manthey, N., Saptawijaya, A.: Improving resource-unaware SAT solvers. In: Fermüller, C.G., Voronkov, A. (eds.) LPAR 2010. LNCS, vol. 6397, pp. 519–534. Springer, Heidelberg (2010)
11. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 129–144. Springer, Heidelberg (2010)
12. Järvisalo, M., Heule, M.J.H., Biere, A.: Inprocessing rules. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 355–370. Springer, Heidelberg (2012)
13. Lynce, I., Marques-Silva, J.: Efficient haplotype inference with Boolean satisfiability. In: AAAI 2006. pp. 104–109. AAAI Press (2006)
14. Manthey, N., Philipp, T., Wernhard, C.: Soundness of inprocessing in clause sharing SAT solvers. In: Järvisalo, M., Van Gelder, A. (eds.) SAT 2013. LNCS, vol. 7962, pp. 22–39. Springer, Heidelberg (2013)
15. Marić, F.: Formalization and implementation of modern SAT solvers. J. Autom. Reason. 43(1), 81–119 (2009)
16. Marques Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers 48(5), 506–521 (1999)
17. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: DAC 2001. pp. 530–535. ACM, New York (2001)
18. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Abstract DPLL and abstract DPLL modulo theories. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS, vol. 3452, pp. 36–50. Springer, Heidelberg (2005)
19. Philipp, T.: Expressive Models for Parallel Satisfiability Solvers. Master thesis, Technische Universität Dresden (2013)
20. Rossi, F., Beek, P.v., Walsh, T.: Handbook of Constraint Programming. Elsevier Science Inc., New York (2006)
21. Silva, J.P.M., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: ICCAD 1996. pp. 220–227. IEEE Computer Society, Washington (1996)

Dynamical Change of the Perceiving Properties of Neural Networks as Training with Noise and Its Impact on Pattern Recognition

Roman Nemkov

Department of Information Systems & Technologies, North-Caucasus Federal University. 2, Kulakov Prospect, Stavropol, Russian Federation

nemkov.roman@yandex.ru

Abstract. General parameters of convolutional networks (kernels) are set in the learning process. Also in addition to the method of training the quantity of information that is passed through the kernel influences the quality of setting. This quantity of information depends on the size of training sample and the concentration of receptive fields. You can increase the concentration of the re-ceptive fields for a fixed training set size due to the multilayer coating of arbitrary maps with fields of different types, that will be equivalent to the use of noisy training sample. This can increase the networks performance in the test.

Keywords: convolutional networks, training with noise, different types of receptive fields.

1 Introduction

To date, problem of invariance is the main and yet unsolved problem of pattern recognition: the same object may have substantially different from each other external characteristic (shape, colour, texture, etc.) as well as it may be differently displayed on the retina (view from different angles), which greatly complicates its classification. This global problem within the neural network technologies can be solved by to create big training sets [1]. If creating such sets are difficult then this sets are expanded by the addition of noise. [4–6]. A neural network can be regarded as a pyramidal hierarchical graph then noise can be created by changing communications between nodes in a graph [2, 3] or changing perceiving properties in nodes of a graph [8]. Convolutional neural networks (CNNs) have three perceiving properties in a node-neuron: a receptive field (RF), an activation function and a method for producing a weighted sum (simple weighted sum or higher-order polynomial). A change of RFs is the easiest and most promising way of the three. The same pattern can be differently perceived by changing the RFs, that leads to the creation of noise. The influence of noise (which was created due to changing the shape of the fields) is rinvestigated on the pattern recognition in this article.

2 The Generation of Noise by Changing the Receptive Fields

Training with noise in the context of gradient descent for a neural network can be written as

$$\frac{\partial E}{\partial w} + \varepsilon = \nabla, \tag{1}$$

where $\frac{\partial E}{\partial w}$ is the gradient vector from the network's weights, ε is the additional noisy component corrects the gradient vector. The gradient vector after a correction can't exactly point to the local minimum, but training with noise has two benefits: generalization ability increases and local minimums can be better overcome during the gradient descent. When you change perceiving properties in the nodes of networks you have the same training with noise, where ε can be explicitly written:

$$\varepsilon = \frac{\partial E}{\partial w}(new\_perception) - \frac{\partial E}{\partial w}(standart\_perception), \tag{2}$$

where$\frac{\partial E}{\partial w}(new_perception)$ is the gradient vector from weights (the perceiving properties have been already changed), $\frac{\partial E}{\partial w}(standart\_perception)$ is the gradient vector from weights with the standart perceiving properties (RFs have square form).

The changing of perceiving properties in the nodes of a network occur due to changing the shape of the RFs. Each element of RF has neighbors. The neighbors are elements located in the one or two discrete steps from current element. Thus, the value of current element (within RF) can be replaced by the neighboring value. If you do this operation for all elements of RF then weighted sum will be changed, hence output of neuron will be also changed. The replacement is shown on the Fig. 1.

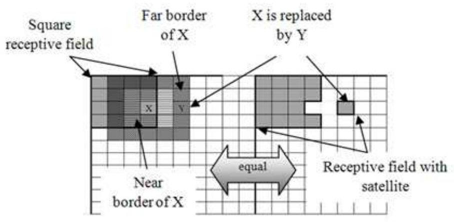


Fig. 1. The replacement of neuron-pixel X by Y with the help of changing the shape of the receptive field (the receptive field with the satellite).

The map (which is the input for the current neurons) receives another covering of RFs, but the kernels of convolution layer (which passed through themselves this new covering) remain the same. The quantity of information affecting the kernels increases and the kernels can extract the best invariant features. This process is shown in Fig. 2.

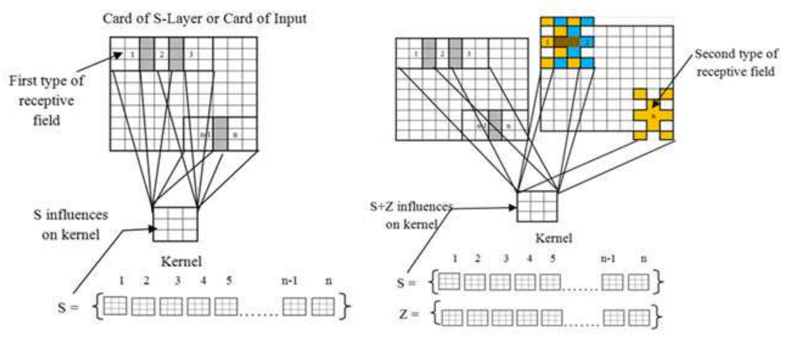

Card of S-Layer or Card of Input
First type of receptive field
S influences on kernel
Kernel
S =
Second type of receptive field
S+Z influences on kernel
Kernel
S =
Z =

Fig. 2. Discrete perception of information with the help of convolution layer (C-Layer) (Left). The same perception, but with the help of C-Layer with different receptive fields. The pattern will be perceived by the first type of receptive field in the first stroke. In the second stroke the same pattern will be perceived by the second type of receptive field (Right).

This technology can expand the training set by the patterns which are created depending on where (how far away from the current element) elements of RFs take the information for a replacement. Let all elements of RFs are replaced then additional training sets, which will be obtained by this technology, is shown in Fig. 3.

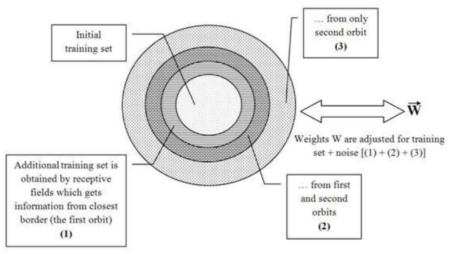

Initial training set
... from only second orbit (3)
W
Weights W are adjusted for training set + noise [(1) + (2) + (3)]
Additional training set is obtained by receptive fields which gets information from closest border (the first orbit) (1)
... from first and second orbits (2)

Fig. 3. Additional sets which will be obtained by the changing of RFs.

Any convolutional layers may have their coverings, hence the change of perception can be on the different layers. If CNN has three convolutional layers then the quantity of combinations (or “refracting prisms”) will be 2^3-1=7 (one “prism” is a standard perception). The unique pattern will be obtained within the frames of each scheme-“prism”. A strategy is also important for a marking the particular layer using RFs. There are two opposing strategies: the RF is chosen by random way and is superimposed on desired location or the same type of RF with a specific index is superimposed on all desired locations. The second strategy can model the primitive affine transformations if the RF simulates a shift for all its elements.

Patterns need create with the combinations of all “refracting prisms”, with using the both strategies, with using the RFs which are fully updated for maximum coverage of any of the three additional sets.

3 Experiment

MNIST was chosen for experiments with noise. This is due to the fact, that the most schemes of the creating of noise have been tested in this set. The architecture of CNN is shown in Fig. 4.

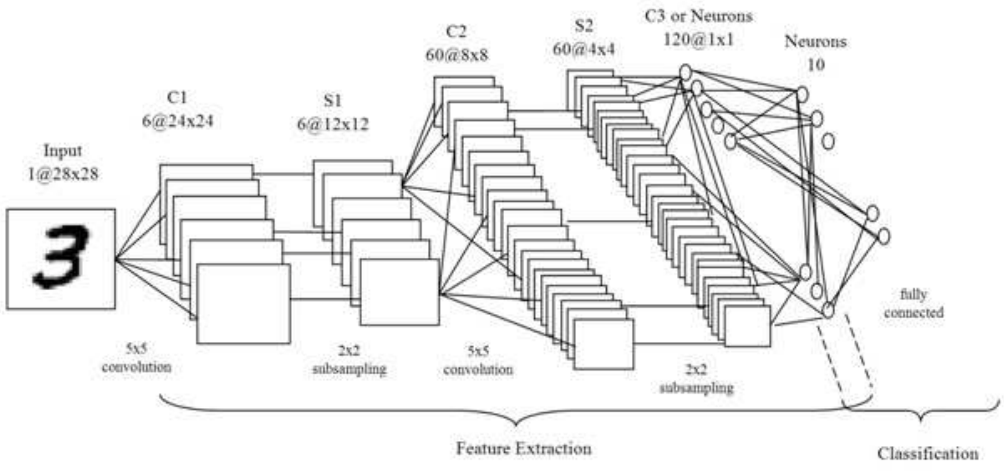


Fig. 4. The architecture of CNN for the work with MNIST set.

The simplest algorithm of gradient descent (without momentum, weights decay and other tricks) has been used for maximum simplicity and repeatability of the experiment. The initial value of the learning rate (η) is equal to 0.005, after every 100 epochs new value are obtained from the old value by multiplying by 0.3. Error function is mean-square error (MSE). The pattern is recognized if the error on the output layer does not exceed the value of 0.001. Pools of RFs for convolutional layers are shown in Figure 5.

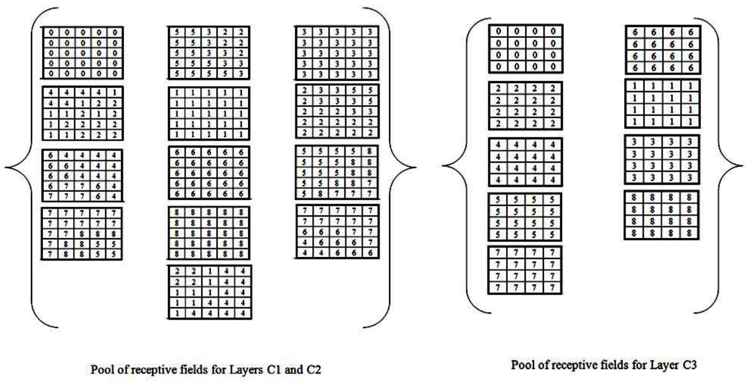

Pool of receptive fields for Layers C1 and C2
Pool of receptive fields for Layer C3

Fig. 5. Pools of RFs for convolutional layers.

The RF with arbitrary index is set in the proper position by the strategy of markup. Geometrical interpretation of the index for shift is shown in Fig. 6.

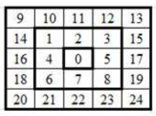


Fig. 6. The geometric interpretation of index for the shift of element of RF.

Thus, the noise from the first additional training set (Fig. 3, set (1)) was used. Comparative results are given in Table 1.

Table 1. The comparison between different learning algorithms

Algorithm	Distortion	Error	Ref.
2 layer MLP (MSE)	affine	1.6%	[4]
SVM	affine	1.4%	[5]
Tangent dist.	affine+thick	1.1%	[4]
LeNet-5 (MSE)	affine	0.8%	[4]
2 layer MLP (MSE)	elastic	0.9%	[6]
CNN (MSE)	this distortions	1.2%	this paper
Best result	elastic	0.23%	[7]

This is a good result that has been achieved without the involvement of additional noise from the sets (2) or (3) (Fig. 3). Research has shown that the change of perceiving properties in the nodes of CNN can effectively expand the training set and reduce the error of generalization. Also, this technology is easy compatible with the elastic distortions and dropconnects or dropouts.

References

[1] Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M.A., Senior, A., Tucker, P., Yang, K., Ng, A. Y.: Large Scale Distributed Deep Networks. NIPS, 2012.

[2] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Improving Neural Networks by Preventing Co-adaptation of Feature Detectors. CoRR, 2012.

[3] Wan, L., Zeiler, M. D., Zhang, S., LeCun, Y., Fergus, R.: Regularization of Neural Networks using DropConnect. ICML 3, volume 28 of JMLR Proceedings, page 1058-1062.

[4] LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based Learning Applied to Document Recognition. Proceedings of the IEEE. v. 86, pp. 2278-2324, 1998.

[5] Decoste, D., Scholkopf, B.: Training Invariant Support Vector Machines. Machine Learning Journal. vol 46, No 1-3, 2002.

[6] Simard, P., Steinkraus, D., Platt, J.C.: Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. ICDAR, page 958-962. IEEE Computer Society, 2003.

[7] Ciresan, D., Meier, U., Schmidhuber, J.: Multicolumn Deep Neural Networks for Image Classification. In Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), CVPR '12, pages 3642-3649, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-1-4673-1226-4.

[8] Nemkov, R., Mezentseva, O.: The Use of Convolutional Neural Networks with Non-specific Receptive Fields. The 4-th International Scientific Conference " Applied Natural Science 2013", Novy Smokovec, High Tatras, Slovak Republic, Oktober 2-4, 2013, p.148.

A Pipeline for Supervised Formal Definition Generation

Alina Petrova

Technische Universität Dresden
alina.v.petrova@gmail.com

Abstract. Ontologies play a major role in life sciences, enabling a number of applications. Obtaining formalized knowledge from unstructured data is especially relevant for biomedical domain, since the amount of textual biomedical data has been growing exponentially. The aim of this paper is to develop a method of creating formal definitions for biomedical concepts using textual information from scientific literature (PubMed abstracts), encyclopedias (Wikipedia), controlled vocabularies (MeSH) and the Web. The knowledge representation formalism of choice is Description Logic as it allows for integrating the newly acquired axioms in existing biomedical ontologies (e.g. SNOMED) as well as for automated reasoning on top of them. The work is specifically focused on extracting non-taxonomic relations and their instances from natural language texts. It encompasses the analysis, description, implementation and evaluation of the supervised relation extraction pipeline.

Keywords: knowledge representation, non-taxonomic relationships, ontology learning

1 Introduction

Formalization of biomedical knowledge has long been an area of active research. Existing biomedical knowledge resources vary considerably in terms of their formalization principles, from databases and data collections (e.g. MEDLINE[1]), to taxonomies and controlled vocabularies (e.g. MeSH[2]), to proper ontologies with rich formal semantics (e.g. SNOMED[3]). They also vary greatly with respect to the domains and areas they cover, size, age, ways of maintaining and integrating new knowledge etc. Formally representing the biomedical knowledge can bridge the gap between existing resources and enrich them as well as process the newly generated knowledge that comes in abundance and is publicly accessible.

Research in life sciences is characterized by the exponential growth of the published scientific materials: articles, patents, technical reports etc. MEDLINE, one of the biggest bibliographic databases for biomedicine, currently contains

[1] http://www.ncbi.nlm.nih.gov/pubmed
[2] http://www.nlm.nih.gov/mesh/
[3] http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html

more than 23 million articles. The average amount of newly added articles comprises 15000 items per week. To handle such an amount of information, multiple initiatives have been launched for the purpose of organizing biomedical knowledge formally, e.g. using ontologies [3]. An ontology is a complex formal structure that can be decomposed into a set of logical axioms that state different relations between formal concepts. Together the axioms model the state of affairs in a domain. With the advances in Description Logics (DL) the process of designing, implementing and maintaining large-scale ontologies has been considerably facilitated [1]. In fact, DL has become the most widely used formalism underlying ontologies. Several well-known biomedical ontologies, such as GALEN or SNOMED CT [4] are based on DL. SNOMED CT has adopted the lightweight description logic *EL++* that allows for tractable reasoning.

There are several benefits of formal knowledge representation. First of all, it enables efficient information integration; already existing knowledge about the entity can be aggregated from multiple resources, and the new knowledge can be easily integrated. Secondly, formal knowledge representation makes it possible to automatize a number of crucial tasks that deal with information processing: efficient search, validation and reasoning. Finally, formal representation can support knowledge visualization which itself can bring about further insights about the domain, i.e., facilitate knowledge discovery.

