FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Local Optimization for Multi-Context Systems with Constraint Pushing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science (Computational Logic) (MSc)

im Rahmen des Studiums

Computational Logic (Erasmus-Mundus)

eingereicht von

Seif El-Din Bairakdar
Matrikelnummer 0928104

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Eiter
Mitwirkung: Dipl.-Ing. Dr. techn. Michael Fink
Univ. Ass. Dipl.-Ing. Thomas Krennwallner

Wien, 12.4.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Local Optimization for Multi-Context Systems with Constraint Pushing

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computational Logic) (MSc)

in

Computational Logic (Erasmus-Mundus)

by

Seif El-Din Bairakdar
Registration Number 0928104

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: O. Univ.-Prof. Dipl.-Ing. Dr. techn. Thomas Eiter
Assistance: Dipl.-Ing. Dr. techn. Michael Fink
Univ. Ass. Dipl.-Ing. Thomas Krennwallner

Vienna, 12.4.2011

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Seif El-Din Bairakdar
14 El-Shahid Mahmoud Foaad st., Heliopolis, Cairo 11341, Egypt

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

In memory of my father

Acknowledgements

First and foremost, I would like to express my deepest gratitude to Prof. Dr. Thomas Eiter, for his patience, understanding, insight and continuous support throughout the duration of my studies at TUW. In particular, I want to highlight the fact that he offered me the opportunity to join his research team for a short period of time, which bore the fruits of my very first publications in the scientific community. Moreover, I am very grateful for the student grant that I received from the Austrian Science Fund (FWF) as well as the partial sponsorship for the extra program fees.

I would like to acknowledge one of the unforgettable characters that I have met in my life so far, Thomas Krennwallner. Despite our never-ending arguments and debates that have covered a huge variety of topics, I admit that I really enjoyed most of our conversations. He shared his extensive knowledge and assisted with most of the encountered technical problems, to mention a few: theoretical formalizations, C++, design patterns, LaTeX, Linux, etc.

I would also like to express my thanks for Dao Tran Minh for his efficient help and support, especially with all the C++ implementation and experimentation issues, as well as his kind input with regards to theoretical questions. Additionally, I want to thank all the KBS staff for their support that was always available upon request.

Furthermore, I want to express my gratefulness towards the EMCL joint commission for accepting my application for the EMCL program, as well as the European Commission for granting me the Erasmus Mundus scholarship to study in Germany and Austria.

I want also to extend my thanks to friends who have tolerated my mood swings and jargon-filled conversations, helped with proofreading the thesis, designing the comics, adjusting the poster, and most importantly just for being who they are.

Last, but by no means least, I would like to thank my family in general, and specifically my mother. Without her continuous support, motivation and encouragement that have been provided throughout my life, none of this would have been possible. As for my father, your sudden and unexpected departure, left a great vacuum within me. I wish you were still around. I hope that I made both of you proud. Thank you.

Abstract

Multi-Context Systems (MCS) are formalisms that enable the inter-linkage of single knowledge bases called contexts, via bridge rules. In this thesis, we focus on Brewka and Eiter-style heterogeneous nonmonotonic MCS and develop a novel distributed algorithm for computing the equilibria of such MCS. We examine previous approaches for distributed MCS evaluation that have been implemented as part of the DMCS system. Moreover, the notion of association rules and the process of association rule extraction from the data mining field are recalled. None of the available techniques addressed the issue of local optimization within a distributed evaluation, which is the motivation behind our work. Our approach for local constraint pushing (DMCS-SLIM), relies on the coupling of some optimization techniques for distributed MCS evaluation with local association rules extraction. We prove DMCS-SLIM to be sound and complete. Furthermore, we present a prototypical implementation, which is used for empirical evaluation. We performed exhaustive set of experiments against several runtime parameters of our system as well as comparisons with existing approaches. We observed that our approach has potential to surpass the current approaches.

Kurzfassung

Multi-Context Systeme (MCS) sind Formalismen, die es einzelnen Wissensbasen (den Kontexten) erlauben, mittels sogenannten Bridge Regeln verbunden zu werden. In dieser Arbeit konzentrieren wir uns auf heterogene nicht-monotone MCS von Brewka und Eiter, und entwickeln einen neuartigen verteilten Algorithmus zur Berechnung der Equilibria dieser MCS. Wir untersuchen bisherige Ansätze für verteilte MCS-Auswertung, die als Teil des DMCS Systems implementiert wurden. Darüber hinaus verwenden wir sogenannte Assoziationsregeln und Assoziationsregel-Extraktion aus dem Bereich des Data Mining. Keine der verfügbaren Techniken beantwortet die Frage der lokalen Optimierung innerhalb einer verteilten Evaluation, die die Motivation unserer Arbeit ist. Unser Ansatz des Constraint Pushings (genannt DMCS-SLIM) beruht auf der Kopplung von Optimierungsverfahren zur verteilten MCS Auswertung mit lokaler Assoziationsregeln-Extraktion. Wir beweisen Korrektheit und Vollständigkeit von DMCS-SLIM. Darüber hinaus präsentieren wir eine prototypische Implementierung, die wir zur empirischen Auswertung heranziehen. Wir führten ausgiebige Experimente durch und testeten unser System mit mehreren Laufzeit-Parametern, und verglichen unseren Ansatz mit bereits bestehenden. Wir beobachteten, dass unser Ansatz das Potenzial besitzt, die aktuellen Ansätze zu übertreffen.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Russell and Norvig [2003] define *artificial intelligence* as the study and design of intelligent agents. Throughout history and within every major civilization, humans have long been fascinated about intelligent machines. McCorduck [2004] claims that thinking machines and artificial beings can be traced back to ancient Egyptian and Greek civilizations. In this thesis, we focus on contemporary theories with regards to subfields within artificial intelligence.

Lakemeyer and Nebel [1994] define *knowledge representation* as

> The area of artificial intelligence that deals with the problem of representing, maintaining, and manipulating knowledge about an application domain.

One can consider knowledge representation as one of the central subfields within the field of artificial intelligence, of which *knowledge based systems* (KBS) is of special interest to us. Akerkar and Sajja [2009] define KBS as computer-based system that uses and generates knowledge from data, information and knowledge. The key difference between KBS and traditional computer systems, is the capability to understand the information being processed before producing a decision.

Kadie [1988] explains *nonmonotonic* reasoning as a pattern of reasoning that allows an agent to make and retract conclusions from inconclusive evidence. Consider the following standard example for explanation. During a session between a person A and a nonmonotonic reasoner B. A informs B that "Tweety is a bird", and poses the question "Can Tweety fly?". B will check its knowledge about birds and reply "Yes, Tweety can fly". At this point, A provides another fact, "Tweety is a Penguin" and poses the same query "Can Tweety fly?". Thus, B will retract its previous conclusion and respond by "No, Tweety cannot fly".

A *distributed system*, as defined by Dollimore et al. [2005], is

> A system in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.

This definition captures a broad range of systems and ideas where networked computers can be usefully deployed. Modern distributed systems are also *heterogeneous*, as they comprise sub-systems built on many different platforms (Devanbu and Wohlstadter [2001]). There are

several reasons such as physical constraints, market conditions, available technical skills, and history.

In the last years, the rise of distributed systems as well as the World Wide Web gave a huge boost to the increasing interest in systems comprised of multiple knowledge bases.