1.1 Two Examples of Biomedical Knowledge Formalization

In this section we present two recent works in which the application of formal ontologies to biomedical knowledge produced interesting results that demonstrate the usefulness and the potential of knowledge formalization in the biomedical domain. In [4] the authors used the Foundational Model of Anatomy (FMA), an ontology of human anatomy, where concepts are linked with the *part-of* relation. They annotated images of penetrating injuries with anatomic concepts, thus disambiguating the visible regions of the body, and then performed logical reasoning over the ontology to predict the possible internal damages caused by the injury. The project was conducted by the U.S. Defense Advanced Research Projects agency and has life-saving importance. [5] ran an even more large-scale and ambitious project. Its aim was to create a robot scientist a robot that conducts independent research, that is, sets a hypothesis, tests it experimentally and reasons about the acquired data by interpreting the results, all on its own. The developed robot had a rich knowledge base on the backbone that was used at all stages of the research process. The robot was provided with a general biomedical database as well as with a formal model of yeast metabolism, and it autonomously generated and validated experimentally functional genomics hypotheses, thus becoming the first machine that made a scientific discovery without human intervention. The two works illustrate the huge range of applications that formal knowledge resources can have in life sciences as well as their unbounded potential. Not only do they help sustain the ever-growing collection of already published results, but they can also lead to knowledge discovery through formal reasoning.

1.2 What is Formal Definition Generation?

Formal definition generation (FDG) is a type of knowledge modeling that translates a natural language definition into a formal representation using some formal language notation. FDG can be viewed as the automatic acquisition of complex axioms for an ontology. Unlike the taxonomy acquisition, which seeks to identify parent-child relations in text and is usually based on simple patterns [6], FDG focuses on highly expressive axioms containing various logical connectives and non-taxonomic relation instances.

Formal definition generation can be illustrated by an example: a natural language sentence with a classic definitional structure *A is a type of B that has a specific property C* is translated into a formal representation $A \equiv B \sqcap \exists hasProperty.C$. The formalism of choice here is Description Logic (DL).

Some definitions can be straightforwardly rewritten into a formal language:

Acenocoumarol: a coumarin that is used as an anticoagulant.

It is a definition taken from the MeSH controlled vocabulary. If we assume that *Acenocoumarol*, *Coumarin* and *Anticoagulant* are valid biomedical concepts, the definition can be encoded by means of a simple DL in the following way:

$Acenocoumarol \equiv Coumarin \sqcap \exists used\_As.Anticoagulant$

The encoding is very simple since there exists an almost perfect one-to-one correspondence between the lexical items in the definition and the elements of the formal syntax. However, this is not the case for the majority of the sentences. FDG does not boil down to a mere re-writing of textual definitions using a different notation; instead, it is a complex task that requires thorough analysis and understanding of utterances and their constituents. Below are the examples of MeSH definitions that are far more difficult to process:

> *Acetolactate Synthase*: *a flavoprotein enzyme that catalyzes the formation of acetolactate from 2 moles of pyruvate in the biosynthesis of valine and the formation of acetohydroxybutyrate from pyruvate and alpha-ketobutyrate in the biosynthesis of isoleucine.*
>
> *Lissamine Green Dyes*: *green dyes containing ammonium and aryl sulfonate moieties that facilitate the visualization of tissues, if given intravenously.*

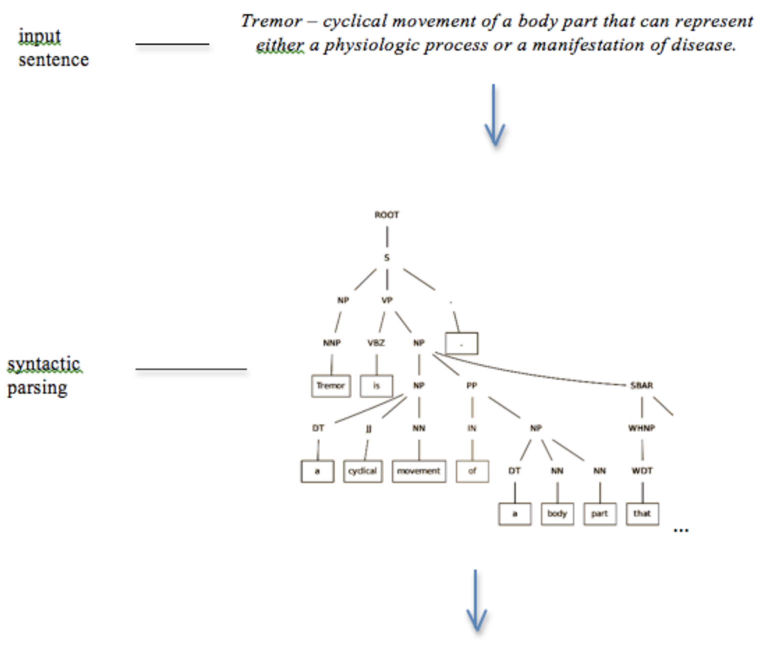
Even definitions for which finding a formal representation appears to be trivial may in fact contain various pitfalls. How exactly should the definition *Acepromazin is a phenothiazine that is used in the treatment of psychoses* be formalized? Should *the treatment* correspond to an independent concept that is linked to emphAcepromazin by the *used_in* relation? Or should it rather correspond to the relation *treats* that takes as arguments *psychosis* and *phenothiazine*, and ultimately *acepromazin*? The answer to this question is not obvious and is heavily dependent on the way one chooses to model the knowledge.

2 Methodology

This section describes the pipeline for formal definition generation (FDG) from text. The core step in FDG is non-taxonomic relation extraction: not only expressive relation instances account for the most part of definition formulas, but they also require such tasks as concept annotation and taxonomy detection as preprocessing steps. Hence, the FDG pipeline in essence tackles the task of relation extraction. It consists of the following main steps, illustrated by Figure 1 and discussed in the subsequent sections:

– syntactic parsing of the input sentence;
– semantic annotation of the sentence with biomedical concepts;
– extraction of semantic triples from syntactic paths between the annotated concepts;
– classification of the triples as pertaining to specific biomedical relations;
– final formula generation based on the labeled triples.

Each step is discussed in detail in the subsequent sections. Figure 1 illustrates the pipeline using the MeSH definition of *Tremor*:



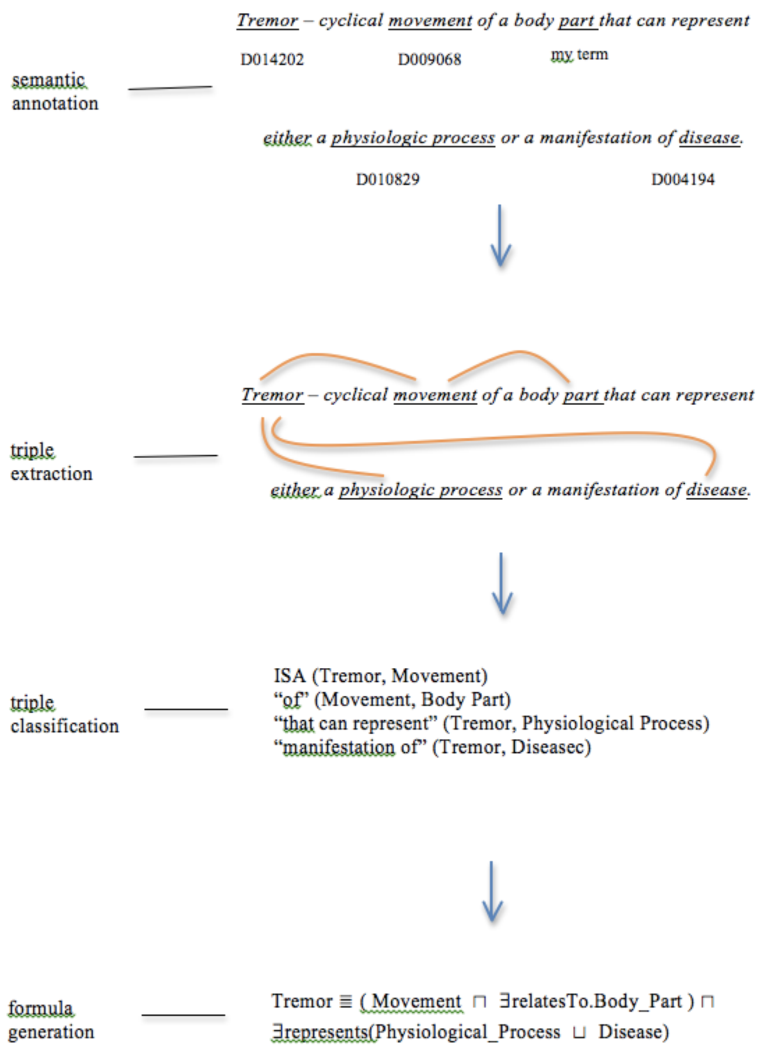

Tremor – cyclical movement of a body part that can represent
D014202
D009068
my term
semantic annotation
either a physiologic process or a manifestation of disease.
D010829
D004194
Tremor – cyclical movement of a body part that can represent
triple extraction
either a physiologic process or a manifestation of disease.
ISA (Tremor, Movement)
"of" (Movement, Body Part)
"that can represent" (Tremor, Physiological Process)
"manifestation of" (Tremor, Diseasec)
triple classification
formula generation
Tremor ≡ (Movement ⊓ ∃relatesTo.Body_Part) ⊓ ∃represents(Physiological_Process ⊔ Disease)

Fig. 1. Formal definition generation pipeline. The definition for *Tremor* is used as an example.

2.1 Parsing and Annotation of Input Sentences

Parsing and annotation can be viewed as two pre-processing steps of the pipeline. They both rely on separate components, syntactic parsers and semantic annotators respectively, enriching the initial textual input with additional information, i.e. the syntactic dependencies and the occurrences of concept mentions in the sentence. In the present work we use external parser and annotator, since their creation is itself a stand-alone research problem which lie outside the scope of formal definition generation problem.

Given an input text and an ontology that describes the domain, concept annotation, also called semantic indexing or concept recognition, is the task of finding in text mentions of ontology concepts and mapping the corresponding lexical tokens to concepts. Typically, biomedical concept annotators aim at recognizing textual occurrences of diseases, drugs, genes, body parts, species and in principle, any other conceptual entity that exists in the input ontology.

There are multiple third-party biomedical annotators available online. As a rule, they use specific knowledge resources, i.e. ontologies and thesauri, as repositories of concepts they aim to find in texts. One of the most widely used one is MetaMap [7]. It is a dictionary-based system that indexes biomedical text with UMLS concepts [8]. We use MetaMap as the annotator of choice.

2.2 Triple Extraction

After a text string is annotated with biomedical concepts, the next step is to group these concepts into relational instances and to form a preliminary structure of the formal definition. It is the task of the triple extraction component. The parser takes as input a textual definition pre-annotated with biomedical concepts as well as its syntactic parse tree and produces structures of the form *concept_A – relational_string – concept_B*, which we call *unlabeled triples.*
The triple extraction component runs as follows:

1) **detect the parent term**, if it is present
At this step we rely on the information provided by the ontology used for the semantic annotation: if the term that appears first in the definition is not recognized by the annotator, i.e. it is not considered as concept by the ontology, then it belongs to a relational string of some triple (*Abdominal Wall* example); otherwise it is a parent concept (*Cattle Diseases* example).

Abdominal Wall: the outer margins of the abdomen, extending from the osteocartilaginous thoracic cage to the pelvis.
Cattle Diseases: diseases of domestic cattle of the genus bos.

2) **group coordinated concepts into conjunctive or disjunctive sets**
Detecting coordination is one of the very important issues in predication extraction. Coordinated concepts are organized into sets with one representative concept. Whenever this concept is part of a triple, the rest of the concepts automatically form triples as well, using the same relational string and the same concept as the second argument, e.g.:

Vesicular stomatitis Indiana virus: the type species of vesiculovirus causing a disease symptomatically similar to foot-and-mouth disease in cattle, horses, and pigs.

> *Foot-and-Mouth Disease — in — Cattle*
> *Foot-and-Mouth Disease — in — Horses*
> *Foot-and-Mouth Disease — in — Swine*

Triples for coordinated concepts will further be transformed into conjunction and disjunction of concepts in DL notation of the definition formula.

3) **organize the concepts into concept pairs**
This is the key step in the definition parsing as it shapes the resulting triples. The process is heavily dependent on the syntactic structure of the sentence. One straightforward way of linking concepts together would be to follow the dependency paths across the syntactic tree and to link every concept with the nearest dominating one (and to link the top concept with the head concept). However, while parsing the definition, we would like to collect as much information about the head term as possible. For this reason we link annotated concepts with the head term whenever it is possible and does not violate the common sense. In fact, for the majority of the triples both ways of constructing triples are possible and comprehensive. For example, in the following definition:

> *Classical Lissencephalies: disorders comprising a spectrum of brain malformations representing the paradigm of a diffuse neuronal migration disorder*,

if *Classical Lissencephalies* is a malformation that represents *Diffuse neuronal migration disorder*, we can induce that it represents a disorder. Thus, by linking concepts occurring in the definition with the main term we skip this induction process.

4) **extract relational strings**
To finish the formation of unlabeled triples, we need to accompany concept pairs with relational strings that contain the mention of the respective relation in text. The intuitive approach is to take the strings that are located in between the two concepts in the pair. It has two major disadvantages, though: the in-between string might either contain more than just the relation mention and thus cause noise during classification, or it might as well not contain the mention altegether. To avoid such mistakes, we extract only the substring between the current concept and the preceding one, independently of the position of the second concept:

> *Hypothalamic Hormones: peptide hormones produced by neurons of various regions in the hypothalamus.*
> *Peptide hormones — produced by — Neurons*
> *Peptide hormones — of various regions in — Hypothalamus.*

5) **detect negation**
The negation is detected in the relation string using simple patterns. If the string

contains tokens like not or other than, the triple is considered to be negated. This information is useful and should be propagated till the step of formula generation. After the parsing is completed, the parser outputs unlabeled triples of the form *concept_A — relational_string — concept_B*, possibly accompanied with the NEGATION mark. The number of triples for a definition depends directly on the number of annotated concepts.

2.3 Triple Classification

The last step of the formal definition generation pipeline takes as input the unlabeled triples generated by the parser and substitutes the relational strings with the relation labels reducing the relation instances to the invariants of some domain-specific relation.

Labeling the text strings with relation names is an instance of the text classification task. Relational instances, or triples, represent the training/testing examples and form the learning corpus. Every instance is represented as a set of features and passed to a machine learning algorithm. The learning instances are labeled with a class — a relation name in our case. The model is then trained using the labeled instances and is used for the classification of new instances that do not yet have a label. We assume that a relational instance corresponds to a precisely one relation, thus the task of relation labeling is a single-label classification.

Thus, the most inportant step of triple classification is to train such a model that will perform accurately on the new input definitions. The model cannot be trained on the textual instances per se, but rather on their formal representation as sets of features. In this work we extracted two types of features from the relational triples: lexical and semantic features.

Lexical features correspond to relational string of the triple. We used the so-called character ngrams as lexical features. Given a textual instance I and a value for the parameter n, we now examine I as an ordered series of characters instead of words. For the extraction of the character ngrams, we are using a sliding window of size n, and we do not exclude space characters in order to capture patterns across token boundaries. For example, for a string *is pneumoconiosis caused by* the character bi-grams are: *pn,ne,eu,um,mo,oc,co,on,ni,io,os,si, c,ca,au,us,se,ed,d , b,by*. Each instance I (in our case the text between the two concepts) can be represented as a feature vector, features being ngrams and the value of each feature being 0 or 1, depending on whether a term occurs in the instance (1) or not (0).

Ngrams are able to implicitly capture a huge variety of information about a string. In particular, character ngrams can reflect word order, lemmas, stems and grammatical forms of words, important morphemes, to name a few. All this information is utilized at a cheap cost: no sophisticated linguistic analysis is required for the ngram extraction. Obviously, a single ngram does not play a big role in the labeling process, but several ngrams of the same string taken together can yield a strong signal of a particular class. For example, a set of trigrams {*cau,aus,use,sed, ed, ed ,d b, by*} extracted from a relational string *caused by* captures not only the stem form of the verb, but also the fact that it is used in passive voice and is followed by a preposition by, which reflects a very common lexical pattern for the causative relation.

Unlike lexical features, **semantic features** account for the argument concepts of the triple. As semantic features, we used concept types of the arguments, i.e. broad semantic categories of concepts. In order to determine the types of triple concepts, one needs an underlying terminology where all the concepts are combined into an ontology. Every concept can be reduced to some upper concept of broad semantics, which will be considered as a semantic type and used as a features. In our work we used the UMLS Semantic Network as the source of types. The UMLS Semantic Network [9] is an upper ontology for the biomedical domain which forms the top level of the UMLS concept hierarchy. It has 133 semantic types and 54 semantic relations. Types and relations are very broad and are used for high-level categorization and interlinking of concepts. Types are assigned to all the concepts of the UMLS, thus the type information is easily accessible.

Thus, every relational instance, i.e. triple, is represented as a set of all character tri-grams extracted from the relational string, and a pair of concept types fo the relational arguments. A model then is trained on all processed instances and is used for the classification of unseen instances.

2.4 Formula Generation

When all the triples are extracted from the input textual definition, the last step is to combine them into a formula. In the current version of the pipeline we follow the formalization of biomedical knowledge used in SNOMED CT ontology and quantify all relational instances existentially. All triples are then combined conjunctively.

3 Evaluation and Discussion

As it has been stated at the beginning of the paper, relation extraction is at the core of formal definition generation. Therefore, in this section we discuss the performance of our pipeline with respect to relation extraction. In particular, we separately evaluate the steps of triple extraction and triple classification.

3.1 Triple Extraction

As a preliminary evaluation of the parser, we have run it over a small corpus of 40 definitions and manually annotated the generated triples as correct or incorrect. MeSH serves the source for textual definitions to be parsed. The triples were evaluated as follows: we mark a triple as correct if the two concepts serving as arguments of the relation are chosen correctly and the relational string is also parsed correctly (it does not miss anything). If any of the two conditions was violated, the triple was considered incorrect. For 40 randomly selected MeSH definitions annotated with 147 concepts from MeSH and from the extended vocabulary the parser generated 110 triples. 98 triples are manually labeled as correct, only 11 triples (10%) are incorrect. In particular, for 32 definitions out of 40 the triples are generated correctly (80%).

3.2 Triple Classification

For the evaluation of relation classification process given the set of features we designed (see section 2.3), we relied on the external corpus. An external corpus is needed to exclude the errors passed from the previous steps of FDG pipeline which would affect the classification performance. We used *SemRep Gold Standard* corpus, which consists of 500 MEDLINE sentences manually annotated with relational triples. The annotation includes concepts, concept types, relational strings and relation labels. The corpus contains 1357 instances of 26 distinct relations. The the top occurring relations are *process_of, location_of, part_of, affects, treats.* The corpus was used for training and testing of the SVM classifier using 10-fold cross-validation. The tests were run for top 5, top 10 and for all 26 relations. The resulting F-measure is 94%, 89.1%, 82.7%, respectively. It should be noted, that the top 5 relations account for 63% of all relational instances, which means that with our learning method we can classify the majority of relational instances with an expremely high F-measure of over 90%.

3.3 Related Approaches

One previous approach of Formal Definition Generation is described in [10]. The authors reformulate the task into an automatic acquisition of ontology axioms from natural language texts. The formalism of choice is *SHOIN*, a very expressive DL that is able to model negation, conjunction, disjunction, and quantitative restrictions. The developed system *LExO* is based on full syntactic parsing of input sentences. The dependency tree is transformed into DL formulas through a chain of hand-written syntactic rules that take into account parts of speech, sentence positions, tree positions and syntactic roles of all words. The rules cover a broad set of syntactic structures, such as relative clauses, prepositional, noun and verbal phrases.

Another related approach [11] belongs to the area of ontology acquisition. Ontologies consist of terminological axioms (TBox) and assertional facts (ABox). In this paper, we focus on acquiring a special but common TBox knowledge — formal definitions — from texts. [11] mainly studies the ABox extraction, whereas the enlisted TBox acquisition approaches are mainly based on syntax transformation.

The mentioned systems have several limitations in formalizing the definitional sentences, which stem from their rule-based nature. Two main problems are semantically ambiguous relation mentions, e.g., *of*, and relation mentions with similar semantics, but dissimilar lingistic form, e.g., *Causative_agent* relation in SNOMED CT can be expressed both by *caused_by* and *due_to.* Natural language is versatile and complicated, and the same meaning can be expressed in multiple ways. Hence, it is not possible to cover all ways of expression of a relation by hand-crafted rules. In our work we attempt to solve this issue by applying machine learning techniques to learn the models of axioms, thus avoiding hand-crafted patterns on the lexicon or the syntactic structure of a sentence and instead implicitly learning probable language relation expressions.

4 Conclusion

In this work we addressed a novel problem of generating formal definitions from textual descriptions of biomedical concepts. Formal definition generation is a complex task. We approached it from a text mining perspective and split it into several consecutive steps. We were particularly focused on the non-taxonomic relation extraction as expressive relations contain the core information about the concepts to be defined. We implemented and evaluated relation extraction and relation classification steps, integrating state-of-the-art domain resources and external tools, i.e. semantic annotators, achieving high-performance and and setting the scene for further research of the FDG problem. Formal definition generation pipeline can be used as a standalone tool for concept formalization, and it can also be integrated into ontology learning tools as a semi-automatic tool for the assistance to domain experts.

References

1. F. Baader, I. Horrocks and U. Sattler. *Description logics.* In Handbook on Ontologies, 2004.
2. G. Tsatsaronis, A. Petrova, M. Kissa, Y. Ma, F. Distel, F. Baader, M. Schroeder. *Learning Formal Definitions for Biomedical Concepts.* In OWLED, 2013.
3. O. Bodenreider. *The Unified Medical Language System (UMLS): integrating biomedical terminology.* Nucleic Acids Research, 32(Database issue): D267-D270, Jan 2004.
4. D. L. Rubin, O. Dameron, Y. Bashir, D. Grossman, P. Dev, and M. A. Musen. *Using ontologies linked with geometric models to reason about penetrating injuries.* Artificial Intelligence in Medicine, 37(3):167-176, 2006.
5. R. D. King, J. J. Rowland, W. Aubrey, M. Liakata, M. Markham, L. N. Soldatova, K. E. Whelan, A. Clare, M. Young, A. Sparkes, S. G. Oliver, and P. Pir. *The robot scientist Adam.* IEEE Computer, 42(8):46-54, 2009.
6. T. Wächter. 2010. *Semi-automated Ontology Generation for Biocuration and Semantic Search.* PhD thesis. Technische Universität Dresden, Germany.
7. A. R. Aronson. *MetaMap: Mapping Text to the UMLS Metathesaurus.* Bethesda, MD: NLM, NIH, DHHS (2006).
8. Unified Medical Language System, `http://www.nlm.nih.gov/research/umls/`
9. S. Schulze-Kremer, B. Smith, and A. Kumar. *Revising the UMLS semantic network.* Medinfo (2004): 1700-4.
10. J. Volker, P. Hitzler and P. Cimiano. *Acquisition of OWL DL axioms from lexical resoures.* In ESWC, pages 670-685, 2007.
11. P. Cimiano. *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications.* Springer-Verlag New York, Inc., 2006.
12. A. Petrova. *Learning Formal Definitions for Biomedical Concepts.* Master thesis. Technische Universität Dresden, Germany.

Properties of Majority Transformations under Random Processes Parameters Measurement

Elena Stepanova[⋆] and Alexey Liagin and Alla Pletukhina

North-Caucasus Federal University zik@ncstu.ru

Abstract. Statistical inference problem arises when you want to give the best, in some sense, the answers to a limited number of observations. When it comes to problems of statistical inference, it is assumed that it is possible to obtain a random sample, is set consisting of the realizations of independent and identically distributed random variables. The article attempts to assess quality parameters of a random process normally distributed through the application of order statistics (particularly the selected median) with a limited sample size.

In practice it is often required to determine the quantitative attribute (variables) against noise background. Lets assume that its initially known what kind of distribution a sign exactly possesses. There comes the task of assessing parameters that determine this distribution. If its known that the attribute under examination is normally distributed in the entire assembly, which is typical for mixture of signal and Gaussian noise at noise-to-signal ratio greater than unity, it is necessary to evaluate (calculate approximately) mathematical expectation and root-mean-square deviation, as these two parameters completely determine normal distribution.

Usually at a researchers disposal there is only sample data, e.g. quantitative attribute values $x_1, x_2, ..., x_n$ received as a result of N observations (observations assumed to be independent). Test variable is expressed in terms of this data.

When considering $x_1, x_2, ..., x_n$ as independent random variables $X_1, X_2, ..., X_n$, we can state that to find a statistical evaluation of the unknown parameter theoretical distribution means to find a function of the observed random variables, which gives an approximate value of the parameter estimated.

Most frequently in practice, for equally accurate measurements, parameters of random process distribution are evaluated by general average, which is reasonable for a large number of values of random variables observed. However for a small number of measurements, with high degree of unequal accuracy, as is known from mathematical statistics, sample median estimate is more effective than sample average.

Let physical process be described as a function of time $X(t)$. On the signal parameter tester signal $X = X(t) + n(t)$ is applied where $X(t)$ - measuring signal, $n(t)$ - external influence (noise). Signal parameters are evaluated at a certain time interval $\Delta(t)$.

[⋆] the presenter, email: cherry-sweet@yandex.ru

In mathematical statistics we know integrated (averaged) evaluation methods $x(\mathit{\Delta}t) = \lim\limits_{n\to\infty} \sum\limits_{i=i}^{n} \frac{x_i}{n}$, where $i = 1...n$ - equal for the majority of equally accurate measurements from the entire assembly $X(t) \subset f(x_1, x_2, ..., x_n)$. However, these techniques make evaluation of the random process parameters consistent, unbiased and effective only under controlled (predictable) changes of the parameters of random process.

When changes of random process parameters are unpredictable nonparametric method for estimation of the process parameters is interesting. Its principle is as follows: in the measurement time interval $\mathit{\Delta}(t)$ several measurements of the random process parameter $x_j = x + x_j$ $(j = 1...k)$ are made. Sensor output signals contain n_j measurement random errors caused by external influences (noise) which properties are characterized by probability densities $(f_j(x))$ Results of measurements (random variables x_j) form set of variate values

$$x_1, x_2, ..., x_k \tag{1}$$

so that $x_1 < x_2 < ... < x_k$. In case of odd number of measurements $(K = 2k+1)$ mean proportional of set of variate values (1.4.2) is taken as valuation of Z parameter of the process measured.

$$Z = X(k+1) \tag{2}$$

For sample size $K = 2k + 1 = 3$ suggested measurement algorithm can be implemented on (max and min) transformations. Then mathematical model and algorithm of measurement can be represented as $Z = med(x_1, x_2, x_3) = \max\{\min(x_1, x_2), \min(x_1, x_3), \min(x_2, x_3\}$

For symmetrically distributed measurement errors estimate (2) is unbiased. Obviously, the error of estimate (2) is equal to the errors median n_j

$$e = Z - x = n_(k+1) \tag{3}$$

For practice, a case is important when the distribution of measurement errors on the observation interval is different, the differences characteristics are not known beforehand and cannot be used for optimal algorithms measurements apply. This situation occurs while the rapid changes in the measured parameter of physical process on the observation interval, for example, rapid changes in the amplitude of the analog signal, the frequency alteration and phase of harmonic oscillations - the information carriers in a rapidly changing interference intensity on the observation interval and a number of other situations.

Lets compare by statistical efficiency and estimation accuracy of the physical parameter process on sample median with a simple averaging for the case of unequal dimensions.

For practice, a case is interesting when the sample size for the measurement interval K = 3, the measurement errors are normally distributed with zero mean and variances $\sigma_1^2, \sigma_2^2, \sigma_3^2$ and it is unknown, what specific measurement corresponds to a certain level of error.

According to [1], the probability density estimation errors algorithm sample median:

$$f_2(x) = \sum_{i=1}^{3} f_i(x)\{F_j(x)[1 - F_b(x)] + F_b(x)[1 - F_j(x)]\}$$
$$(i, j, e = 1, 2, 3; i \neq j \neq e) \qquad (4)$$

At a normal distribution inaccuracy, the formula looks like this:

$$F_i(x) = 0,5\left[1 + erf\left(\frac{x}{\sqrt{2}\sigma_i}\right)\right], where\ erf(y) = \frac{2}{\sqrt{\pi}}\int\limits_0^y e^{-t^2}dt.$$

Error variance σ_2^2 for the sample median, the corresponding distribution (5) turns out to be [1]

$$\sigma_{(2)}^2 = \sigma_1^2\left\{\frac{\lambda_1^2+\lambda_2^2}{2} - \frac{1}{\pi}\left[\arctan\frac{1}{A} + \frac{A^2+\lambda_1^2\lambda_2^2}{A(1+A^2)}\right] - \frac{\lambda_1^2}{\pi}\left[\arctan\frac{\lambda_1^2}{A} + \right.\right.$$
$$\left.\left. + \frac{\lambda_1^2(A^2+\lambda_2^2)}{A(A^2+\lambda_1^4)}\right] - \frac{\lambda_2^2}{\pi}\left[\arctan\frac{\lambda_2^2}{A} + \frac{\lambda_2^2(A+\lambda_1^2)}{A(A^2+\lambda_2^4)}\right]\right\} \qquad (5)$$

From (6) follows that if the error variance of any two measurements are limited and the errors of the third dimension are infinitely large variance, and the measurement results practically unreliable, error variance is found to be $\sigma_2^2 = \frac{1}{2}(\sigma_1^2 + \sigma_2^2)$.

This means that the assessment on sample median virtually eliminated false data.

The other situation is observed at an average measurement results. The calculation is defined as:

$$\sigma^2 = \frac{1}{9}(\sigma_1^2 + \sigma_2^2 + \sigma_3^2) \qquad (6)$$

That is: an unlimited increase of dispersion errors in one of the measurements leads to unlimited increase of error variance estimates.

At symmetric distribution laws $F_i(x)$ of errors and when they do not contain systematic components, the probability $P(\Delta)$ in the case of unequal probability measurement is defined as:

$$P(\Delta) = 1 - 2\left[\int\limits_0^{\Delta}\{f_1(x)(F_2(x)[1 - F_3(x)] + F_3(x)[1 - F_2(x)]) + f_2(x)(F_1(x)[1 - \right.$$
$$\left. -F_3(x)] + F_3(x)[1 - F_1(x)]) + F_3(x)(F_1(x)[1 - F_2(x)] + F_2(x)[1 - F_1(x)])\}dx\right]$$

As a result of integration, we obtain the following:

$$P(\Delta) = F_1(\Delta)F_2(\Delta) + F_1(\Delta)F_3(\Delta) + F_2(\Delta)F_3(\Delta) - 2F_1(\Delta)F_2(\Delta)F_3(\Delta) - \frac{1}{2} \qquad (7)$$

In case of a normal error distribution at $\sigma = \sigma_2 = \sigma_3$ and $\sigma_3 \neq \sigma$ probability $P(\Delta)$ is:

$$P(\Delta) = 1 - \frac{1}{2}\Phi\left(\frac{\Delta}{\sqrt{2}\sigma\delta}\right)\left[1 - \Phi^2\left(\frac{\Delta}{\sqrt{2}\sigma}\right)\right] - \Phi\left(\frac{\Delta}{\sqrt{2}\sigma}\right), \qquad (8)$$

where $\delta = \frac{\sigma_3}{\sigma}$.

when unequal measurements, the sample median has the best performance in terms of quality considered criterion than the sample mean an order of magnitude or more. This demonstrates the feasibility of applying the algorithm sample median for measuring the parameters of stochastic processes on the background noise and the impact of external influences, both on a physical process, and the measuring device. Practical confirmation the latest are researches for example, in [2,3,4].

References

[1] Signal processing based on an ordered choice. Guilbaud E. P., Chelpanov I.B. - Moscow: Soviet Radio, 1975.

[2] Properties of order statistics in the measurement of non-energy parameters of harmonic oscillations. Lyagin A.M., Pletuhina A.A., Pletuhina E.P. Topical issues and innovation in the economy, governance, education, information technology. Proceedings of the international conference. Issue. 6. T1. Stavropol.

[3] Majoritarian transformation algorithm for signal processing RPM at low sample sizes. Lyagin A.M., Pletuhina A.A., Pletuhina E.P. Stavropol, Kislovodsk, 2012. Fifth International Scientific Conference "Information and Communication Technologies in science, business and education" (Infocom-5). Collection of scientific papers. Part 1.

[4] Device distinguish discrete phase-modulated signals based on an ordered choice. Lyagin A.M., Pletuhina A.A., Pletuhina E.P., Mashchenko A.A. Herald, North - Caucasus University, the scientific journal, 2013, 1 (34), pages 39 - 42

Identification of Exploitation Conditions of the Automobile Tire while Car Driving by Means of Hidden Markov Models

Denis Tananaev, Galina Shagrova, Victor Kozhevnikov

North-Caucasus Federal University, Stavropol, Russia

d.d.tananaev@gmail.com,g_shagrova@mail.ru,
viktor_kozhevnikov@inbox.ru

Abstract. This article describes the implementation of the Hidden Markov Models for identification of exploitation conditions of the automobile tire by means of analyzing tire noise while car driving. This requires the development of special recognition algorithms of tire noise and cleaning of the signal from the background noise, it can be done by means of extraction of the clean signal from the noise by adaptive filters and by pattern recognition methods, typically used in speech recognition, to recognize a tire noise corresponding to a particular operating condition. In this way, we can diagnose the condition of a tire while car driving, which will reduce overloaded tire wear, due to improper use to a minimum and help prevent accidents as a result of tire failure.

Keywords: Hidden Markov models, Adaptive filters, tire noise, pattern recognition, feature extraction

1 Introduction

The problem of the road transport accidents, caused by the failure of automobile tire, is one of the most important ones for traffic safety. A key reason for the failure of automobile tire is its increased wear as the result of improper use. It may be caused by many factors: the collapse of the incorrect angles of convergence, high or low tire pressure, overheating, etc. It is impossible to control all the factors, influencing the dynamics of tires while driving, and, therefore, there is a need for a comprehensive new indicator. We think that this indicator is the sound of tires. There is a lot of research of tire dynamics in the field of automobile safety. In general, models of tire/road noise can be divided into four major types. The first type includes statistical models. A popular example of this approach is introduced in the article by Sandberg, U. and Descornet, G. [1]. The second type is composed of physical models. The examples of such a modeling approach are analysed in the book by Kropp, W. [2]. The third type of models for tire/road noise is hybrid theoretical models. The examples of hybrid theoretical models are described by De Roo, F., Gerretsen, E. and Hamet, J.F., Klein, P. [3, 4]. Finally, statistical models can be extended with pre or post processing, based on well-known physical relations, often derived from theoretical models. The examples of hybrid statistical models are introduced by Beckenbauer, T. and Kuijpers A. [5]. We think the disadvantage of these models is that they only describe the noise generation mechanisms of the tire, independently of the condition of the tire. In contrast, we attempt to model dependencies between tire sounds and tire conditions, based on the hypothesis that the operational status of the tire is reflected in its noise characteristics. We must develop dedicated recognition algorithms of tire noise and also algorithms of clearing up the signal of the background noise. It can be done by means of extraction of the clean signal from the noise by adaptive filters and pattern recognition to classify a tire noise as corresponding to a particular operating condition.