The following is a briefly summarized list of the main contributions of this thesis:

- We formally present an adaptation of the notion of association rules under the framework of multi-context systems.
- We define an automatic procedure for association rule extraction.
- We extend an existing distributed algorithm for evaluating multi-context systems by augmenting it with constraint pushing strategy.
- We provide a prototypical implementation of our distributed approach.
- We perform a thorough experimental evaluation between several variants of the novel approach as well as previous distributed techniques.

In the sequel, we peek the interest of the reader by presenting a brief introduction and history overview of multi-context systems. Then, we briefly discuss the existing evaluation strategies for such systems, thus leading to the motivation of our work. After that, we present an organizational overview for the thesis.

1.1 Multi-Context Systems

The evolution of modern multi-context systems as we know them nowadays, took place over a long period of time with respect to the relatively young field of artificial intelligence. In fact, contexts as formal objects were first introduced with lean formalization by McCarthy [1993].

One of the goals of artificial intelligence has always been to simulate human intelligence. To this end, McCarthy [1993] described the use of the formalized contexts as one of the essential tools for reaching human-level intelligence by logic-based methods. In fact they described the ability to overcome the original restrictions of limited axiomatization of contexts as *transcending*, where it is needed to be able to comprehend the discovery of someone else, even if we cannot make the discovery ourselves.

Over the years, there were several research proposals for multi-context systems, most notably by Giunchiglia and Giunchiglia [1992], who mathematically defined contexts, inter-contextual information flow and an epistemologically adequate theory for reasoning. Giunchiglia and Serafini [1994] devised systems that allows the usage of multiple distinct logical languages with properties that do not hold in modal logic such as the propagation of inconsistency through the hierarchy of theories. Moreover, a parallel evaluation of monotone multi-context systems with classical theories using SAT solvers for the contexts was investigated by Roelofsen et al. [2004].

The separation between locally available information and combined information from all the existing contexts is the key to almost all of the existing multi-context systems proposals. In Ghidini and Giunchiglia [2001], the distinction was specified by defining two principles, namely the:

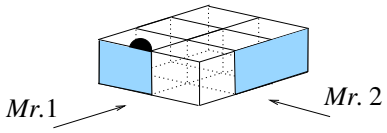


Figure 1.1: The magic box

- **Locality Principle**: reasoning uses only part of what is potentially available (e.g., what is known, the available inference procedures). The part being used while reasoning is what we call context (of reasoning); and the
- **Compatibility Principle**: there is compatibility among the reasoning performed in different contexts.

Illustrating these main intuitions, let us consider an example from Ghidini and Giunchiglia [2001] illustrated in Figure 1.1, McCarthy's magic box. We have placed a ball in a six-sector box, for our agents to locate. There are two agents Mr. 1 and Mr. 2, each looking at the box from a different angle. Mr. 1 can only determine whether the ball is to his left by using the proposition letter l or to his right by using another proposition letter r. Similarly Mr. 2, has the proposition letters r and l, in addition to c, which indicates that the ball is in the center. It is clear that neither of the two agents can pinpoint the exact location of the ball independently, as none of them possesses a depth factor. Thus, the locality principle can be considered as Mr. 1's and Mr. 2's individual perception of the environment. On the other hand, if both agents can communicate and share the partial information that they have, then they can easily decide the ball's location. The compatibility principle comes into play here as they are both observing the same box.

A unifying formalism for the aforementioned information distinction is the multi-context systems. Informally, if a context can be denoted by a C_i, one can consider each multi-context system as a collection of contexts, $M = (C_1, \ldots, C_n)$, where the inter-linking between contexts is achieved using bridge rules. Roughly, bridge rules offer a formal representation for the relation between atoms from different contexts in order for some atom to be valid at a certain context. A bridge rule schema for context C_k is: $s \leftarrow (c_1 : p_1), \ldots, (c_n : p_n)$, which means that s will be true at C_k if p_i is true w.r.t. C_i. Thus, one can formalize the magic box using propositional logic as follows:

In this work, we are interested in the framework devised by Brewka and Eiter [2007], as it generalizes previous approaches in contextual reasoning and allows for heterogeneous and nonmonotonic multi-context systems, i.e., with different, possibly nonmonotonic logics in its contexts (thus furthering heterogeneity), and bridge rules may use default negation (to deal, e.g., with incomplete information). Hence, nonmonotonic multi-context systems interlinking monotonic context logics are possible.

1.2 Multi-Context Systems Evaluation

Evaluating multi-context system $M = (C_1, \ldots, C_n)$ is a computational task that requires the usage of some special form of belief states $S = (S_1, \ldots, S_n)$. Roughly, one can think of each

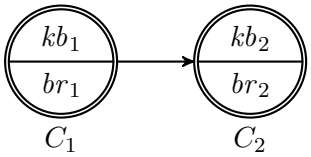


Figure 1.2: Magic box represented by a multi-context system

belief state as a collection of the local belief sets of each context that are compliant with the information stored in its knowledge base. Additionally, each belief set should contain information imported from other contexts via the bridge rules. During evaluation, it is customary to represent a multi-context system graphically. The magic box can have a pictorial representation as depicted in Figure 1.2.

Example 1 Let Mr. 1 and Mr. 2 from the Magic box be represented by contexts C_1 and C_2, respectively. Thus, as multi-context system, they can be expressed as $M = (C_1, C_2)$. The knowledge within each context is represented by a knowledge base. The bridge rules represent a the dependencies of a context on other contexts. Thus, for M we have:

- $kb_1 = \left\{ \begin{array}{lll} l_1 \vee r_1 & \leftarrow & \\ \text{not } r_1 & \leftarrow & \end{array} \right\}$;

  $br_1 = \left\{ \begin{array}{lll} l_1 \vee r_1 & \leftarrow & (2 : l_2) \\ l_1 \vee r_1 & \leftarrow & (2 : c_2) \\ l_1 \vee r_1 & \leftarrow & (2 : r_2) \end{array} \right\}$;

- $kb_2 = \left\{ \begin{array}{lll} l_2 \vee c_2 \vee r_2 & \leftarrow & \\ l_2 & \leftarrow & \end{array} \right\}$; and

  $br_2 = \emptyset$

kb_1 describes the knowledge of Mr. 1, where he knows that the ball can either be on the left or the right, and that it is currently not on the right. br_1 represents the knowledge that Mr. 1 will gain upon communication with Mr. 2. Similarly, kb_2 describes the knowledge of Mr. 2, where he knows that the ball can either be on the left, center, or right and that it is currently not on the left. br_2 is empty as there is no information to be gained from Mr. 2.

After evaluation, we get one belief state representing the equilibrium of the system, $S = (\{l_1\}, \{l_2\})$.

Eiter et al. [2005] define HEX-programs as nonmonotonic logic programs admitting higher-order atoms as well as external atoms. HEX-programs can deal with the external knowledge and reasoners of various natures. This is the reason why Brewka and Eiter [2007] decided to use HEX-programs to evaluate multi-context systems. They described the encoding of a multi-context system M into a HEX-program P_M, such that its computed minimal models (answer sets) correspond to the equilibria of M.

As each multi-context system represents a collection of contexts, then in theory it should be possible to adopt a distributed approach for system evaluation. To this end, Dao-Tran et al. [2010] provided a distributed technique for multi-context system evaluation, that requires each context to perform its local computation separately, and to communicate its partial solutions with the other contexts in order to discover the unified solutions.