2 Data Preparation

2.1 Adaptive Filtering

First, it is necessary to clear the tire signal from the background noise. It can be done by using adaptive filters. In our research we use adaptive filter, based on the least mean square algorithm [6], which is realized in the Matlab Simulink (see Fig.1).
The acoustic signal $x(t)$, which contains the tire signal $s(t)$ and noise $l(t)$ is recorded by the first microphone, which is installed near the tire. The pattern of noise $l_0(t)$ is recorded by the second microphone, which is located near the engine of the automobile. There is a correlation between $l(t)$ and $l_0(t)$. The output of the adaptive filter will contain the measure of the noise $\hat{l}_0(t)$. The error of the filter will contain a clear tire acoustic signal $\hat{s}(t)$. The spectrogram of the clear tire signals which we received as the results of the experiments (the experiments are described in Section 4) is shown in Fig 2.

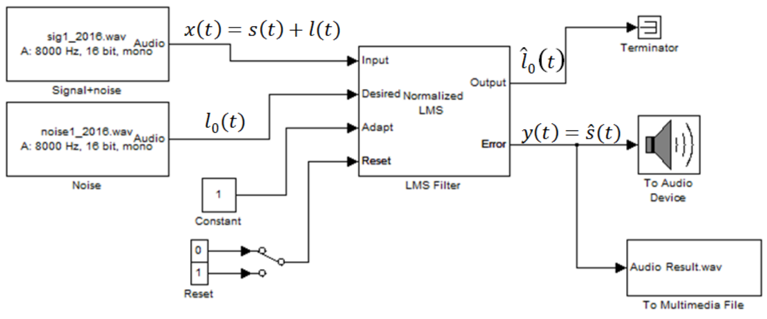


Fig. 1. The scheme of adaptive filter from Matlab Simulink

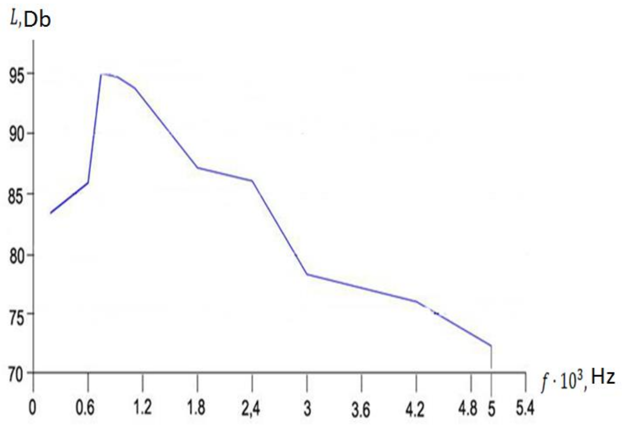


Fig. 2. The tire signals spectrogram

The frequency range of the clean acoustic signals of the tire is between 400-5000 Hz.

2.2 Feature Extraction

The next step is the feature extraction. The purpose of this step is to parameterize the raw tire signal waveforms into sequences of feature vectors. Here we use both FFT-based and LPC-based analysis with the purpose to identify which approach is better for the tire noise coding. The feature techniques are based on the widely known methods MFCC and LPCC [7] which are often used for speech recognition. We process the signal with the frame size 25 msec and frame period 10 msec (Fig.3).

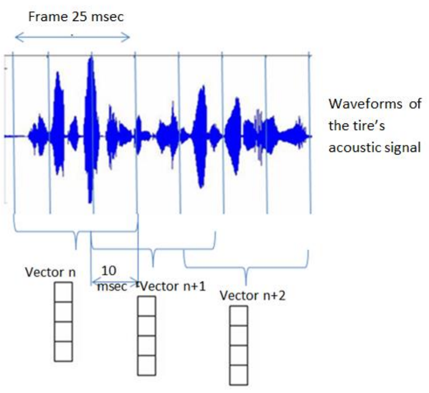


Fig. 3. Framing of the waveforms of the tire acoustic signal

The tire noise feature vectors were parameterized as follows: if the target parameters are MFCC, we use C_0 as the energy component. We use a Hamming window in FFT. The filterbank has 26 channels. In output we receive 12+1 (C_0) coefficients. The performance of the tire noise recognition system can be enhanced by adding time derivatives (delta and acceleration coefficients) to the basic static parameters [7]. If the target parameters are LPCC, we use linear prediction of the 14th order. The filterbank size is 22 channels and in output we receive 12 coefficients. Then we add delta and acceleration. After feature extraction procedure we have 39 dimensional MFCC vectors or if we use the LPCC method - 36 dimensional vector.

3 HMM Training and Recognition

3.1 Topology of the HMM

We use the left-right HMM with seven hidden states (see Fig.4) for identification of the tires exploitation condition. The first and the last states (S_1 and S_7) are not emitted as we need these nodes to create composed HMM (see Fig.5).

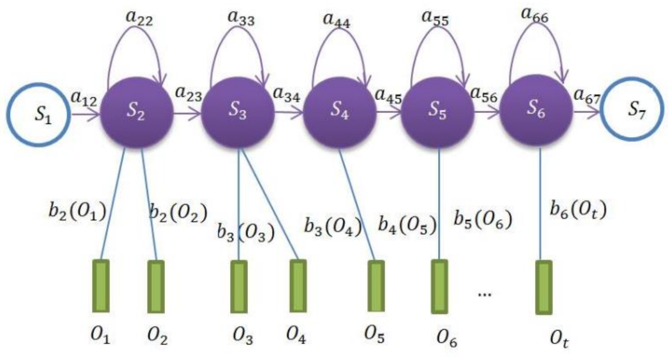


Fig. 4. The left-right HMM for identification of the tire exploitation condition

Here N – number of hidden states of the model (N=7); $A = \{a_{ij}\}$ – the matrix of the transition probabilities:

$$a_{ij} = P\left[q_{t+1} = \frac{S_j}{q_t} = S_i\right], 1 \leq \text{i}, \text{j} \leq \text{N} \tag{1}$$

q_t- hidden state of the HMM ($S_1, \dots, S_7$) at the moment t; j – next state of HMM; i – actual state of HMM; $B = b(O_t)$ – observation probability; $O_1, \dots, O_t$ – feature vectors of the tire noise.

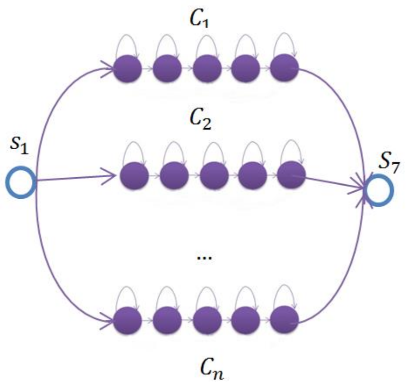


Fig. 5. Composed HMM; $C_{1,\dots,}C_n$ –exploitation conditions of the tire

3.2 HMM Training

For HMM training we use the same method as for speech recognition [7]. We record a training database of the tire noise which relate to every exploitation condition of the tire. It is necessary to make 3-5 recordings of the tire noise

10-15 seconds long for every exploitation condition with the purpose to create the robust recognition system. Then for each exploitation condition of the tire $C_{1,\dots,}C_n$ we initialize one HMM with seven hidden states.
Using maximum likelihood we estimate the matrix of transitions between the states in the hidden part of the model. After that we estimate the mean $\hat{\mu}_j$ and the matrix of covariance $\hat{\sum}_j$ by means of these formulas:

$$\hat{\mu}_j = \frac{1}{T}\sum_{t=1}^{T} O_t \tag{2}$$

$$\hat{\sum}_j = \frac{1}{T}\sum_{t=1}^{T}\left(O_t - \mu_j\right)\left(O_t - \mu_j\right)^T \tag{3}$$

where T – is a number of the feature vectors;
Then we can calculate the observation probability of the feature vectors of the tire noise:

$$\mathrm{b_j}(O_\mathrm{t}) = \frac{1}{\sqrt{(2\pi)^{\mathrm{n}}|\hat{\sum}_j|}}\mathrm{e}^{-\frac{1}{2}(\mathrm{O}-\hat{\mu}_j)'\hat{\sum}_j{}^{-1}(\mathrm{O}-\hat{\mu}_j)} \tag{4}$$

Where n – is a dimensionality of the feature vectors.
It is necessary to estimate corresponding probability for each state, and to use the Viterbi algorithm [7] for reassigning the observation vectors for each state. We re-estimate model parameters in this way until we stop getting their improvements.
The next step is to create $M = 16$ Gaussian mixtures [9]. It is necessary to create a robust system of the tire exploitation condition recognition.
We use the Baum – Welch [8] algorithm to define $\psi_{jm}^{r}(t)$ – the probability of observation vector being in the particular state. Here R is the number of training data $1 \le r \le R$. After that, we re-estimate the parameters of the model. The observation probability $b_j(O_t)$ is:

$$b_j(O_t) = \sum_{\mathrm{m}=1}^{\mathrm{M}} \mathrm{c_{jm}}\mathrm{N}(\mathrm{O_t}; \mu_{\mathrm{jm}}, \sum\nolimits_{\mathrm{jm}}) \tag{5}$$

$$N(O_t; \mu_{jm}, \sum\nolimits_{jm}) = \frac{1}{\sqrt{(2\pi)^n|\sum_{jm}|}} e^{-\frac{1}{2}(O-\mu_{jm})'\sum_{jm}{}^{-1}(O-\mu_{jm})} \tag{6}$$

Re-estimation of the mean and covariance matrix is:

$$\mu_{jm} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{Tr}\psi_{jm}^{r}(t)o_t^r}{\sum_{r=1}^{R}\sum_{t=1}^{Tr}\psi_{jm}^{r}(t)} \tag{7}$$

where T_r – is the number of the observation vectors.

$$\hat{\Sigma}_{jm} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{Tr} \psi_{jm}^{r}(t)(O_{st}^{r} - \hat{\mu}_{jm})(O_{t}^{r} - \hat{\mu}_{jm})^{T}}{\sum_{r=1}^{R} \sum_{t=1}^{Tr} \psi_{jm}^{r}(t)} \tag{8}$$

The weights of the Gaussian mixture components are:

$$c_{jm} = \frac{\sum_{r=1}^{R} \sum_{t=1}^{Tr} \psi_{jm}^{r}(t)}{\sum_{r=1}^{R} \sum_{t=1}^{Tr} \sum_{l=1}^{M_s} \psi_{jm}^{r}(t)} \tag{9}$$

We re-estimate the parameters of the model until $b_j(O_t)$ stop getting improvements of the model parameters.

3.3 Recognition

We use the Viterbi decoding [7] for the tire noise recognition (Fig.6). This algorithm could be used to find the maximum likelihood state sequence of HMM and identify the tire exploitation condition. Let $\delta_t(j)$ represent the maximum likelihood of the observing tire noise vectors O_i to O_t in state j at time t. This likelihood can be computed efficiently using the following recursion:

$$\delta_t(j) = \max_i\{\delta_{t-1}(i) \cdot a_{ij}\} \cdot b_j(O_t) \tag{10}$$

where

$$\delta_1(1) = 1 \tag{11}$$

$$\delta_1(j) = a_{1j} b_j(O_1) \tag{12}$$

The maximum likelihood for observing sequence of vectors O_i to O_t given the HMM model:

$$\delta_T(N) = \max_i\{\delta_T(i) \cdot a_{iN}\} \tag{13}$$

As for the re-estimation case, the direct computation of likelihoods leads to underflow, so it will be better to compute log likelihood:

$$\delta_t(j) = \max_i\{\delta_{t-1}(i) + \log(a_{ij})\} + \log(b_j(O_t) \tag{14}$$

This algorithm can be visualized as searching the best path through a matrix, where the vertical dimension represents the states of the HMM and the horizontal dimension represents the frames of the tire noise.

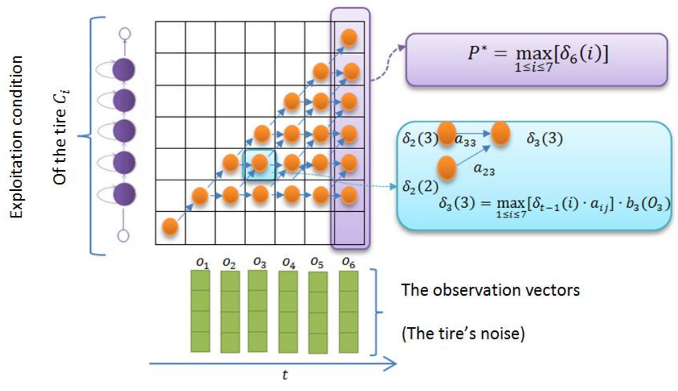

Exploitation condition
Of the tire C_i
$P^* = \max_{1 \le i \le 7}[\delta_6(i)]$
$\delta_2(3)$
a_{33}
$\delta_3(3)$
a_{23}
$\delta_2(2)$
$\delta_3(3) = \max_{1 \le i \le 7}[\delta_{t-1}(i) \cdot a_{ij}] \cdot b_3(O_3)$
o_1 o_2 o_3 o_4 o_5 o_6
The observation vectors
(The tire's noise)
t

Fig. 6. – Scheme of the Viterbi decoding

Each large dot in the picture represents the log probability of observing that frame at that time and each arc between dots corresponds to the log transition probability. The log probability of any path is computed simply by summing the log transition probabilities and the log output probabilities along that path. The paths grow from left-to-right, column-by-column. At time t, each partial path $\delta_{t-1}(i)$ is known for all states i, hence, equation 14 can be used to compute $\delta_t(j)$,thereby, extending the partial paths by one time frame.

4 Experiments and Results

4.1 Experiments

We carried out field tests with the purpose to record the tire noise while car driving with different exploitation conditions of the tire. Our experiment is based on the standards ISO 10844 [10] and ISO 13325:2003 [11], which determine the conditions for the tire noise measurement, but we included the following changes:

- The noise of the tire was measured with the engine working
- Microphones were installed near the front right wheel (Fig.7) with the purpose to provide adaptive filtering of the background noise

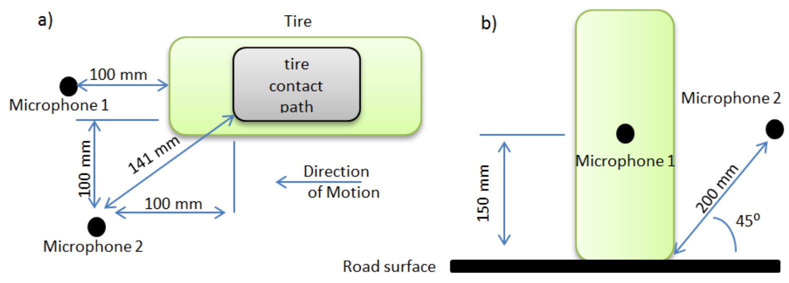


Fig. 7. – Scheme of the microphones' positions, during field tests: a) Upside view b). Front view

We recorded the tire noise with three different speeds of the automobile 20, 40 и 60 km per hour and three different pressure levels: 1.9, 2.1 and 2.3 atmospheres. The automobile used for field tests was Mitsubishi L200 (year of construction: 2011), with new tires 265/75R16.

4.2 Evaluation

We made three different experiments. For each experiment we used 405 records of the tire noise, the total duration of 1 hour 41 minute 15 seconds for HMM training.

Table 1. The experiment results

Features	HMM (1 Gaussian)	HMM (8 Gaussian mixtures)	HMM (16 Gaussian mixtures)
The results of the tire pressure identification			
LPC/LPCEPSTRA	78%	87.5%	88.2%
MFCC	68%	77.4%	78.2%
The results of the automobile's speed identification			
LPC/LPCEPSTRA	81.2%	94.3%	95.7%
MFCC	78.6%	89.4%	91.8%
The results of the identification of the tire speed and pressure			
LPC/LPCEPSTRA	61.4%	74.7%	75%
MFCC	58.6%	59.4%	61.9%

To evaluate the efficiency of the system we used 50 records, a total duration of 12 minutes 30 seconds. As we can see in table 1 the accuracy of our method for the tire pressure is 88,2%; for the automobile speed - 95,7%; and for both the speed and tire pressure - 75%.

5 Conclusions

We have found the correlation between the tire noise and the tire exploitations characteristics. The cleaning mechanism, based on adaptive filters, and the recognition mechanism, based on the HMM have shown prospective results. We found out that the performance of the recognition system depends on exploitation parameters. They show better results for the automobile speed than for the tire pressure identification. Moreover, we have also discovered, that the performance of the recognition system runs low when more than one parameter are identified.

6 References

1. Sandberg, U. and Descornet, G. (1980). Road surface influence on tire/road noise– part I & II. Proceedings of INTERNOISE 1980, Miami, Florida.
2. Kropp, W. (1989). Structure-borne sound on a smooth tyre. Applied Acoustics, 26, 181-192.
3. De Roo, F. and Gerretsen, E. (2000). TRIAS - tyre road interaction acoustic simulation model. Proceedings of INTERNOISE 2000, Nice, Italy.
4. Hamet, J.F. and Klein, P. (2000). Road texture and tire noise. Retrieved April 10, 2006 fromhttp://www.inrets.fr/ur/lte/publications/publications-pdf/web-hamet/in00_674.pdf.
5. Beckenbauer, T. and Kuijpers, A. (2001). Prediction of pass-by levels depending on road surface parameters by means of a hybrid model. Proceedings of INTERNOISE 2001, The Hague, the Netherlands.
6. Deacons, V. (2009). MATLAB 6: 5 SP1/7/7 SP1 / SP2 + Simulink 7 5/6 Tools artificial intelligence and bioinformatics Monograph Imprint: Moscow: SOLON-PRESS, 2009.
7. Steve Young, Gunnar Evermann, Phil Woodland [and others]. The HTK Book. Cambridge University Engineering Department. 2001-2002
8. Rabiner, L. R. A tutorial on Hidden Markov Models and Selected Applications in Speech Recognition / L. R. Rabiner // Proceedings of the IEEE. Vol. 77. February 1989. № 2. P. 257 – 284.
9. McLachlan, G. and Peel, D. Finite Mixture Models. NewYork: Wiley Interscience, 2000.
10. ISO 10844:2011 .Acoustics - Specification of test tracks for measuring noise emitted by road vehicles and their tyres / By International Organization for Standardization (ISO). Geneva: ISO, 2011
11. ISO 13325:2003 Tyres - Coast-by methods for measurement to tyre-to-road sound emission / By International Organization for Standardization (ISO). Geneva: ISO, 2003.

DBMS Index for Hierarchical Data Using Nested Intervals and Residue Classes

Vladimir Volonkin

Institute of Information Technologies and Telecommunications of the North-Caucasus Federal University, Stavropol, Russia
`vl.voronkin@raxperi.com`

Abstract. In the work an index based on B+ tree and oriented to storage of tree which are coded by nested intervals method with usage of system of residual classes is described.

Keywords: nested intervals, residue classes, hierarchical data.

1 Introduction

A hierarchy in relational databases (RDB) in present time is implemented ineffectively. Existed methods have definite drawbacks and intent as usual on either quick data reading truckle to record or vice versa.

There are a few possible ways of an effectiveness increasing of a tree-type structures storage: an improvement of existed methods of a hierarchy storage, a development and a research of new methods or refusal of relational databases using in favour of NoSQL databases: graph, document-centric and other types.

All methods of a hierarchy structures presentation in RDB can be grouped into two main categories:

1. Methods of trees encoding.
2. Hierarchical / recursive SQL extensions.

We should note methods of hierarchy presentations, which are based on combining of several methods, for example: method, combining materialized path and adjacency list [1].

Till recently there were two main methods of hierarchy storage (graph) in group of trees encoding methods: nested sets [2] and materialized path [5].

In reference [1] Vadim Tropashko has suggested a modification of materialized path: a method of nested intervals for storage of tree-type structures. The offered method, based on conception of the materialized path in graphs and the continued fractions theory, eliminates a problem of a data redundancy in the materialized path.

Main problems of the method are following: a necessity of calculation with a great numbers, a significant complexity of an execution of partial operations in particular trees rearrangement during transference of subtrees.

To eliminate problems concerned with nested intervals authors [3] offered to encode nested intervals with using of systems of residual classes (SRC).

Using of SRC makes a number multiprocessing possible, relieves from the necessity to work with great numbers and bignums, but imposes some constraints on its using:

– a possibility to present only restricted amount of numbers;
– an absence of effective algorithms for comparison of numbers in SRC.

For effective presentation of trees in databases it is necessary to solve following problems:

– an elimination of working with great numbers and bignums, which are appearing by using of the nested intervals method;
  The problem solving is posed in [3]. A consequence of this method using is problem of increasing of number storage redundancy. This problem is following from nature of number, which is presented in SRC.
– high complexity of operations execution to rearrange tree;
  The problem can be partly solved by using of residue numbers. Interdependency between residues of numbers, which are expressed in SRC, that allows to execute arithmetics with residues simultaneously. For realization of an opportunity of parallel calculations with number residues of nested intervals expressed in SRC, without taking interdependency into consideration, numbers should be in particular order: a parent node should be described necessarily before a descendant node. An alternation of sibling nodes doesnt matter.
– an absence of indexing methods for numbers expressed in SRC;
  At the moment there are no any indexing methods for numbers expressed in SRC. An indexation of numbers expressed in SRC by using traditional algorithms is unfeasible because numbers expressed in SRC dont seem possible to arrange in series as in a decimal as in a SRC representation. This feature issues from the nature of number expressed in SRC.
– a significant performance penalty at storage of big trees in DB (more than 1 mln records);
  The problem is mainly conditioned by increasing of a key length during derivation of big trees and by a big quantity of calculations needed to an execution of tree derivation operations.

Offered in the work approach to the index derivation and processing allows to solve problems described above.

There is provided a structure of index based on the B+ tree and oriented on index multiprocessing in not relational DB.

2 Structure of the Index

2.1 Encoding of the Tree by the Method of Nested Intervals

The method of nested intervals is an expansion of nested sets model, using of continued fractions.

Lets present a tree:

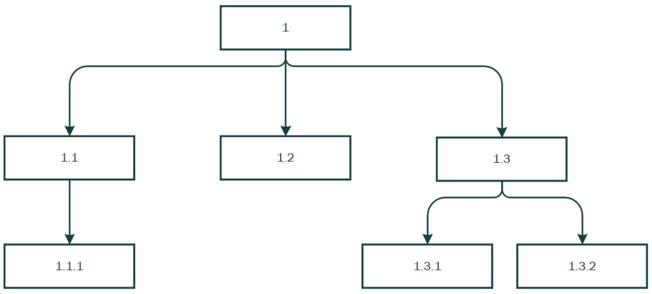


Fig. 1. An examle of tree

Encode the node 1.3.2 using of continued fractions 1:

$$1.3.2 = 1 + \frac{1}{3 + \frac{1}{2 + \frac{1}{x}}} = \frac{9x + 4}{7x + 3}; \tag{1}$$

A result 1 is an nested interval:

$$(\frac{9}{7}, \frac{4}{3}) \tag{2}$$

An interval (2) determines a range, where all descendant nodes will be encoded, and consists of two parts: 9/7 directly a node code, 4/3 the parents node code. For unique identification of the node it is enough to use only a node code, and to calculate the parents node code only if necessary.

2.2 Systems of Residual Classes in the Index

A presentation of number in the SRC is based on a notion of deduction and the Chinese remainder theorem. The SRC is determined of set of coprime modules which are called a basis. In the SRC [4] numbers are represented in following way:

$$A(\alpha_1, \alpha_2, ..., \alpha_n); \tag{3}$$

$$\alpha_i = A - [\frac{A}{p_i}]p_i, (\forall_i \in [1, n]), \tag{4}$$

$$P = \prod_{i=1}^{n} p_i \tag{5}$$

Where $p_1, p_2, ..., p_n$ - system modules, P - volume of system range. Assume $p_1 = 2, p_2 = 3, p_3 = 5, P = 30$.

Consider, how the node code is encoding in the SRC:
Encode a node 1.3.2 by using of formulas (3) and (4):

$$1.3.2 = (\frac{9}{7}, \frac{4}{3}) \tag{6}$$

We will use only the node code:

$$\alpha_{11} = 17 = (\alpha_1, \alpha_2, \alpha_3). \tag{7}$$

$$\alpha_1 = 9 - [\frac{9}{2}] * 2 = 1, \tag{8}$$

$$\alpha_2 = 9 - [\frac{9}{3}] * 3 = 0, \tag{9}$$

$$\alpha_3 = 9 - [\frac{9}{5}] * 5 = 4, \tag{10}$$

$$\alpha_{12} = 7 = (1, 1, 2). \tag{11}$$

It is following from the nature of the SRC that a number expression in the SRC imposes constraints on a number length.

A maximum quantity of numbers, which could be stored in a number expressed in the SRC, is always less than a quantity of numbers, which could be stored in the same quantity of bits as a decimal number.

So a maximum whole unsigned number, which can be stored in 4 bytes, equals:

$$N_{max} = 2^{32} = 4294967296 \tag{12}$$

It is known that modules of the system of residual classes $p_1, p_2, ..., p_n$ should be prime in pairs numbers. In this case there is a single non-negative decision modulo P an equation system which describes residue numbers:

$$x \equiv \alpha_1(mod(p_1)), x \equiv \alpha_2(mod(p_2)), , x \equiv \alpha_k(mod(p_k)) \tag{13}$$

Obviously the more equal P and N_{max}, the less of a redundancy of numbers storage in the SRC. Hence, to find a minimum pressure of numbers record in the SRC it is necessary to try maximally great prime in pairs residue number base.

As an example we will try optimum bases, which are maximally not exessive for number length of 4 bytes: 255, 254, 253, 251.

$$P = 255 * 254 * 253 * 251 = 4113089310 \tag{14}$$

Thus storing residues should be maximally great and prime in pairs numbers, which could be got into such quantity of bits that fits to one residue.

For an increasing of number quantity in index, which can be expressed in the SRC, a dynamic change algorithm of residues quantity in number expressed in the SRC.

In the case when a situation of number repletion, which is expressed in the SRC, appears (situation when a sum of several SRC is exceeded P) the number is added by additional residue and in that way the maximum quantity of stored numbers P increases.

In the case when a large quantity of residues exists, with the purpose of storage redundant decrease a converting of system bases is performed with increase of each residue bit length .

In the case of using 2 bytes instead of one we can store (15) numbers for one residue storage in 4 bytes.

$$P = 65535 * 65534 = 4294770690 \tag{15}$$

It follows that (15)has less data redundance in comparison with (14) and allows to express more numbers quantity in the SRC.

2.3 Structure of the Index

The main problem of using an approach, offered in [3], is an impossibility of index derivation by numbers expressed in the SRC with traditional methods, because it is impossible to compare directly numbers expressed in the system of residual classes. To compare such numbers it is required to execute certain arithmetic conversions. As usual to compare numbers expressed in the SRC we should convert each number to the radix numeration system for further compare. From the point of view of productivity this approach is not effective because of computation efforts to number conversion from the nonpositional notation to the radix numeration system and because of necessity to work with great numbers and bignums.

An algorithm of index derivation is offered to solve the problem concerned with of impossibility of index derivation by numbers expressed in the SRC with traditional methods.

Note a tree from the figure 1 as a line (16), arranging nodes in order of a tree traversal from left to right. The traversal is realized as in a left-side tree traversal in case of using of nested sets.

$$[1.1[1.1.1]1.2, 1.3[1.3.1, 1.3.2]] \tag{16}$$

For illustrative purposes descendant nodes are enclosed in brackets.

Encode each node of the tree using the method of nested intervals and convert numbers to the SRC. As a system of residual classes bases we use modules: 3, 5, 7.

To solve a problem of intervals cross-cups we use early known decision, which is based on chain fractions properties: refusal of using index '1' as an element of materialized path, and a root of tree equals "2.2".

As it was said earlier for identification of tree node it is not necessary to calculate all interval, where descendant nodes are. It is enough to calculate an interval begin. In this case the interval begin identifies tree node by unique way and is node code.

Results of conversions are represented in table:

Initial materialized path	Modified materialized path	Node code (interval begin)	Node code in the SRC
1	2.2	5/2	(2,0,5)/(2,2,2)
1.1	2.2.2	12/5	(0,2,5)/(2,0,5)
1.1.1	2.2.2.2	29/12	(2,4,1)/(0,2,5)
1.2	2.2.3	17/7	(2,2,3)/(1,2,0)
1.3	2.2.4	22/9	(1,2,1)/(0,4,2)
1.3.1	2.2.4.2	49/20	(1,4,0)/(2,0,6)
1.3.2	2.2.4.3	71/29	(2,1,1)/(2,4,1)

Rewrite the line (16), replaced there elements of reified path by a value of node code in the SRC:

$$\frac{(2,0,5)}{(2,2,2)}\left[\frac{(0,2,5)}{(2,0,5)}\left[\frac{(2,4,1)}{(0,2,5)}\right]\frac{(2,2,3)}{(1,2,0)},\frac{(1,2,1)}{(0,4,2)}\left[\frac{(1,4,0)}{(2,0,6)},\frac{(2,1,1)}{(2,4,1)}\right]\right] \tag{17}$$

Note that the tree in the line (17) is represented in sorted-out state. And the line, which encodes the tree, may be unambiguously formed in a process of tree processing if a transactional integrity of execution of modification operation its tops and edges.

For illustrative purposes futher we will operate with elements of materialized path meaning element codes in the SRC.

Rewrite the line (16), removed from materialized path data about parent node:

$$1\big[1[1]2,3[1,2]\big] \tag{18}$$

Switch from string indication of the line (18) to binary. The figure 2 clearly shows intervals of encoded tree, which are stored in index.

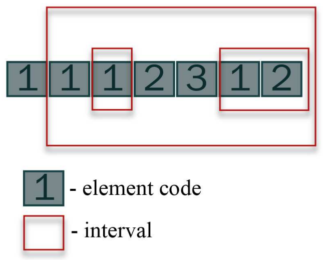


Fig. 2. Nested intervals of encoded tree

Offered index structure is the most productive when using in combination with highly productive data warehouse. As an example we will consider realization of this index in NoSQL DBMS MongoDB.

In binary form the index is stored as pages (documents) consecution with contiguous information. Besides node code each index record contains this document (or link to it) and reference to parent node. The index structure is presented in the figure 3.

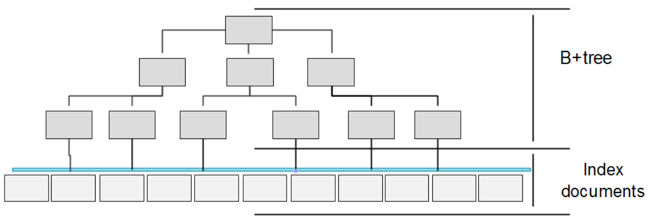


Fig. 3. Index structure

The index includes B+ tree and documents, which store data about nodes and relations between them.

A simplified diagram of index documents is presented in the figure 4. This scheme extends the figure 2 by adding new connections between records.

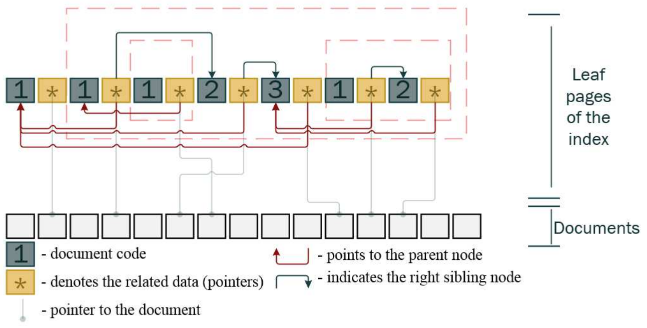

1
*
1
*
1
*
2
*
3
*
1
*
2
*
Leaf pages of the index
Documents
1 - document code
* - denotes the related data (pointers)
- pointer to the document
- points to the parent node
- indicates the right sibling node

Fig. 4. A simplified diagram of index

Each record in index document store the following data: record code, link to the right sibling node, document corresponding tree node, or link to it, and link to parent node.

In spite of the fact that this scheme stores relations between tree nodes, search of required node is possible only by walkthrough of all records. For elimination of this problem it is necessary to derivate index for access of tree tops, which is encoded in the SRC. As index we will choose structure of type B+ tree with sheet-like records, which references to top codes of the line corresponding to tree.

To solve the problem of comparing numbers expressed in the SRC it needs to derivate B+ tree according to numbers expressed in the SRC as ordinary array. Comparing of two numbers of the SRC in this case occurs as comparing of usual areas. This scheme allows to disregard from the SRC conception by adding of some redundance.

Sheet-like records of the tree point to corresponding records in index sheet pages.

A simplified diagram of index is represented on the figure 5.

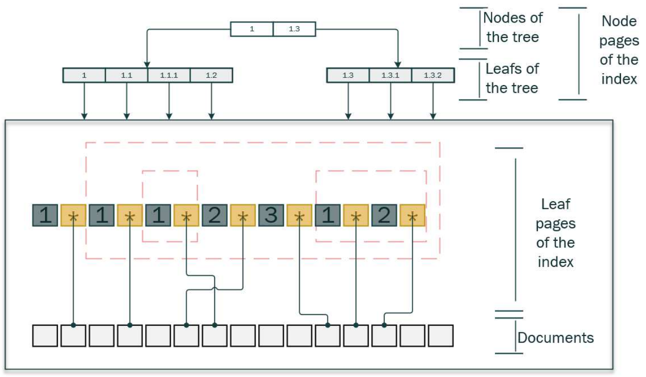


Fig. 5. Schematic representation of the index structure

However because of the fact that residues value in the SRC represented as array doesnt correspond to decimal value of number expressed in the SRC, elements in sheet pages of the tree would point to elements in sheet pages of index randomly (figure 6).

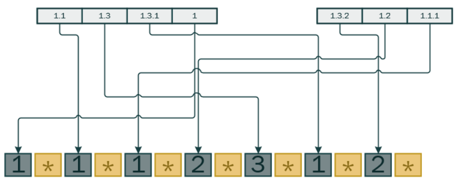


Fig. 6. Schematic representation of an accordance of records in sheet pages of the tree to records in sheet pages of the index

Encoded numbers in the SRC and switched from line representation to binary, we get the index structure, which is in the figure 7.