Bairakdar et al. [2010a] discussed the shortcomings of the approach provided by Dao-Tran et al. [2010] and presented a distributed algorithm for a multi-context system evaluation. The described approach makes use of some meta information, such as the topological structure of the multi-context system, where it achieved economical small representations of context dependencies, as well as minimized data transmissions during context communication.

1.3 Motivation

Given the multi-context system framework under consideration, and all the existing distributed evaluation techniques, we noticed that no approach has investigated the possibility of optimizing the local evaluation at the contextual level.

In this thesis, we investigate the idea of optimizing local solving at each context with a distributed evaluation environment for multi-context systems. We attempt to develop an approach that both: prunes the search space, and reduces the run time. Thus, improving the local solving within each context.

The idea of our approach is that during system evaluation, each context should augment the queries issued to any other context by some kind of constraints. These constraints will prune the successor's search space, i.e., pushing forward information in the form of constraints to help in guiding the evaluation algorithm.

To illustrate the intuition behind our motivation consider the following scenario (Figure 1.3). Let C_1 and C_2 represent two people who have decided to share a large pizza at a restaurant. They are discussing which kind of toppings they should order. Naturally, both of them have to agree on the same toppings before an order is placed. Moreover, it might be the case that both of them have different sets of preferences. Let C_1 be not eager to consume meat at that point of time. This entails that both of them cannot order any pizza with toppings such as: chicken, beef, turkey, etc. Thus, as a response to the query imposed by C_1 with regards to the acceptable pizza toppings, C_2 should have been informed by C_1 that he is only interested in vegetarian toppings upfront. As a result, C_2 will have a pruned list of possible items to base his decision upon. Thus C_2's search space will be reduced, as well as the time required to reach a decision.

Pushing information forward is fruitful technique that is not new in distributed systems. For example, in the field of distributed database management systems, Bernstein et al. [1981] introduced *semijoin* which was enhanced by Najjar and Slimani [1998]. When the data is distributed at different sites, and large communication costs are incurred when performing regular joins in one step, semijoins are used. As a first step, semijoins help reducing the size of the relations to minimize the data transmission cost for processing queries.

The constraints are generated automatically at each context. They make use of the existing bridge rules and provide an insight on the truth values of the atoms, that are expected to hold among the other contexts referenced in the bridge rules. Thus, these contexts will only provide

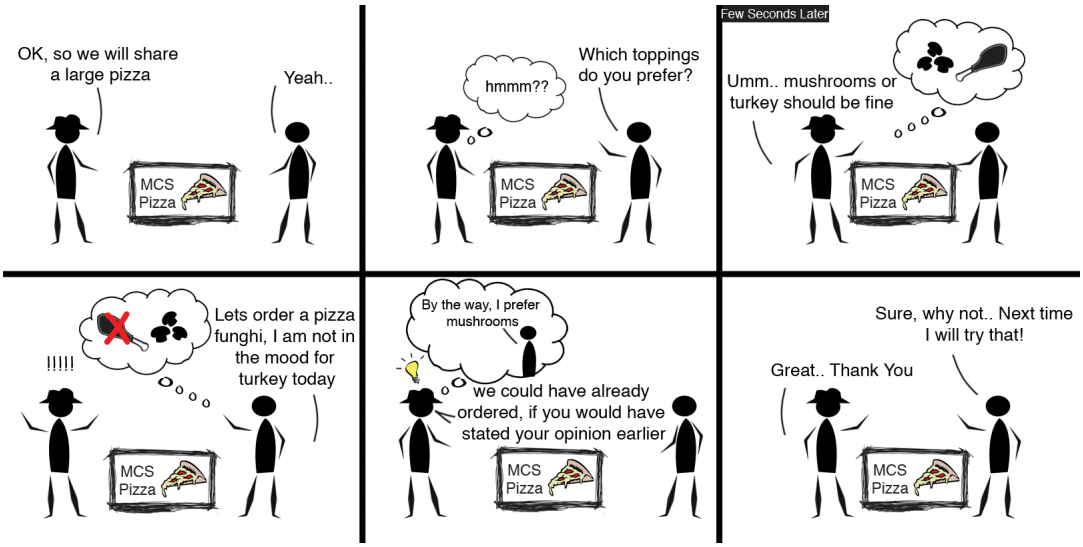


Figure 1.3: Ordering a pizza

models fitting such constraints. In our work, we utilized the notion of association rules introduced by Agrawal et al. [1993] from data mining to represent such constraints.

1.4 Thesis Contribution

In this thesis, we attempt to optimize the local evaluation process at each context within a distributed evaluation of heterogeneous nonmonotonic multi-context system by pushing constraints forward from one context to the next.

After careful consideration, we noted that the notion of association rules from data mining field can be utilized as a formalization of such constraints. To this end, we formally presented an adaptation of the association rules under the framework for monotonic heterogeneous multi-context systems.

Extracting association rules given a data set is a well known problem within the data mining field. In our thesis, we are interested in the automatic extraction of association rules from within each context during local computation. Thus, we defined an automatic procedure for association rule extraction.

After reviewing the existing distributed evaluation approaches, we decided to extend one of them by augmenting it for the constraint pushing strategy. We prove that our approach for multi-context systems distributed evaluation is sound and complete.

In order to evaluate our approach empirically, we provide a prototypical implementation, where we perform an exhaustive set of experiments using several run time parameters of our approach, as well as previous distributive techniques.

Our initial hypothesis was that the constraint pushing strategy would undoubtedly reduce the amount of partial solutions computed during local computations. The interpretation of our

benchmarking results shows that indeed there is an improvement with respect to the number of locally computed solutions as it had dropped by 20% in some setups and reached even 40% in other cases.

Additionally, we guess that certain parameters could have a significant effect on the overall performance. The experimental results reveal that it was indeed the case for some parameters like the maximum allowed size for association rules during the extraction process.

1.5 Thesis Organization

In this thesis, we explore the idea of local optimization for distributed evaluation of heterogeneous nonmonotonic multi-context systems using constraint pushing strategy. The remainder of this thesis is divided into five chapters, where:

- Chapter 2: we provide the necessary technical background information that covers the standard multi-context systems framework terminology. We also cover all the existing distributed approaches for multi-context systems evaluation, as well as the association rules original standard notations.
- Chapter 3: we adapt the notions of association rules and association rule extraction to the multi-context framework. We present our distributed approach for distributed equilibrium evaluation for multi-context systems using the local constraints pushing strategy, prove its soundness and completeness, and explore possible optimization techniques.
- Chapter 4: we introduce the prototypical implementation of our new Algorithm, which is the basis for the empirical evaluation.
- Chapter 5: we thoroughly evaluate our approach with regards to the existing distributed algorithms. The interpretation of our experimental results showed that indeed there is an improvement with respect to the number of locally computed solutions as it had dropped by 20% in some setups and reached even 40% in other cases.
- Chapter 6: we conclude our findings and discuss possible future work.

In Appendix A, we present a full trace of our running example using the new algorithm. The interested reader can consult that part to understand the inner workings of the algorithm. Furthermore, we refer to some excerpts as smaller examples throughout the thesis.