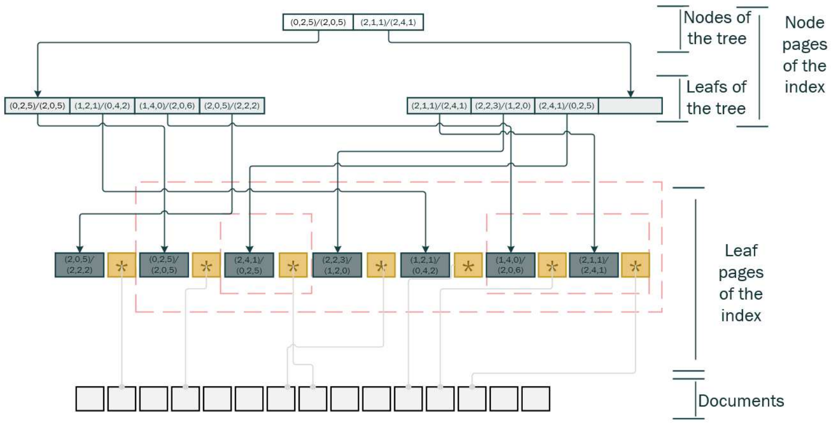


Fig. 7. Schematic map of the index structure

Such record allows to read trees and subtrees quickly and to perform operations of recording: an addition, a removal and displacement of nodes/subtrees.

Sheet-like index pages are replaced on the disc in the form of the double-linked list (figure 8)in the order of the tree nodes traversal. All links are using a multilevel addressing.

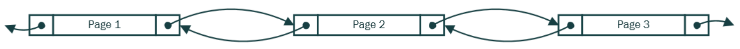


Fig. 8. A consecution of pages in the index file

High-speed operations of tree manipulation in this index are possible due to during data storage in the form of consecution of pages/documents a task of migration of data parts (for instance, during a subtree removal) comes to fragmentation/unification of boundary pages (which contain data partly) and to changing of links between pages with the purpose of maintenance of tree consecution arrangement in the index.

As result of the fact that each residue of number expressed in the SRC doesnt depend on other residues of the same number, we get an possibility of the high-speed converting of the subtree with using of parallel computations.

3 Experiments and Results

Testing results of presented in the article index structure in comparison with the realization of trees storage by the method of nested intervals and with the index of B+ tree type are shown in the table 1. As DBMS it was choosen NoSQL DBMS MongoDB. Offered index is realized as separate module, which expands a functionality of MongoDB. For parallel computations the graphics processor NVIDIA and the technology CUDA were applied.

Table 1. Results of productivity comparasing

Characteristic	Offering index	Nested intervals + B+ tree
Time of node insertion	0.00005 sec	0.00007 sec
Time of node removal	0.00005 sec	0.000001 sec
Time of subtree moving (10000 nodes)(CPU)	28.8 sec	25.4 sec
Time of subtree moving (10000 nodes)(GPU)	0.0007 sec, without data copying (between CPU and GPU) 0.42 sec with data copying (between CPU and GPU)	A supporting of GPU in MongoDB is not realized

4 Conclusion

In the work the index structure for storage of hierarchies in DB is suggested. For derivation of the index it is used the B+ tree, which is necessary for high-speed finding of document locations on the disc, and array of documents which are the index base. Also the method of index derivation on numbers expressed in the SRC is represented.

Using of this index structure in common with methods, allowing to realize parallel computations, permits to increase speed of working with trees.

Offered index structure can be applied in databases of NoSQL style to increase of productivity of index structure processing.

This approach can be improved by adding of methods of vector residue compression for decreasing of overhead expenses on storage and increasing of data processing rate.

References

1. V. Tropashko: Nested Intervals Tree Encoding with Continued Fractions. ACM SIGMOD, Volume 34, Issue 2, 2005
2. J. Celko: Joe Celkos Trees and Hierarchies in SQL for Smarties. Morgan Kaufmann, 2004
3. A. Malikov, A. Turyev: Nested Intervals Tree Encoding with System of Residual Classes. ICEICE No.2, 2011
4. N. Chervyakov: Modular Parallel Computing Structures of Neuroprocces System. FIZMAT, 2003
5. V. Tropashko: Trees in SQL: Nested Sets and Materialized Path: [Electronic resource]. 2003. URL: https://communities.bmc.com/docs/DOC-9902

Answer Set Programming and CLASP
A Tutorial

Steffen Hölldobler and Lukas Schweizer

International Center for Computational Logic
Technische Universität Dresden, 01062 Dresden, Germany
`sh@iccl.tu-dresden.de` `lukas@janeway.inf.tu-dresden.de`

Abstract. We provide a tutorial on answer set programming, a modern approach towards true declarative programming. We first introduce the required theoretical background in a compact, yet sufficient way and continue to elaborate problem encodings for some well known problems. We do so by also introducing the tools `gringo` and `clasp`, a sophisticated state-of-the-art grounder and solver, respectively. In that way we cover theoretical as well as practical aspects of answer set programming, such that the interested reader will gather sufficient knowledge and experience in order to continue discovering the field of answer set programming.

1 Introduction

Answer set programming (ASP) is a modern approach to declarative programming, where a user focusses on declaratively specifying his or her problem. ASP has its roots in deductive databases, logic programming, logic based knowledge representation and reasoning, constraint solving, and satisfiability testing. It can be applied in a uniform way to search problems in the classes P, NP, and NP^{NP} as they occur in application domains like planning, configuration, code optimization, database integration, decision support, model checking, robotics, system syntheses, and many more. We assume the reader to be familiar with propositional and first-order logic as well as with logic programming [5,13,3].

2 The Interview Example

A college in the USA uses the following rules for awarding scholarships to students: (1) Every student with a GPA of at least 3.8 is eligible. (2) Every minority student with a GPA of at least 3.6 is eligible. (3) No student with a GPA under 3.6 is eligible. (4) The students whose eligibility is not determined by these rules are interviewed by the scholarship committee.

These rules can be encoded in the following program:

$$\begin{array}{ll} \mathit{eligible}(X) & \leftarrow \mathit{highGPA}(X) \\ \mathit{eligible}(X) & \leftarrow \mathit{minority}(X) \wedge \mathit{fairGPA}(X) \\ \neg\mathit{eligible}(X) & \leftarrow \neg\mathit{fairGPA}(X) \\ \mathit{interview}(X) & \leftarrow \sim \mathit{eligible}(X) \wedge \sim\neg\mathit{eligible}(X) \end{array} \tag{1}$$

where *highGPA*(X) and *fairGPA* denote that the GPA of student X is at least 3.8 and at least 3.6, respectively, $\neg$ and $\sim$ denote classical and default negation, respectively, and we assume that all rules are universally closed. One should observe that the last rule specifies that *interview*(X) holds if neither *eligible*(X) nor $\neg$*eligible*(X) can be determined.

Now suppose the scholarship selection committee learns that John has a GPA of 3.7*, but his application does not give any information on whether he does or does not belong to a minority. What happens with John? Will he be invited to an interview?* This additional information can be specified by:

$$\begin{array}{rl} \mathit{fairGPA}(\mathit{john}) & \leftarrow \\ \neg \mathit{highGPA}(\mathit{john}) & \leftarrow \end{array} \tag{2}$$

3 Answer Sets

Propositional Programs For the moment we restrict ASP to propositional programs. Towards the end of this section we will extend it to first-order logic. The alphabet is the usual alphabet of propositional logic extended by the connective $\sim$denoting default negation. The set of *literals* consists of all propositional variables and their (classical) negations. A *rule* is an expression of the form

$$L_1 \vee \ldots \vee L_k \vee \sim L_{k+1} \vee \ldots \vee \sim L_l \leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge \sim L_{m+1} \wedge \ldots \wedge \sim L_n, \tag{3}$$

where all L_i, $1 \leq i \leq n$, are literals and $0 \leq k \leq l \leq m \leq n$. A *program* is a finite set of rules.

For a rule r of the form shown in (3) we introduce the following notation:

$$\begin{array}{ll} \mathit{head}(r) & = \{L_1, \ldots, L_k\} \cup \{\sim L_{k+1}, \ldots, \sim L_l\} \\ \mathit{head}^+(r) & = \{L_1, \ldots, L_k\} \\ \mathit{head}^-(r) & = \{L_{k+1}, \ldots, L_l\} \\ \mathit{body}(r) & = \{L_{l+1}, \ldots, L_m\} \cup \{\sim L_{m+1}, \ldots, \sim L_n\} \\ \mathit{body}^+(r) & = \{L_{l+1}, \ldots, L_m\} \\ \mathit{body}^-(r) & = \{L_{m+1}, \ldots, L_n\} \end{array}$$

Rule r is said to be a *constraint* if *head*$(r) = \emptyset$, i.e., constraints are of the form

$$\leftarrow L_{l+1} \wedge \ldots \wedge L_m \wedge \sim L_{m+1} \wedge \ldots \wedge \sim L_n; \tag{4}$$

it is said to be *classical* if *head*$^-(r)$ = *body*$^-(r) = \emptyset$, i.e., classical rules are of the form

$$L_1 \vee \ldots \vee L_k \leftarrow L_{l+1} \wedge \ldots \wedge L_m. \tag{5}$$

One should observe that for all classical rules r we find *head*(r) = *head*$^+(r)$ and *body*(r) = *body*$^+(r)$. A program is said to be *classical* if all its rules are classical.

Answer Sets for Classical Programs Let $\mathcal{P}$ be a classical program and $\mathcal{M}$ be a satisfiable set of literals, i.e., a set which does not contain a complementary pair A, $\neg A$ of literals. $\mathcal{M}$ is said to be *closed* under $\mathcal{P}$ iff for every (classical) rule $r \in \mathcal{P}$ we find that

$$\mathit{head}(r) \cap \mathcal{M} \neq \emptyset \quad \text{whenever} \quad \mathit{body}(r) \subseteq \mathcal{M}. \tag{6}$$

If we identify $\mathcal{M}$ with an interpretation such that $\mathcal{M}(L) = \mathit{true}$ iff $L \in \mathcal{M}$ for all literals L, then condition (6) states that whenever the body of a (classical) rule r is true under $\mathcal{M}$, then at least one literal in the head of r must be true as well. If condition (6) is satisfied for all rules in $\mathcal{P}$, then $\mathcal{M}$ is a *model* for $\mathcal{P}$. $\mathcal{M}$ is said to be an *answer set* for $\mathcal{P}$ iff $\mathcal{M}$ is minimal among the sets closed under $\mathcal{P}$ (relative to set inclusion).

As an example consider the program $\mathcal{P}$ consisting of the following rules:

$$\begin{aligned} s \vee r &\leftarrow \\ \neg b &\leftarrow r \end{aligned} \tag{7}$$

One may read these rules as *either the sprinkler is on* (s) *or it is raining* (r) and *if it is raining, then the color of the sky is not blue.* The sets $\{s, r, \neg b\}$, $\{s, \neg b\}$, $\{s\}$, $\{r, \neg b\}$ are closed under $\mathcal{P}$, but only the latter two are minimal and, thus, are answer sets. On the other hand, the sets $\emptyset$, $\{r\}$ and $\{r, s\}$ are not closed under $\mathcal{P}$. One should observe that if we add to $\mathcal{P}$ the constraint

$$\leftarrow s \tag{8}$$

then $\{s\}$ is no longer an answer set for the extended program. This example illustrates a general property of constraints: adding a constraint to a program affects its collection of answer sets by eliminating the answer sets which *violate* the constraints.

Reducts Let $\mathcal{P}$ be a program and $\mathcal{M}$ a satisfiable set of literals. The *reduct of* $\mathcal{P}$ *relative to* $\mathcal{M}$, in symbols $\mathcal{P}|_{\mathcal{M}}$, is defined as

$$\{\mathit{head}^+(r) \leftarrow \mathit{body}^+(r) \mid r \in \mathcal{P} \wedge \mathit{head}^-(r) \subseteq \mathcal{M} \wedge \mathit{body}^-(r) \cap \mathcal{M} = \emptyset\}, \tag{9}$$

where the literals in $\mathit{head}^+(r)$ and $\mathit{body}^+(r)$ are disjunctively and conjunctively connected, respectively [10].

As an example consider the program $\mathcal{P} = \{\neg p \leftarrow {\sim} p\}$. We obtain:

$$\begin{aligned} \mathcal{P}|_{\emptyset} &= \{\neg p\} \\ \mathcal{P}|_{\{p\}} &= \emptyset \\ \mathcal{P}|_{\{\neg p\}} &= \{\neg p\} \end{aligned} \tag{10}$$

One should observe that the reduct of a program relative to a satisfiable set of literals is a classical program.

Answer Sets for Programs Let $\mathcal{P}$ be a program and $\mathcal{M}$ a satisfiable sets of literals. $\mathcal{M}$ is said to be an *answer set* for $\mathcal{P}$ iff $\mathcal{M}$ is an answer set for $\mathcal{P}|_{\mathcal{M}}$. Hence, $\mathcal{M}$ is an answer set for $\mathcal{P}$ iff $\mathcal{M}$ is minimal and closed under $\mathcal{P}|_{\mathcal{M}}$.

Returning to the example discussed in the previous paragraph we observe that $\emptyset$ is not closed under $\mathcal{P}|_{\emptyset} = \{\neg p\}$ as $\neg p \not\in \emptyset$, $\{p\}$ is closed under $\mathcal{P}|_{\{p\}} = \emptyset$ but not minimal as $\emptyset \subset \{p\}$, and $\{\neg p\}$ is minimal and closed under $\mathcal{P}_{\{\neg p\}} = \{\neg p\}$. Hence, the only answer set for $\mathcal{P} = \{\neg p \leftarrow {\sim} p\}$ is $\{\neg p\}$. Hence, the rule $\neg p \leftarrow {\sim} p$ captures *negation by failure*: if one cannot show that p holds, then $\neg p$ is true.

Interested readers may try to compute answer sets for the following programs:

$$\{p \leftarrow {\sim} q\},\ \{p \leftarrow {\sim}\neg p\},\ \{q \leftarrow p \wedge {\sim} q,\ p \leftarrow,\ q \leftarrow\}.$$

What happens if we delete $q \leftarrow$ from the last example?

First-Order Programs Let $\mathcal{P}$ be a first-order program like the programs shown in (1) and (2). Let $\mathcal{T}$ be a set of terms. The set of *ground instances of* $\mathcal{P}$ *relative to* $\mathcal{T}$, in symbols $g\mathcal{P}$, is defined as follows:

$$\{r\theta \mid r \in \mathcal{P} \text{ and } \theta \text{ is a ground substitution for } r \text{ with respect to } \mathcal{T}\} \tag{11}$$

There is a bijection between the ground atoms occurring in $g\mathcal{P}$ and a suitable large set of propositional variables and, hence, $g\mathcal{P}$ is equivalent to a propositional program.

4 The Interview Example Revisited

Let $\mathcal{P}$ be the set of rules shown in (1) and (2) and let $\mathcal{T} = \{john\}$. Then, $g\mathcal{P}$ contains the following rules:

$$\begin{array}{ll} eligible(john) & \leftarrow highGPA(john) \\ eligible(john) & \leftarrow minority(john) \wedge fairGPA(john) \\ \neg eligible(john) & \leftarrow \neg fairGPA(john) \\ interview(john) & \leftarrow {\sim}\, eligible(john) \wedge {\sim}\neg eligible(john) \\ fairGPA(john) & \leftarrow \\ \neg highGPA(john) & \leftarrow \end{array} \tag{12}$$

Now let

$$\mathcal{M} = \{fairGPA(john),\ \neg highGPA(john),\ interview(john)\}. \tag{13}$$

The reduct of $g\mathcal{P}$ relative to $\mathcal{M}$ contains the following rules:

$$\begin{array}{ll} eligible(john) & \leftarrow highGPA(john) \\ eligible(john) & \leftarrow minority(john) \wedge fairGPA(john) \\ \neg eligible(john) & \leftarrow \neg fairGPA(john) \\ interview(john) & \leftarrow \\ fairGPA(john) & \leftarrow \\ \neg highGPA(john) & \leftarrow \end{array} \tag{14}$$

$\mathcal{M}$ is minimal and closed under this reduct and, hence, is the only answer set for $g\mathcal{P}$. Reasoning with respect to this answer set tells the selection committee that it should invite John for an interview.

Now suppose that the selection committee learns during the interview that John belongs to a minority. Let

$$\mathcal{P}' = \mathcal{P} \cup \{minority(john) \leftarrow\}. \tag{15}$$

In this case, $\mathcal{M}$ is no longer an answer set because the new rule is an element in $\mathcal{P}'|_{\mathcal{M}}$ and, consequently, $\mathcal{M}$ is not closed under $\mathcal{P}'|_{\mathcal{M}}$. It is easy to see that

$$\mathcal{M}' = \{fairGPA(john),\ \neg highGPA(john),\ minority(john),\ eligible(john)\} \tag{16}$$

is the only answer set for $\mathcal{P}'$. This example demonstrates that reasoning with respect to answer sets is non-monotonic as the addition of new knowledge may lead to the revision of previously drawn conclusions.

5 ASP Modeling in Practice

We now want to provide an inside on the methodology of how problems are encoded into programs that can be processed by ASP tools like `gringo` and `clasp` [6]. The first tool, `gringo`, takes a program and transforms it to its propositional equivalent - i.e. *grounding* the first-order program as defined in Section 3. For the resulting program, `clasp` is able to compute all answer sets – which refers to the *solving* process and `clasp` is therefore a *solver*.

We have chosen some well known problems taken from constraint programming and SAT-solving to demonstrate step by step the modeling process and its underlying paradigm.[1] From now on, we will use the syntax for programs accepted by `gringo` and explain each new construct used.[2]

Sudoku *The famous number riddle Sudoku represents a constraint problem, typically defined on a* 9×9 *board, where numbers* $1 \dots 9$ *are placed on each cell. The goal is to complete a given board such that in each row, column, and square the numbers* $1 \dots 9$ *occur exactly once.*

Now lets develop an encoding of Sudokus as programs. We do not bother about how to solve a Sudoku at all, since we take a truly declarative approach and simply describe the problem. Explaining the game to another person, one would start with defining the board layout, i.e. there is a 9×9 grid, and the

[1] We use the terms *modeling*, *encoding* or *reducing* a problem rather than *programming*, because in ASP one does actually not program in terms of control structures specifying a solving strategy.