Chapter 2

Preliminaries

In this chapter, we introduce the basic building blocks of our work. We present all the technical background information required throughout the thesis. We cover the following areas in our discussions: Answer Set Programming, Multi-Context Systems, and Association Rules.

This chapter is divided into three sections. In Section 2.1, we provide a brief introduction to Answer-Sets Programming. In Section 2.2, we recall the Multi-Context Systems formalism, where we follow the evolution of distributed MCS and recall the most recent applicable decomposition techniques. In Section 2.3, we introduce the notion of Association Rules from the Data Mining field.

2.1 Answer Set Programs

Declarative programming is a programming paradigm that describes what a program should do rather than specifying how it should be done, i.e., describing the logic behind a computation without explicitly describing the control flow. Answer Set Programming (ASP) falls under this programming paradigm.

ASP's core has been introduced by Gelfond and Lifschitz [1988, 1991]. Throughout the years, ASP has been under constant development, we recall several notions from the detailed historical overview provided by Eiter et al. [2009].

There are several classes of logic programs, we recall the Disjunctive Logic Program (DLP), which subsumes other classes like normal logic programs.

In the sequel we restrict the notations for all the ASP related definitions to those as in (Eiter et al. [2009]) for simplicity.

Definition 1 (Eiter et al. [2009]) *Let $\mathcal{A}$ be a finite alphabet of atomic propositions. A* disjunctive rule *r is of the form*

$$a_1 \vee \cdots \vee a_k \leftarrow b_1, \ldots, b_m, \mathit{not}\, b_{m+1}, \ldots, \mathit{not}\, b_n \tag{2.1}$$

for $k \geq 0$ and $n \geq m \geq 0$, where $a_1, \ldots, a_k, b_1, \ldots, b_n \in \mathcal{A}$.

Let r be a disjunctive rule. The head of the rule r is defined as $H(r) = \{a_1, \ldots, a_k\}$, the body of the rule as $B(r) = B^+(r) \cup B^-(r)$, where $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$.

Syntactically, an answer set program P is a finite set of rules r in the form of (2.1). For introducing the semantics of P, we have to properly define an interpretation of P. An *interpretation* is a set $I \subseteq \mathcal{A}$ of atoms. Intuitively, $a \in I$ means a is true, and $a \notin I$ that a is false.

Definition 2 (Eiter et al. [2009]) *Let P be an answer set program, and let I be an interpretation. Then:*

- *I satisfies a rule r, ($I \models r$), if $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$.*
- *I satisfies P, if $I \models r$, $\forall r \in P$.*

Gelfond and Lifschitz [1991] defined the *GL-reduct* P^I of P relative to an interpretation I, which is the program obtained from P by deleting,

- every rule $r \in P$ such that $B^-(r) \cap I \neq \emptyset$
- all $not\, b_j$, for $b_j \in B^-(r)$, for every remaining rule r.

Definition 3 (Eiter et al. [2009]) *Given an answer set program P, an answer set of P is any interpretation I such that I is a $\subseteq$- minimal model of P^I.*

An application of the GL-reduct and answer sets generation, is shown in the following example.

Example 2 Let P be an answer set program, where P contains the rules

$$\begin{array}{rcl} a \vee b & \leftarrow & \\ d & \leftarrow & a, \mathrm{not}\, c \ . \\ e & \leftarrow & b \end{array}$$

P has two answer sets, namely: $I_1 = \{a, d\}$ and $I_2 = \{b, e\}$. The GL-reduct for each of the answer sets is as follows:

$$P^{I_1} = \left\{ \begin{array}{rcl} a \vee b & \leftarrow & \\ d & \leftarrow & a \\ e & \leftarrow & b \end{array} \right\} \text{and } P^{I_2} = \left\{ \begin{array}{rcl} a \vee b & \leftarrow & \\ d & \leftarrow & a \\ e & \leftarrow & b \end{array} \right\}$$

There are several other classes of logic programs in addition to disjunctive logic programs, where the structure of the rules is the only variant. For example, a *normal logic program* has the same structure, but the heads of the rules do not contain disjunction and consist of one atom only.

2.2 Heterogeneous Nonmonotonic Multi-Context Systems

In Section 1.1, we discussed the history behind the evolution of Multi-Context System (MCS). We take this discussion a step further by approaching MCS formally and recalling the technical details behind that notion in this section. To this end, we recall some of the basic definitions from the framework of heterogeneous nonmonotonic multi-context systems by Brewka and Eiter [2007], on which our work is based upon. Afterwards, we discuss a couple of distributed evaluation techniques. First, we cover the basic distributed evaluation technique, then we consider some applicable decomposition methodologies which eventually amount to another distributed approach.

2.2.1 Basic Multi-Context Systems Definitions

The framework for heterogeneous nonmonotonic multi-context systems was developed by Brewka and Eiter [2007]. As our work is entirely based on this framework, we proceed by recalling some of its notions.

Definition 4 (Brewka and Eiter [2007]) *A* logic *is, viewed abstractly, a tuple* $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$*, where*

1. $\mathbf{KB}_L$ *is a set of well-formed knowledge bases, each being a set (of formulas),*
2. $\mathbf{BS}_L$ *is a set of possible belief sets, each being a set (of formulas), and*
3. $\mathbf{ACC}_L \colon \mathbf{KB}_L \rightarrow 2^{\mathbf{BS}_L}$ *is a function describing the "semantics " of the logic by assigning to each* $kb \in \mathbf{KB}_L$ *a set of acceptable belief sets.*

This formalization of logic covers many (non)monotonic KR formalisms like description logics, modal logics, defeasible logic, normal logic programs under answer set semantics, default logic, etc. In our work we focus on (propositional) answer set programs logic (ASP logic), which is defined as.

Definition 5 *A (propositional)* ASP logic L *may be defined, such that:*

- $\mathbf{KB}_L$ *is the set of answer set programs over a (propositional) alphabet* $\mathcal{A}$*,*
- $\mathbf{BS}_L = 2^{\mathcal{A}}$ *contains all subsets of atoms,*
- $\mathbf{ACC}_L$ *assigns each* $kb \in \mathbf{KB}_L$ *the set of all its answer sets (more details about ASP follow later).*

Other logics can easily be expressed utilizing such formalization. Consider the next example for another kind of logic.

Example 3 A *Default logic* (Reiter [1980]) L may be defined, such that:

- $\mathbf{KB}_L$ is the set of default theories over an alphabet $\mathcal{A}$,

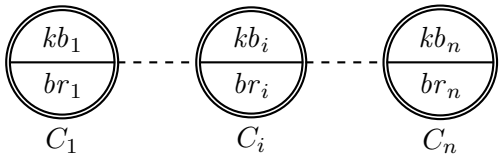


Figure 2.1: An abstract representation for a multi-context system

- $\mathbf{BS}_L$ is the set of deductively closed sets of $\mathcal{A}$-formulae,
- $\mathbf{ACC}_L$ assigns each $kb \in \mathbf{KB}_L$ the set of $\mathbf{KB}_L$'s extension.

Within a multi-context system, contexts are interlinked via bridge rules to other contexts. They are the means by which information flow occurs. Formally, a bridge rule is defined as follows.