[2] We use version 3.0.4 of `gringo`, and 2.1.1 of `clasp`. One should be careful with `gringo` versions ≥ 4, since the syntax is different than the one introduced in this tutorial.

notion of nine rows and columns as well as nine non-overlapping squares of size 3×3. We can express these by the following facts:

$$\begin{array}{l} \mathit{number}(1..9).\ \mathit{row}(0..8).\ \mathit{column}(0..8). \\ \mathit{square}(0,0..2,0..2).\ \mathit{square}(1,0..2,3..5).\ \mathit{square}(2,0..2,6..8). \\ \mathit{square}(3,3..5,0..2).\ \mathit{square}(4,3..5,3..5).\ \mathit{square}(5,3..5,6..8). \\ \mathit{square}(6,6..8,0..2).\ \mathit{square}(7,6..8,3..5).\ \mathit{square}(8,6..8,6..8). \end{array} \tag{17}$$

Rules are separated by a dot. *Facts* are rules with an empty body, in which case the implication symbol is omitted in `gringo` and `clasp`. $\mathit{number}(1..9)$ is a syntactical shorthand notation representing the nine facts $\mathit{number}(1)$, $\mathit{number}(2)$, ..., $\mathit{number}(9)$. We use numbers as constants to make use of these syntactic abbreviations. One could of course also use alpha-numeric constants. Saving these lines to a file `sudoku.lp`, we can already have a look at the resulting program by typing the command:

```
gringo sudoku.lp --text
```

The `--text` option prints the grounded program in a human readable way. One should see now that all facts are listed line by line and shorthand expressions are fully spelled out. But there is no need for actual grounding, as we have not used first-order variables yet. Therefore, we want to change the way the squares are defined.

$$\begin{array}{lll} \mathit{square}(0,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X < 3,\ Y < 3. \\ \mathit{square}(1,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X < 3,\ Y > 2,\ Y < 6. \\ \mathit{square}(2,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X < 3,\ Y > 5. \\ \mathit{square}(3,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X > 2,\ X < 6,\ Y < 3. \\ \mathit{square}(4,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X > 2,\ X < 6,\ Y > 2,\ Y < 6. \\ \mathit{square}(5,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X > 2,\ X < 6,\ Y > 5. \\ \mathit{square}(6,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X > 5,\ Y < 3. \\ \mathit{square}(7,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X > 5,\ Y > 2,\ Y < 6. \\ \mathit{square}(8,X,Y) & \text{:-}\ \mathit{row}(X),\ \mathit{column}(Y),\ X > 5,\ Y > 5. \end{array} \tag{18}$$

Now, each square is defined via a rule, specifying the corresponding ranges of the X and Y values, where :- is the syntactic equivalent to the implication symbol ($\leftarrow$) used so far. Replacing the *square* facts by these new rules in the `sudoku.lp` file and calling gringo again, yields the same result as before. This is interesting as we would expect grounded rules where X and Y are replaced by the constants $1 \ldots 9$. Instead we only find the grounded *square* atoms in the output as intended. It is easy to see that with a naive grounding approach, the resulting grounded program for these nine rules and the nine constants would already be quite large. The `gringo` grounder applies highly sophisticated grounding strategies. In our case, this even includes the full evaluation of the rule bodies. One should note that this is only possible in our case since in (17) we used the naturals numbers $0 \ldots 8$ as constants for rows and columns and the comparison relations ($<$, $>$, $=$, $!=$, $>=$, $<=$) are already predefined for natural numbers. We are not going into details of grounding strategies here; the interesting reader might want to look in [8] for details.

So far we have just encoded the basic Sudoku board. Explaining the game further, one would continue with providing the rules of the game, viz. that (a) in each of the 81 cells exactly one number in $\{1 \ldots 9\}$ can be placed, and (b) in each row, column and square a number is allowed to occur only once. Regarding (a), we introduce the rules

$$\begin{array}{rl}
\mathit{cell}(X,Y,1) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,4),\ \texttt{not}\ \mathit{cell}(X,Y,5),\\
& \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,8),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,2) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,4),\ \texttt{not}\ \mathit{cell}(X,Y,5),\\
& \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,8),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,3) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,4),\ \texttt{not}\ \mathit{cell}(X,Y,5),\\
& \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,8),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,4) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,5),\\
& \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,8),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,5) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,4),\\
& \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,8),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,6) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,4),\\
& \texttt{not}\ \mathit{cell}(X,Y,5),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,8),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,7) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,4),\\
& \texttt{not}\ \mathit{cell}(X,Y,5),\ \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,8),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,8) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,4),\\
& \texttt{not}\ \mathit{cell}(X,Y,5),\ \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,9).\\
\mathit{cell}(X,Y,9) \text{ :- } & \mathit{row}(X),\ \mathit{column}(Y),\\
& \texttt{not}\ \mathit{cell}(X,Y,1),\ \texttt{not}\ \mathit{cell}(X,Y,2),\ \texttt{not}\ \mathit{cell}(X,Y,3),\ \texttt{not}\ \mathit{cell}(X,Y,4),\\
& \texttt{not}\ \mathit{cell}(X,Y,5),\ \texttt{not}\ \mathit{cell}(X,Y,6),\ \texttt{not}\ \mathit{cell}(X,Y,7),\ \texttt{not}\ \mathit{cell}(X,Y,8).
\end{array}$$

each expressing that we can assert a number, e.g. 1, to the cell at position (X, Y) if we can ensure that none of the remaining numbers $2 \ldots 9$ is already asserted to (X, Y). We do so by using default negation `not`, the syntactic symbol for $\sim$ as used in `gringo` and `clasp`.

Now lets have a closer look on the semantics of these rules, as they have a very special character and role in programs. Assume we only have the first rule, then applying the rule yields a new fact stating that on position (X, Y) number 1 is placed, if X is a row and Y is a column and none of the remaining numbers $2 \ldots 9$ is already assigned to position (X, Y). Since we do not have further initial knowledge telling us that there is already another number placed on cell (X, Y), the rule is applicable for every position (X, Y) leading to a grid full of 1's. Considering all nine rules now, we are faced with some *non-determinism*, as for each cell we need to *guess* whether we place a 1 by applying the first rule, or lets say a 7 by applying the seventh rule. Note that the order of rules in a program does not play any role. In consequence, these nine rules define our (complete) *search space*, namely all 9^{81} number placements possible on a 9×9 board. Usually, such rules are called *generating* or *guessing* rules and represent and essential part of program encodings - we will later discuss this in more detail.

Apparently, we need to encode the Sudoku rules in (b) in order to suppress those guesses not representing a proper number assignment. We define the constraints

$$\begin{array}{l} \text{:- } cell(X,Y1,N),\ cell(X,Y2,N), Y1 \mathrel{!=} Y2. \\ \text{:- } cell(X1,Y,N),\ cell(X2,Y,N), X1 \mathrel{!=} X2. \\ \#hide. \\ \#show\ cell/3. \end{array} \tag{19}$$

which express that we would derive *false*, whenever we have the same number twice in one row (first rule) or column (second rules), respectively. With the help of the statements *#hide* and *#show cell*/3 `clasp` is instructed to print only the predicate *cell*/3 and to hide the other elements of an answer set.

Furthermore, we need a notion for a number to be in a square, and consequently rule out answer sets where a number does not occur in every square.

$$\begin{array}{rl} in\_square(S,N) & \text{:- } cell(X,Y,N),\ square(S,X,Y). \\ & \text{:- } number(N),\ \texttt{not}\ in\_square(S,N),\ square(S,\_,\_). \end{array}$$

Predicate *in_square* holds for square S and number N if we find N in a cell (X,Y) which belongs to square S. The second rule, i.e., the constraint, ensures that every number occurs in every square. Note that underscores, as occurring in the very last *square* predicate, represent anonymous variables which can be used if their assignment is not important, i.e. if they are not used in another predicate of the rule.

One might ask why $square(S,\_,\_)$ is actually needed? Are the first two body literals not sufficient? One can try dropping it from the constraint and call the grounder again. The grounder will state that the variable S is *unsafe* and, therefore, the rule is *unsafe*. In fact, `gringo` requires a program to be *safe*, which is the case if every rule occurring in the program is *safe*. A rule is *safe*, if every variable occurring in the rule occurs in some of the rule's positive body literals. State-of-the-art grounders like `gringo` require programs to be safe in order to guarantee termination.

Putting things together, we have fully encoded all rules of Sudoku. One should observe that we have not defined anything instructing how an instance of Sudoku needs to be solved – and we will not do so subsequently. This should emphasize the truly declarative approach. Saving the elaborated rules again to file `sudoku.lp` and calling

```
gringo sudoku.lp | clasp
```

will ground the program and pipe the output to `clasp`. We should get an output similar to the following:

```
clasp version 2.1.1
Reading from stdin
Solving...
Answer: 1
cell(0,1,6) cell(0,2,5) cell(0,4,3) cell(0,8,7) cell(1,0,1)
cell(1,2,7) cell(1,5,5) cell(2,2,8) cell(2,8,1) cell(3,3,2)
cell(3,5,1) cell(4,2,6) cell(4,4,8) cell(4,6,3) cell(5,3,5)
cell(5,5,3) cell(6,0,5) cell(6,6,6) cell(7,3,8) cell(7,6,4)
cell(7,8,3) cell(8,0,4) cell(8,4,7) cell(8,6,2) cell(8,7,1)
cell(0,0,2) cell(0,3,1) cell(0,5,8) cell(0,6,9) cell(0,7,4)
cell(1,1,4) cell(1,3,9) cell(1,4,2) cell(1,6,8) cell(1,7,3)
cell(1,8,6) cell(2,0,3) cell(2,1,9) cell(2,3,6) cell(2,4,4)
cell(2,5,7) cell(2,6,5) cell(2,7,2) cell(3,0,8) cell(3,1,5)
cell(3,2,3) cell(3,4,9) cell(3,6,7) cell(3,7,6) cell(3,8,4)
cell(4,0,9) cell(4,1,1) cell(4,3,7) cell(4,5,4) cell(4,7,5)
cell(4,8,2) cell(5,0,7) cell(5,1,2) cell(5,2,4) cell(5,4,6)
cell(5,6,1) cell(5,7,8) cell(5,8,9) cell(6,1,3) cell(6,2,2)
cell(6,3,4) cell(6,4,1) cell(6,5,9) cell(6,7,7) cell(6,8,8)
cell(7,0,6) cell(7,1,7) cell(7,2,1) cell(7,4,5) cell(7,5,2)
cell(7,7,9) cell(8,1,8) cell(8,2,9) cell(8,3,3) cell(8,5,6)
cell(8,8,5)
SATISFIABLE
Models : 1+
Time : 0.124s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.030ss
```

If `clasp` is called without options, it returns one answer set in case of satisfiability and indicates with `Models: 1+` that there are additional answer sets. The call

```
gringo sudoku.lp | clasp --number 5
```

requests an enumeration of up to five answer sets. Using `--number 0` requests all answer sets and should be used carefully since, like in our case, there are many possibilities to fill an empty Sudoku board.

Our encoding so far produces fully filled Sudoku boards, which all agree on the defined constraints and therefore represent solved Sudoku games. However, we do not yet solve a given partially filled Sudoku board. We need to provide a so-called *problem instance*. For example, we add subsequent *cell* facts.

$$\begin{array}{l}
cell(0,0,3).\ cell(0,4,8).\ cell(0,6,6).\ cell(0,8,7).\\
cell(1,1,1).\ cell(1,6,4).\ cell(1,8,9).\\
cell(2,0,8).\ cell(2,1,9).\ cell(2,4,6).\ cell(2,5,7).\\
cell(3,1,6).\ cell(3,3,1).\ cell(3,4,9).\ cell(3,6,7).\\
cell(4,2,9).\ cell(4,3,6).\ cell(4,4,5).\ cell(4,8,2).\\
cell(5,2,2).\ cell(5,7,1).\\
cell(6,1,5).\ cell(6,4,4).\ cell(6,8,3).\\
cell(7,1,4).\ cell(7,3,2).\ cell(7,7,9).\ cell(7,8,8).\\
cell(8,1,8).\ cell(8,2,6).\ cell(8,4,3).\ cell(8,6,1).
\end{array}$$

We save the assertions to a new file `suoku_instance.lp` and call `gringo` and `clasp` again:

```
gringo sudoku.lp sudoku_instance.lp | clasp --number 0
```

With `--number 0` we also instruct `clasp` to enumerate all models, which in our case should be exactly one.

Guess & Check Paradigm Let us reconsider the involved modeling steps in the Sudoku example. We started by specifying a fixed board layout by providing appropriate facts. The possible actions – placing exactly one number on each cell of the board – were modeled as *guessing* rules. We can see that part of an answer set program as the *guessing part*, where the search space for answer sets is defined. Secondly, we encoded constraints in order to *check* whether a generated solution is an answer set according to the rules of the Sudoku game.

This approach of modeling is often referred to as the *guess* & *check* or *generate-and-test paradigm* (see e.g. [8]). This is also motivated by *NP* problems, having a non-deterministical guessing of prospective solutions and subsequent checking. The interested reader might look at [12], from whom the paradigm originates.

At the end we added to our problem specification a concrete *problem instance.* The problem specification allows us to add any instance of some 9×9 Sudoku game. Therefore, such problem specifications are said to be *uniform.* We will use this separation into *problem specification* following the guess and check paradigm, and *problem instance* for all subsequent examples.

Graph Coloring We want to illustrate the *guess* & *check* paradigm again with a very concise and nice example - the *graph coloring problem.* In detail, *the problem asks whether there is some coloring of the nodes of a given undirected graph using n colors, such that no two nodes connected via an edge share the same color.* We restrict the example to $n = 3$. For encoding the problem we need the notion of *color*, *node*, *edge* and the *coloring* of nodes. We do so and stick to our paradigm by

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathit{color}(\mathit{green}).\ \mathit{color}(\mathit{red}).\ \mathit{color}(\mathit{blue}).} \\
\mathit{coloring}(X,\mathit{green}) & \text{:- } \mathit{node}(X),\ \texttt{not}\ \mathit{coloring}(X,\mathit{red}),\ \texttt{not}\ \mathit{coloring}(X,\mathit{blue}). \\
\mathit{coloring}(X,\mathit{red}) & \text{:- } \mathit{node}(X),\ \texttt{not}\ \mathit{coloring}(X,\mathit{green}),\ \texttt{not}\ \mathit{coloring}(X,\mathit{blue}). \\
\mathit{coloring}(X,\mathit{blue}) & \text{:- } \mathit{node}(X),\ \texttt{not}\ \mathit{coloring}(X,\mathit{green}),\ \texttt{not}\ \mathit{coloring}(X,\mathit{red}). \\
 & \text{:- } \mathit{coloring}(X1,C),\ \mathit{coloring}(X2,C),\ \mathit{edge}(X1,X2).
\end{array}$$

which fully specifies the problem. We have three colors specified in the first three facts, three guessing rules as well as a single constraint eliminating solutions where two nodes in an edge relation have identical coloring. We encode the *color-guess* analogously to the *number-guess* in the previous Sudoku example.