Definition 6 (Brewka and Eiter [2007]) *Let $L = \{L_1, \ldots, L_n\}$ be a set of logics. An L_i-*bridge rule *over L, $1 \leq k \leq n$, is of the form*

$$s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \text{not}\,(c_{j+1} : p_{j+1}), \ldots, \text{not}\,(c_m : p_m) \tag{2.2}$$

where $1 \leq c_k \leq n$, p_k is an element of some belief set of L_{c_k}, $1 \leq k \leq m$, and for each $kb \in \mathbf{KB}_k : kb \cup \{s\} \in \mathbf{KB}_i$.

Roughly speaking, bridge rules represent additional knowledge of a context that depends on the knowledge of another (external) context. In fact, one can modify a context's knowledge base by adding s, if it is believed in the other contexts.

Now that we have formally introduced logics and bridge rules, we can recall the syntactic definition of a multi-context system.

Definition 7 (Brewka and Eiter [2007]) *A* multi-context system (MCS) *$M = (C_1, \ldots, C_n)$ consists of* contexts *$C_i = (L_i, kb_i, br_i)$, $1 \leq i \leq n$, where $L_i = (\mathbf{KB}_i, \mathbf{BS}_i, \mathbf{ACC}_i)$ is a logic, $kb_i \in \mathbf{KB}_i$ is a knowledge base, and br_i is a set of L_i-*bridge rules *over $\{L_1, \ldots, L_n\}$.*

Recall our previous understanding of a context within a multi-context system as an agent behaving in a multi-agent environment. In that case, one can think of each context as having its own language that might be different from the other contexts, i.e., the logic it utilizes. Additionally, one can consider the knowledge base of the context as its own set of beliefs and knowledge and the bridge rules as the means to gather the required information from other contexts as well as passing its knowledge to other contexts within the same environment. An abstract graphical representation for a MCS, without considering the bridge rules requirements is depicted in Figure 2.1.

The semantics of a multi-context system is by *belief states*. Informally, one can think of each belief state as the current state of each context with respect to its knowledge base and bridge rules. The current state of each context is one of its possible belief sets.

Definition 8 (Brewka and Eiter [2007]) *Let $M = (C_1, \ldots, C_n)$ be an MCS. A* belief state *is a tuple $S = (S_1, \ldots, S_n)$ such that each S_i is an element of $\mathbf{BS}_i$.*

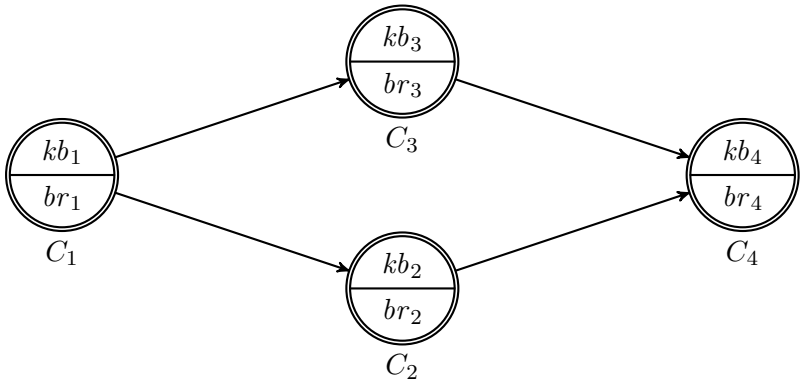


Figure 2.2: Multi-context system from Example 4

Intuitively, S_i should be a belief set of the knowledge base kb_i, while taking into consideration the bridge rules br_i. For this purpose, kb_i is augmented with all the conclusions of all $r \in br_i$ that are applicable.

A bridge rule r of the form (2.2) is applicable in a belief state $S = (S1, \ldots, S_n)$ iff $p_i \in S_{c_i}$, for $1 \leq i \leq j$ and for $j + 1 \leq k \leq m$: $p_k \notin S_{c_k}$.

Let $app(R, S)$ denote the set of all bridge rules $r \in R$ that are applicable in S. Furthermore, for any r of form (2.2), $head(r)$ denotes the literal s, and $B(r) = \{(c_k : p_k) \mid 1 \leq k \leq m\}$.

Generally any belief state for an MCS M, does not represent the whole system properly. It can only represent M as a whole if each rule in each knowledge base and bridge rule in each context are satisfied. To this end, one must make sure that the chosen belief states are in equilibrium.

Definition 9 (Brewka and Eiter [2007]) *A belief state $S = (S_1, \ldots, S_n)$ of a multi-context system M is an* equilibrium*, iff for all $1 \leq i \leq n$, $S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$.*

There are several notions of equilibria discussed in Brewka and Eiter [2007], such as grounded equilibria that works on a restricted class of MCS, and minimal equilibria which guarantee minimality of the equilibria. However, in this work, we focus on the basic notion of equilibrium that we have defined previously. To fully grasp the concept of equilibrium, let us consider the following example.

Example 4 (Dao-Tran et al. [2010]) Let $M = (C_1, C_2, C_3, C_4)$ be an MCS such that all L_i are ASP logics, with alphabets $\mathcal{A}_1 = \{a\}$, $\mathcal{A}_2 = \{b\}$, $\mathcal{A}_3 = \{c, d, e\}$, $\mathcal{A}_4 = \{f, g\}$. Suppose

- $kb_1 = \emptyset$, $br_1 = \{a \leftarrow (2 : b), (3 : c)\}$;
- $kb_2 = \emptyset$, $br_2 = \{b \leftarrow (4 : g)\}$;
- $kb_3 = \{c \leftarrow d;\ d \leftarrow c\}$, $br_3 = \{c \vee e \leftarrow \text{not}\,(4 : f)\}$;
- $kb_4 = \{f \vee g \leftarrow\}$, $br_4 = \emptyset$.

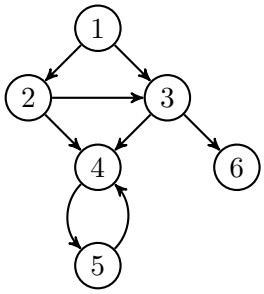


Figure 2.3: Scientist group example

One can check that $S = (\{a\}, \{b\}, \{c, d, \neg e\}, \{\neg f, g\})$ is an equilibrium of M. Additionally, the graphical representation of M is illustrated in Figure 2.2.

We examine the previous example (Example 4) rigorously to understand the concept behind the evaluation techniques for MCS. Context C_1 has an empty knowledge base, kb_1, which means that excluding its bridge rules, br_1, any belief state representing C_1 would have an empty set in place of the respective belief set. However, as each context possesses a set of bridge rules, br_i, in addition to its knowledge base, we have to pick the applicable rules among such a set and augment the kb_i with the head of such rules. C_1's has one bridge rule only, $a \leftarrow (2 : b), (3 : c)$, which means that a will be considered true iff b is valid in C_2 and c is valid in C_3 within the same belief state. Upon further investigation of the remaining contexts of M, one can notice that b and c are both valid in their respective contexts, which leads C_1 to have the belief set of $\{a\}$, which is part of the belief state equilibrium of M.

In the rest of this thesis, we assume that contexts C_i have finite belief sets S_i that are represented by truth assignments $v_{S_i} : \Sigma_i \to \{0, 1\}$ to a finite set Σ_i of propositional atoms such that $p \in S_i$ iff $v_{S_i}(p) = 1$ (as in Brewka and Eiter [2007], such S_i may serve as kernels that correspond 1-1 to infinite belief sets). Furthermore, we assume that the Σ_i are pairwise disjoint and that $\Sigma = \bigcup_i \Sigma_i$.