Because guessing is an essential part, the syntax offers so-called *choice rules.* As defined in `gringo`'s syntax documentation [7], choice rules are of the form

$$\{a_1, \ldots, a_m\} \text{ :- } a_{m+1}, \ldots, a_n, \texttt{not}\ a_{n+1}, \ldots, \texttt{not}\ a_k.$$

where $0 \leq m \leq n \leq k$, and each a_i is an atom for $0 \leq i \leq k$. This allows us to derive any subset of $\{a_1, \ldots, a_m\}$ (head atoms), provided that the body is satisfied. We could encode the following for coloring nodes:

$$\{coloring(X, green), coloring(X, red), coloring(X, blue)\} \text{ :- } node(X).$$

But since we can derive any subset for some node, this would yield nodes having more than one color assigned, or even no color at all. Therefore, the syntax allows an extension to put cardinality restrictions on the subset choice:

$$l\ \{a_1, \ldots, a_m\}\ u \text{ :- } a_{m+1}, \ldots, a_n, \texttt{not}\ a_{n+1}, \ldots, \texttt{not}\ a_k.$$

forcing the subset size to be within the lower bound l and upper bound u. I.e., we assign exactly 1 color to each node by

$$1\ \{coloring(X, green), coloring(X, red), coloring(X, blue)\}\ 1 \text{ :- } node(X). \quad (20)$$

ensuring that whenever we have some node X, we are allowed to add exactly one of the three head atoms to an answer set. We can even generalize rule (20) such that we do not need to partially instantiate the *coloring* predicate in the head. The syntax therefore offers so-called *conditional literals* of the form $l : l_1 : \ldots : l_n$, with $0 \leq i \leq n$. While grounding, l is instantiated under the conditions l_1 to l_n. It can be used for our purpose as follows:

$$1\ \{coloring(X, C)\ :\ color(C)\}\ 1 \text{ :- } node(X). \quad (21)$$

The grounding of expression $\{coloring(X, C)\ :\ color(C)\}$ expands to the one in (20). I.e. the set is generated by instantiating the *coloring* atom where C must be some *color*. We end up with a very compact encoding of the graph coloring problem:

$$\begin{array}{l} color(green).\ color(red).\ color(blue). \\ 1\ \{coloring(X, C)\ :\ color(C)\}\ 1 \text{ :- } node(X). \\ \text{:- } coloring(X1, C), coloring(X2, C), edge(X1, X2). \end{array}$$

For the problem instance, we provide a small graph with 4 nodes and 4 edges:

$$\begin{array}{l} node(1..4). \\ edge(1, 2).\ edge(1, 3).\ edge(3, 2).\ edge(3, 4). \end{array}$$

Saving the problem description to `coloring.lp` and the problem instance to `simple-graph.lp`, we can first get again an impression on the grounder's work:

```
gringo coloring.lp simple-graph.lp --text
```

This yields the following output besides existing facts:

```
1#count{coloring(4,blue),coloring(4,red),coloring(4,green)}1.
1#count{coloring(3,blue),coloring(3,red),coloring(3,green)}1.
1#count{coloring(2,blue),coloring(2,red),coloring(2,green)}1.
1#count{coloring(1,blue),coloring(1,red),coloring(1,green)}1.
:-coloring(3,green),coloring(4,green).
:-coloring(3,red),coloring(4,red).
:-coloring(3,blue),coloring(4,blue).
:-coloring(3,green),coloring(2,green).
:-coloring(3,red),coloring(2,red).
:-coloring(3,blue),coloring(2,blue).
:-coloring(1,green),coloring(3,green).
:-coloring(1,red),coloring(3,red).
:-coloring(1,blue),coloring(3,blue).
:-coloring(1,green),coloring(2,green).
:-coloring(1,red),coloring(2,red).
:-coloring(1,blue),coloring(2,blue).
```

It is interesting to see that the resulting program is already partially evaluated. For example, the *edge* atom in all constraints is missing, which is fine since it was merely used for instantiation of the other two *coloring* atoms. Also the choice rule was turned into four constraints, each for one of the nodes. I.e. the constraint

$$1\ \#count\{coloring(1, blue), coloring(1, red), coloring(1, green)\}\ 1.$$

represents the instantiation of rule (20) for node 1, where the *node* predicate in the body was instantiated with 1 and, therefore, omitted since, in this case, we only care about the head. With this constraint we fail if the *#count* function counts more than 1 occurrences of colorings for node 1 or less, respectively. We compute all answer sets for the union of the problem description (`coloring.lp`) and problem instance (`simple-graph.lp`) with:

```
gringo coloring.lp simple-graph.lp | clasp --number 0
```

For the provided graph, 12 colorings exist. The interested reader should try with an encoding of a more complex graph, e.g. the Petersen graph [15,16], in order to get an impression of `clasp`'s performance.

Traveling Salesman Problem Another combinatorial problem, but with an optimization aspect, is the so-called *traveling salesman problem (TSP)*. We want to demonstrate that answer set programming can also be applied to find optimal solutions. *A salesman is requested to visit some pre-defined cities. In order to be as efficient as possible, he wants to visit every city only once, as well as to travel the shortest roundtrip visiting all cities starting and ending in the same city.*

The problem can be separated into, (*a*) finding roundtrips beginning from and ending in the same city visiting all other cities only once, and (*b*) computing the length of each roundtrip in order to find the shortest. The first is known as the problem of finding *Hamiltonian cycles* in graphs, another well-known *NP*-complete problem. Therefore, we are faced with the optimization aspect to find the minimal Hamiltonian cycle in a given weighted graph, where the nodes in the graph represent our cities and labeled edges between nodes represent connections (e.g. highways or railways) between the cities. We encode the *TSP* problem description in exactly these steps, starting with appropriate guessing rules:

$$1\ \{cycle(X,Y) : edge(X,Y)\}\ 1 \text{ :- } node(X). \tag{22}$$

$$1\ \{cycle(X,Y) : edge(X,Y)\}\ 1 \text{ :- } node(Y). \tag{23}$$

Apparently, for a node to be in a cycle it must have an incoming edge as well as an outgoing edge. In the case of Hamiltonian cycles, a node in the cycle has exactly one incoming and one outgoing edge. This is specified in the rules (22) and (23). Again we use a cardinality restriction (exactly 1) on the head's choice expression $\{cycle(X,Y) : edge(X,Y)\}$. I.e., for every node X, in the first rule we choose exactly 1 of the instantiated $cycle(X,Y)$ literals, which the grounder instantiates with some Y whenever the conditional $edge(X,Y)$ is fulfilled. In (22) we choose an outgoing edge and derive that it belongs to the cycle, whereas in (23) we do so for incoming edges. Since for every node we non-deterministically pick one outgoing and incoming edge, we might have generated a cycle or not. To rule out model candidates not representing cycles, we check whether every node can be reached by every other node and exclude models where there is an unreachable node.

$$reachable(Y) \text{ :- } cycle(s,Y). \tag{24}$$

$$reachable(Y) \text{ :- } cycle(X,Y),\ reachable(X). \tag{25}$$

$$\text{ :- } node(X),\ \texttt{not}\ reachable(X). \tag{26}$$

Rule (24) and (25) define the notion for a node being *reachable* from all other nodes. It is defined recursively, i.e., a node Y is reachable if it is a direct neighbor of the starting node s (recursion base case). Furthermore, we conclude that node Y is reachable if for another reachable node X we find (X,Y) is in the *cycle*. We simply define the constraint (26) to fail whenever we have a node X for which we can not demonstrate its reachability. Saving (22)-(26) to `hamiltonian.lp` we can compute Hamiltonian cycles for some given graph, e.g.

$$\begin{array}{ll}
node(dresden). & node(petersburg). \\
node(novosibirsk). & node(stavropol). \\
node(moscow). & edge(stavropol, novosibirsk). \\
edge(dresden, moscow). & edge(moscow, petersburg). \\
edge(dresden, petersburg). & edge(moscow, stavropol). \\
edge(dresden, stavropol). & edge(moscow, novosibirsk). \\
edge(petersburg, novosibirsk). & edge(Y,X) \text{ :- } edge(X,Y).
\end{array} \tag{27}$$

encodes, without edge weights, the graph depicted in Figure 1. The last rule models symmetry of the *edge* relation in order to only provide one direction via

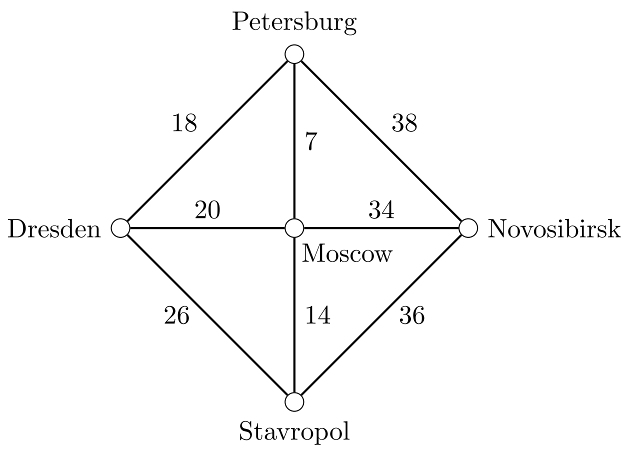


Fig. 1. Example graph representing cities and their connections.

the facts. We save (27) to `map.lp` and call:

```
gringo -c s=dresden hamiltonian.lp map.lp | clasp --number 0
```

With `-c s=dresden` the constant s in (24) is rewritten with *dresden*, in order to define that we want to start from the city of Dresden. We should receive 8 answer sets, each including instantiated *cycle* literals representing edges of the Hamiltonian cycle, similar as in subsequent output.

```
clasp version 2.1.1
Reading from stdin
Solving...
Answer: 1
cycle(dresden,stavropol) cycle(moscow,dresden)
cycle(stavropol,novosibirsk) cycle(petersburg,moscow)
cycle(novosibirsk,petersburg)
...
Answer: 8
cycle(dresden,moscow) cycle(moscow,stavropol)
cycle(petersburg,dresden) cycle(stavropol,novosibirsk)
cycle(novosibirsk,petersburg)
SATISFIABLE
Models : 8
Time : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.000s
```

The interested reader might try `simple-graph.lp` from the previous example: Are there any Hamiltonian cycles?

We can compute cycles now, however, we have not respected the optimizing aspect of the *TSP* problem yet – i.e. Hamiltonian cycles with minimal overall distance. We need additional knowledge representing the distances between the cities, by defining the ternary predicate *distance* and providing the following instances:

$$\begin{array}{ll}
distance(dresden, moscow, 20). & distance(dresden, petersburg, 18). \\
distance(dresden, stavropol, 26). & distance(moscow, stavropol, 14). \\
distance(moscow, petersburg, 7). & distance(moscow, novosibirsk, 34). \\
distance(novosibirsk, petersburg, 38). & distance(stavropol, novosibirsk, 36). \\
distance(X, Y, C) \text{ :- } distance(Y, X, C). &
\end{array}$$

Intuitively, $distance(dresden, moscow, 20)$ represents the fact, that the distance between Dresden and Moscow is 20. And to keep it less complex, we express symmetry of distances via the last rule. Since we want to minimize a cycle's overall length, we need to know the length of each cycle. The syntax offers some build-in constructs for such purposes called *aggregate functions.* They are used within *aggregate atoms*, which are atoms of the form

$$l \ \#A[l_1 = w_1, \ldots, l_n = w_n] \ u.$$

Each literal l_i has an assigned weight w_i which is 1 if not given explicitly, and the function A is applied to the weights of all literals l_i, $1 \leq i \leq n$. Aggregate atoms can be used as constraints, or on the right hand-side of some variable assignment. For example, we use *#sum* in order to assign the sum of all distances between cities in some hamiltonian cycle.

$$circumference(N) \text{ :- } N = \#sum \ [cycle(X, Y) : distance(X, Y, C) = C]. \tag{28}$$

Lets disassemble rule (28). We know conditional literals already from choice rules like the one in (22), i.e. the expression $cycle(X, Y) : distance(X, Y, C)$ will be evaluated while grounding and yields instantiated $cycle(X, Y)$ atoms whenever there is a corresponding $distance(X, Y, C)$. The only difference now, is that we take the distance value C and use it as the weight. For $cycle(dresden, stavropol)$ and $cycle(mosow, dresden)$, we obtain the grounded rule

$$\begin{aligned}circumference(46) \text{ :- } 46 = \#sum[&cycle(dresden, stavropol) = 26, \\ &cycle(moscow, dresden) = 20].\end{aligned}$$

It should be clear to see now how *#sum* yields the sum 46, which is assigned to variable N and therefore the new fact $circumference(46)$ is introduced. We can save the distance facts and rule (28) to `distances.lp` and again request all answer sets via:

```
gringo -c s=dresden hamiltonian.lp map.lp distances.lp | clasp --number 0
```

One should see that each answer set now also includes *circumference* denoting the cycle's overall length. We now only need to instruct `clasp` to provide only the answer set having minimal *circumference* value. We do so using an objective optimization function, in our case *#minimize*.

$$\#minimize\ [circumference(N) = N]. \tag{29}$$

It operates on the same input as aggregate functions, but it does actually not represent a constraint nor a rule. It simply instructs `clasp` to print answer sets with optimal value, in our case the hamiltonian cycle with overall length 121.

```
clasp version 2.1.1
Reading from stdin
Solving...
Answer: 1
cycle(novosibirsk,stavropol) cycle(petersburg,novosibirsk)
cycle(moscow,petersburg) cycle(stavropol,dresden)
cycle(dresden,moscow) circumference(127)
Optimization: 127
Answer: 2
cycle(novosibirsk,stavropol) cycle(petersburg,novosibirsk)
cycle(moscow,dresden) cycle(stavropol,moscow)
cycle(dresden,petersburg) circumference(126)
Optimization: 126
Answer: 3
cycle(novosibirsk,stavropol) cycle(petersburg,moscow)
cycle(moscow,novosibirsk) cycle(stavropol,dresden)
cycle(dresden,petersburg) circumference(121)
Optimization: 121
OPTIMUM FOUND
Models : 1
 Enumerated: 3
 Optimum : yes
Optimization: 121
Time : 0.021s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time : 0.010s
```

We suggest to have a look at `clasp`'s optimization features documented in [7,8], which is quite involved since there are many command options instructing `clasp` on how to deal with optimal solutions. It is also worth to have a look at all other aggregate functions, namely *#count*, *#avg*, *#min* and *#max*.

We provide some more problems, which we suggest the interested reader to model and encode in a similar fashion as we have done in the previous examples. The planning problems *Cannibals and Missionaries* and *Towers of Hanoi* might be slightly more involved. Moreover, one can find more problems, as for example in [9,11,8].

The Seating Problem *Workshop organizers want to arrange the seating of a social dinner in such a way that participants share a table only together with* *participants they don't know (yet), respectively participants for who it is known that they know each other should not sit at the same table. There are n tables with some fixed number m of chairs available. Naturally, participants can only sit at one table and chair, however, tables do not need to be fully seated in case there are less than $m \times n$ seats.*

Cannibals and Missionaries We can also encode planning problems, like the famous one proposed by [2]. *Three missionaries and three cannibals must cross a river using a boat which can carry at most two people, under the constraint that, for both banks, if there are missionaries present on the bank, they cannot be outnumbered by cannibals (if they were, the cannibals would eat the missionaries). The boat cannot cross the river by itself with no people on board.*

Towers of Hanoi Another famous and historic planning problem is the hanoi tower problem. *It consists of three rods, and a number of disks of different sizes which can slide onto any rod. The puzzle starts with the disks in a neat stack in ascending order of size on one rod, the smallest at the top, thus making a conical shape.The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:*

– *Only one disk can be moved at a time.*
– *Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.*
– *No disk may be placed on top of a smaller disk.*

6 Conclusion

In this tutorial, we introduced the theoretical background by means of the interview example, i.e. the notion of a *program*, *rule types*, *answer sets* for programs, as well as the *reduct* of a program. Equipped with the theory, we had a closer practical look at some encodings of problems using the syntax of `gringo` in order to solve the problem with the `clasp` solver. Both tools emerged from the potassco project [6]. The introduced language constructs should cover the basics as far as possible, such that the interested reader should be able to model the three open problems from the previous section. All these problems and especially the provided problem instances are quite small, though `clasp` is able to deal with extreme large problem instances, as annual competitions demonstrate [1].

Especially for readers familiar with logic programming (e.g. in *prolog*), it might have immediately become clear, that we do not have to care about the ordering of rules in any way, which imposes the answer set approach to be truly declarative; and, as [8] point out, is a real separation of *logic* and *control* compared to other approaches. Also one might miss data structures as they are known as nested terms e.g. in prolog. In answer set programs n-ary tuples and (flat, since grounded) terms are the choice for data structures.

Further reading We have not covered details, for example, on using arithmetics or other language extensions, which are fully covered in [7,8]. In general we used the syntax accepted by `gringo` in version 3.0, as documented in [7]. Also, entire lectures could be dedicated to grounding and solving strategies, which is due to the fact that modern solvers use state-of-the-art SAT techniques. In [8] these insights are provided in detail.

We should also note, that although we used the potassco tools in this tutorial, there are several more systems available and used in academical as well as industrial applications. Just to mention, there is the *dlv* system introduced in [11] and professionally maintained by DLVSYSTEM s.r.l., some spin-off company of the university of calabria. Moreover there is the *Smodels* solver and corresponding grounder *Lparse*, which was the very first implementation of the stable model semantics for logic programs, developed by [14]. In fact Lparse imposes some unofficial standard syntax, which is also accepted as input by `clasp`. There is evan an answer set extension to prolog - *Answer Set Prolog (AnsProlog)*. In their comprehensive work, [9] use AnsProlog to introduce the *ASP* approach.

Among these systems, unfortunately their syntax is not standardized yet. Therefore the syntactic structures introduced in this tutorial may not work, or lead to other results when using other tools. However, there is an *ASP* standardization working group elaborating a common input language definition [4]. New versions of `gringo`, up from 4.0, stick to the current version of the standard.

References

1. Mario Alviano, Francesco Calimeri, Günther Charwat, Minh Dao-Tran, Carmine Dodaro, Giovambattista Ianni, Thomas Krennwallner, Martin Kronegger, Johannes Oetsch, Andreas Pfandler, et al. The fourth answer set programming competition: Preliminary report. In *Logic Programming and Nonmonotonic Reasoning*, pages 42–53. Springer, 2013.
2. Saul Amarel. On representations of problems of reasoning about actions. *Machine intelligence*, 3(3):131–171, 1968.
3. Krzysztof R Apt. *From logic programming to Prolog*, volume 362. Prentice Hall London, 1997.
4. Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Francesco Ricca, and Torsten Schaub. Asp-core-2 input language format, 2013.
5. Melvin Fitting. *First-order logic and automated theorem proving.* Springer, 2nd edition, 1996.
6. M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, and M. Schneider. Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124, 2011.
7. Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A user's guide to gringo, clasp, clingo, and iclingo. preliminary draft, 2010.
8. Martin Gebser, Benjamin Kaufmann Roland Kaminski, and Torsten Schaub. *Answer Set Solving in Practice.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan and Claypool Publishers, 2012.

9. M. Gelfond and Y. Kahl. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*. Cambridge University Press, 2014.
10. Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.
11. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL)*, 7(3):499–562, 2006.
12. Vladimir Lifschitz. Answer set programming and plan generation. *Artificial Intelligence*, 138(1):39–54, 2002.
13. J Lloyd. *Foundations of Logic Programming*. Springer, 1987.
14. Ilkka Niemelä and Patrik Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. In *Logic Programming and Nonmonotonic Reasoning*, pages 420–429. Springer, 1997.
15. Julius Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15(1):193–220, 1891.
16. Eric W. Weisstein. Petersen graph. In *MathWorld–A Wolfram Web Resource*. Visited on 11/04/14. `http://mathworld.wolfram.com/PetersenGraph.html`.