2.2.2 Distributed Nonmonotonic Multi-Context Systems

A distributed algorithm for computing equilibria of heterogeneous nonmonotonic multi-context systems was presented in Dao-Tran et al. [2010]. This algorithm provides a direct approach for distributed MCS evaluation. We recall the intuition behind it and provide a brief discussion as it is considered a part of the basis for our work.

Evaluating a system automatically and distributively, means that one can expect each context to perform some local individual evaluation and then all of these partial solutions are merged together to achieve the global solution for the whole system. In fact, this is almost a rough idea of the behavior of this algorithm. First, we introduce our running example, and then we recall some of the definitions and notations from Dao-Tran et al. [2010].

The MCS framework can conveniently capture the following scenario, which we use as a running example.

Example 5 (Bairakdar et al. [2010a]) A group of four scientists, Ms. 1, Mr. 2, Mr. 3, and Ms. 4, just finished their conference visit and are now arranging a trip back home. They can choose between going by train or by car (which is usually slower than the train); and if they use the train, they should bring along some food. Moreover, Mr. 3 and Ms. 4 have additional information from home that might affect their decision.

Mr. 3 has a daughter, Ms. 6. He is fine with either transportation option, but if Ms. 6 is sick then he wants to use the fastest vehicle to get home. Ms. 4 just got married, and her husband, Mr. 5, wants her to come back as soon as possible. He urges her to try to come home even sooner, while Ms. 4 tries to yield to her husband's plea.

If they go by train, Mr. 3 is responsible for buying provisions. He might choose either salad or peanuts. The options for beverages are coke or juice. Mr. 2 is a modest person as long as he gets home. He agrees to any choice that Mr. 3 and Ms. 4 select for vehicle but he dislikes coke. Ms. 1 is the leader of the group and prefers to go by car, but if Mr. 2 and 3 go by train then she would not object. A problem is that Ms. 1 is allergic to nuts.

Mr. 3 and Ms. 4 do not want to bother the group with their circumstances and communicate just their preferences, which is sufficient for reaching an agreement.

Ms. 1 decides which option to take based on the information she gets from Mr. 2 and Mr. 3.

The previous example is illustrated graphically in Figure 2.3. The final decision can only be reached if it satisfies all the different constraints from all the individuals. Now a question presents itself, how can such an evaluation be achieved?

In order to properly evaluate the scenario distributively (or otherwise) in Example 5, we have to provide a formal encoding as an MCS.

Example 6 (Bairakdar et al. [2010a]) The scenario in Example 5 encoded as an MCS $M = (C_1, \ldots, C_6)$, where all L_i are ASP logics and

- $kb_1 = \left\{ \begin{array}{rcl} car_1 & \leftarrow & \text{not } train_1; \\ \bot & \leftarrow & nuts_1 \end{array} \right\}$ and
  $br_1 = \left\{ \begin{array}{rcl} train_1 & \leftarrow & (2 : train_2), (3 : train_3); \\ nuts_1 & \leftarrow & (3 : peanuts_3) \end{array} \right\}$;
- $kb_2 = \{ \ \bot \ \leftarrow \ \text{not } car_2, \text{not } train_2; \ \}$ and
  $br_2 = \left\{ \begin{array}{rcl} car_2 & \leftarrow & (3 : car_3), (4 : car_4); \\ train_2 & \leftarrow & (3 : train_3), (4 : train_4), \text{not } (3 : coke_3) \end{array} \right\}$;
- $kb_3 = \left\{ \begin{array}{rcl} car_3 \vee train_3 & \leftarrow & ; \\ train_3 & \leftarrow & urgent_3; \\ salad_3 \vee peanuts_3 & \leftarrow & train_3; \\ coke_3 \vee juice_3 & \leftarrow & train_3 \end{array} \right\}$ and
  $br_3 = \left\{ \begin{array}{rcl} urgent_3 & \leftarrow & (6 : sick_6); \\ train_3 & \leftarrow & (4 : train_4) \end{array} \right\}$;
- $kb_4 = \{ \ car_4 \vee train_4 \ \leftarrow \ \}$ and $br_4 = \{ train_4 \leftarrow (5 : sooner_5) \}$;

- $kb_5 = \{\ sooner_5 \ \leftarrow \ soon_5\ \}$ and $br_5 = \{\ soon_5 \ \leftarrow \ (4 : train_4)\ \}$;
- $kb_6 = \{\ sick_6 \vee fit_6 \ \leftarrow \ \}$ and $br_6 = \emptyset$.

Within a multi-context system, context reachability plays an important role during a distributed evaluation. We recall the notion of import closure which formally captures this notion.

Definition 10 (Dao-Tran et al. [2010]) *Let $M = (C_1, \ldots, C_n)$ be an MCS. The* import neighborhood of a context C_k *is the set*

$$In(k) = \{c_i \mid (c_i : p_i) \in B(r), r \in br_k\}\ .$$

Moreover, the import closure $IC(k)$ of C_k *is the smallest set S such that (i) $k \in S$ and (ii) for all $i \in S$, $In(i) \subseteq S$.*

For each context within a multi-context system there is a set of reachable contexts via the bridge rule connections. The immediate reachable contexts for a context C_i, denoted $In(i)$, are the contexts that C_i will have to query at some point to properly evaluate the system. Meanwhile, $IC(i)$, shows the bigger picture from C_i's perspective, as this is the full set of contexts that should have finished their evaluation so that C_i can finish its own. Note that both $In(i)$ and $IC(i)$ can be the empty set in case of a leaf node.

Example 7 Consider M in Example 6. Then:

$$In(1) = \{2, 3\}, In(2) = \{3, 4\}, In(3) = \{4, 6\}, In(4) = \{5\}, In(5) = \{4\} \text{ and } In(6) = \emptyset$$

where the import closure of C_1 is $IC(1) = \{1, 2, 3, 4, 5, 6\}$.

Representing partial information generated from local evaluation without considering other contexts in an MCS, requires a different notion, which is formally stated in the following definition.

Definition 11 (Dao-Tran et al. [2010]) *Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} \mathbf{BS}_i$. A* partial belief state *of M is a sequence $S = (S_1, \ldots, S_n)$, such that $S_i \in \mathbf{BS}_i \cup \{\epsilon\}$, for $1 \leq i \leq n$.*

Compared to Definition 8, one notices that there is a representation for unevaluated contexts. Thus, from the point of view of any context, the *partial belief state* either contains some information for the other contexts or it does not care for them as they have not been evaluated yet.

We recall the adaptation of the notion of equilibrium (Definition 9) to match that of partial belief states.

Definition 12 (Dao-Tran et al. [2010]) *A partial belief state $S = (S_1, \ldots, S_n)$ of an MCS M is a* partial equilibrium of M w.r.t. a context C_k *iff $i \in IC(k)$ implies $S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$, and if $i \notin IC(k)$, then $S_i = \epsilon$, for all $1 \leq i \leq n$.*

The previous definition mirrors the notion of equilibrium in case of partial belief states. The partial equilibrium for an MCS M can be considered as equilibria for sub-MCS M' generated at a context C_k. Thus, the aim of each individual evaluation, in a distributed environment, is to calculate a partial equilibrium. Naturally, the ultimate goal of such computation is combining all the other contexts' partial equilibrium in order to eventually compute the partial equilibrium with respect to the root context.

The notion of combining several partial belief states, is formally known as joining the belief states.

Definition 13 (Dao-Tran et al. [2010]) *Given partial belief states $S = (S_1, \ldots, S_n)$ and $T = (T_1, \ldots, T_n)$, their* join *$S \bowtie T$ is defined as the partial belief state $(U_1, \ldots, U_n)$ with*

(i) $U_i = S_i$, *if* $T_i = \epsilon \vee S_i = T_i$, *and*

(ii) $U_i = T_i$, *if* $T_i \neq \epsilon \wedge S_i = \epsilon$,

for all $1 \leq i \leq n$.

Consequently, if we have two sets $\mathcal{S}$ and $\mathcal{T}$ of partial belief states then the *join* is naturally defined as $\mathcal{S} \bowtie \mathcal{T} = \{S \bowtie T \mid S \in \mathcal{S}, T \in \mathcal{T}\}$. The following example illustrates the result of applying the join to two sets of partial belief states.

Example 8 Consider two sets of partial belief states from our running example during evaluation at C_3:

$$\mathcal{S} = \left\{ \begin{array}{l} (\epsilon, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon)\,, \\ (\epsilon, \epsilon, \{train_3\}, \{train_4\}, \epsilon, \epsilon)\,, \\ (\epsilon, \epsilon, \{train_3\}, \{train_4\}, \epsilon, \{sick_6\})\,, \\ (\epsilon, \epsilon, \{train_3\}, \{car_4, train_4\}, \epsilon, \epsilon)\,, \\ (\epsilon, \epsilon, \{train_3\}, \{car_4, train_4\}, \epsilon, \{sick_6\}) \end{array} \right\} \text{ and}$$

$$\mathcal{T} = \left\{ \begin{array}{l} (\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)\,, \\ (\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \epsilon) \end{array} \right\}\,.$$

Their join is given by

$$\mathcal{S} \bowtie \mathcal{T} = \left\{ \begin{array}{l} (\epsilon, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon)\,, \\ (\epsilon, \epsilon, \{train_3\}, \{train_4\}, \epsilon, \epsilon)\,, \\ (\epsilon, \epsilon, \{train_3\}, \{train_4\}, \epsilon, \{sick_6\}) \end{array} \right\}\,.$$

Dao-Tran et al. [2010] developed DMCS, which is a distributed algorithm to find the equilibrium for an MCS M. The adopted approach is, given an MCS M and a starting context C_k, to find all partial equilibria of M w.r.t. C_k in a distributed manner. In DMCS, each context is represented by an instance of the algorithm which runs independently, communication is achieved via partial belief states exchange. Regardless of the logic used by each context, an application of this algorithm provides a method for distributed model building to any MCS, provided that appropriate solvers for the respective context's logic are available.

One of the main features of DMCS is its ability to compute *projected partial equilibria*, i.e., partial equilibria projected to a relevant portion of the signature of the import closure of the starting context. This functionality is achieved by the addition of the *relevant interface*, V, input parameter.

Given a (partial) belief state S and set $V \subseteq \Sigma$ of variables, the *restriction of S to v*, denoted $S|_V$, is given by the (partial) belief state $S' = (S_1|_V, \ldots, S_n|_V)$, where $S_i|_V = S_i \cap V$ if $S_i \neq \epsilon$, and $\epsilon|_V = \epsilon$; the restriction of a set of (partial) belief states $\mathcal{S}$ to V is $\mathcal{S}|_V = \{S|_V \mid S \in \mathcal{S}\}$.

Definition 14 (Dao-Tran et al. [2010]) *The* import interface *of context C_k is* $V(k) = \{p_i \mid (c_i : p_i) \in B(r), r \in br_k\}$. *The* recursive import interface *of C_k is* $V^*(k) = \bigcup_{i \in IC(k)} V(i)$.

The recursive import interface represents the interface of the import closure of C_k. Let C_r be the root context for an MCS M, then *minimum recursive import interface* is $V^*(r)$.

Example 9 Consider M from the running example (Example 6). The recursive import interface of C_1 in M from the running example is:

$$V^*(1) = \{train_2, car_3, train_3, peanuts_3, coke_3, car_4, train_4, sooner_5, sick_6\}.$$

We refrain from discussing the technical details of DMCS, instead we refer the interested reader to Dao-Tran et al. [2010].

Depending on the underlying logic at context C_k, it might be possible to compile br_k into kb_k, yielding a kb'_k, where it will lead to a simplification for the local solving at C_k, given all the partial equilibria of its neighbors. This can e.g. be done in case of ASP logics, as there are well-known transformations of ASP programs kb_k into equivalent classical theories $\phi(kb_k)$, such that the answer sets of kb_k are given by the classical models of $\phi(kb_k)$. This idea hinges on the notion of *loop formulas* (Lin and Zhao [2004], Lee and Lifschitz [2003]). The transformation procedure, $\pi(C_k)$, for each context with an MCS with normal ASP logics is provided in Dao-Tran et al. [2010]. This concept is mainly utilized for the implementation of SAT-solver based DMCS.

Although we do not present the DMCS algorithm itself, it is interesting to observe the control flow during the execution of the running example (Example 6).

Example 10 Let us consider invoking DMCS on C_1 from $M = (C_1, \ldots, C_6)$ in Example 6 with $V = \bigcup_{1 \geq i \geq 6} V(i)$. The initial call would be $C_1.\mathsf{DMCS}(V, \emptyset)$; this would eventually amount to calling all the contexts multiple times in the following order:

$$C_1, C_2, C_3, C_4, C_5, C_4, C_6, C_4, C_5, C_4, C_3, C_4, C_5, C_4, C_6$$

At the end of the execution, $C_1.\mathsf{DMCS}(V, \emptyset)$ computes three equilibria for M:

- $S = (\{train_1\}, \{train_2\}, \{train_3, urgent_3, juice_3, salad_3\}, \{train_4\}, \{soon_5, sooner_5\}, \{sick_6\})$;
- $T = (\{train_1\}, \{train_2\}, \{train_3, juice_3, salad_3\}, \{train_4\}, \{soon_5, sooner_5\}, \{fit_6\})$; and
- $U = (\{car_1\}, \{car_2\}, \{car_3\}, \{car_4\}, \emptyset, \{fit_6\})$.

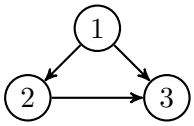


Figure 2.4: Simple MCS

The DMCS was the first approach to tackle the problem of MCS equilibrium evaluation in a distributed manner. Similarly to all ground-breaking algorithms, it could be classified as a basic approach. The authors of this approach had some ideas to improve the performance of the algorithm which included considering overall system topology during evaluation as well as finding means to reduce partial belief states transportation costs. Combining the previous ideas with some of their benchmark observations, an optimized version of the algorithm, DMCSOPT, was created, which we will review in the sequel.

2.2.3 Decomposition of Distributed Nonmonotonic Multi-Context Systems

DMCS is a basic direct approach for distributed equilibrium evaluation in MCS. Hence, the obvious need for optimization and improvement. Then, the authors of DMCS observed some scalability issues during the experiments. One of which is that contexts treat each other generically during evaluation as they are not aware of context dependencies. Another is that the number of models per contexts can scale to a huge size thus degrading the overall evaluation time. In Bairakdar et al. [2010a], some decomposition techniques were introduced with the aim of overall optimization of DMCS.

In the sequel, we recall some of the necessary basic notions and definitions. Additionally we briefly cover the utilized decomposition techniques. Finally, we recall the new algorithm and discuss some of its limitations.

Bairakdar et al. [2010a] present an optimization strategy that pursues two orthogonal goals: (i) to prune dependencies in an MCS and cut superfluous transmissions, belief state building, and joining of belief states; and (ii) to minimize information in transmissions.

One of the motivating ideas behind such optimization strategy is to avoid redundant calls to contexts, thus decreasing network resources consumption as well as local resources.

Example 11 Reconsider the running example (Example 6) with only contexts C_1, C_2, C_3 where all the other contexts and atoms referring to them are removed (Figure 2.4). Thus, C_1 would depend on C_2 and C_3, while C_2 depends on C_3. The straightforward evaluation for this MCS would be invoke both C_2 and C_3 during the execution of C_1.

In turn context C_2, would issue another query to C_3 as C_2's belief sets must be evaluated with respect to C_3's belief sets. Although simple caching strategies would improve the second belief state building in C_3, a transmission of belief states still occurs from C_3 to C_1. A call to C_3 can be avoided if C_2 reports the partial belief states to C_1 with the belief sets of C_3 along side its own belief sets (which are consistent with respect to C_3).

The key notion behind the first part of the optimization strategy is to acknowledge the significant value of information and knowledge that resides in graphical representation of the

contexts within an MCS.

Definition 15 (Bairakdar et al. [2010a]) *The* topology *of an MCS $M = (C_1, \ldots, C_n)$ is the digraph $G_M = (V, E)$, where $V = \{1, \ldots, n\}$ and $(i, j) \in E$ iff some rule in br_i has an atom $(j{:}p)$ in the body.*

The topology for the running example (Example 6) is depicted in Figure 2.3, where the nodes represent the contexts and the arrows represent the information dependency.

As the topology is in fact a directed graph, any further discussion requires standard graph terminologies (Bondy and Murty [2008]). To this end, we recall the notions utilized by Bairakdar et al. [2010a], some of which will not be directly used in this chapter (however, they are crucial for subsequent Chapters).

Graph Theoretic concepts: For any graph $G = (V, E)$ and set $S \subseteq E(G)$ of edges, $G \backslash S$ denotes the subgraph of G that has no edges from S. For a vertex $v \in V(G)$, $G \backslash v$ represents the subgraph of G induced by $V(G) \backslash \{v\}$. A *path* is a sequence of consecutive edges, and a graph is *connected* if there is a path connecting every pair of vertices. A *path graph*, P_n, is a is a tree with two nodes of vertex degree 1, and the other $n - 2$ nodes of vertex degree 2, it can be drawn so that all of its vertices and edges lie on a single straight line. A graph is *weakly connected* if replacing every directed edge by an undirected edge yields a connected graph. A vertex c of a weakly connected graph G is a *cut vertex*, if $G \backslash c$ is disconnected. A *biconnected graph* is a weakly connected graph without cut vertices. A *block* in a graph G is a maximal biconnected subgraph of G. Let $T(G) = (\mathcal{B} \cup \mathcal{C}, \mathcal{E})$ denote the undirected bipartite graph, called *block tree of graph G*, where $\mathcal{B}$ is the set of blocks of G, $\mathcal{C}$ is the set of cut vertices of G, and $(B, c) \in \mathcal{E}$ with $B \in \mathcal{B}$ and $c \in \mathcal{C}$ iff $c \in V(B)$. Note that $T(G)$ is a rooted tree for any weakly connected graph G; for arbitrary graphs, it is a forest.

The rationale behind utilizing the notion of block trees is that one can generate a block tree for a given MCS topology, where each block is optimized separately. Bondy and Murty [2008] offered several block optimization strategies, where in case for acyclic blocks, the edge pruning techniques are in fact *transitive reductions* of the graph G_M. In case of cycles, *ear decomposition* is applied to remove the edges. Thus the end result would be that each node in the original graph is visited only once.

Generating a block tree for our running example, where edge pruning is performed in each block would yield the following.

Example 12 (Bairakdar et al. [2010a]) The topology G_M of M in Example 6 is shown in Figure 2.3. It has two cut vertices, viz. 3 and 4; thus the block tree $T(G_M)$ (Figure 2.5) contains the blocks B_1, B_2, and B_3, which are subgraphs of G_M induced by $\{1, 2, 3, 4\}$, $\{4, 5\}$, and $\{3, 6\}$, respectively. The dashed edges are the removed edges after pruning the subgraphs in each block using either transitive reduction or ear decomposition.

The second goal of the underlying optimization strategy which handles the minimization of information needed for transmission between two neighboring contexts C_i and C_j, requires further notations.

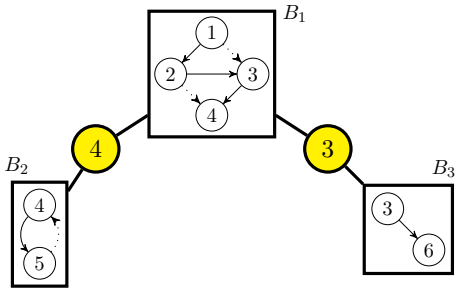


Figure 2.5: Scientist group example decomposition

Definition 16 (Bairakdar et al. [2010a]) *Given an MCS $M = (C_1, \ldots, C_n)$ and a subgraph G of G_M, for an edge $(i, j) \in E(G)$, the* recursive import interface of C_i to C_j w.r.t. G *is $V^*(i,j)_G = \{p \in V^*(i) \mid p \in \Sigma_\ell, j$ reaches ℓ in $G\}$.*

The rationale behind this definition is that if a context is a cut vertex c in G_M, all partial belief states representing the partial equilibria with respect to its successors are projected so that only its own belief sets survive. Thus, all information concerning the successors can be dropped before returning its result to the parent block of c in $T(G_M)$. This does not compromise the computation at the parent. Recursive import interfaces with respect to blocks in G_M capture this property and can be exploited for information minimization during transmission.

Bairakdar et al. [2010a] combined all these optimization techniques into one algorithm that generates the set of removable edges from the topological structure of a multi-context system represented as a block tree. It operates on a block tree T in a DFS-way, where the end result is a pair of all edges removed from blocks in T, and a labelling v for the remaining edges. The labelling v represents the set of interface variables that need to be transfered between contexts. Within each block, all cycles haven been broken (if any), and transitive reduction was used for edge removal. We refrain from discussing the topology optimization algorithm in depth, instead we refer the interested reader to Bairakdar et al. [2010a].

In order to properly adapt DMCS to the new decomposition techniques, one has to recall a further notion. Given an MCS M, and its topology, a stripped version of the topology that includes both the minimal context dependencies as well as transferable interface variables is a *query plan*.

Definition 17 (Bairakdar et al. [2010a]) *A* query plan of an MCS M w.r.t. context C_k *is any labeled subgraph Π of G_M induced by $IC(k)$ with $E(\Pi) \subseteq E(G_M)$, and edge labels $v\colon E(G) \to 2^\Sigma$.*

In our work, we use the query plan as constructed by Bairakdar et al. [2010a], where it is based on set of the removable edges from the aforementioned briefly discussed algorithm.

Combining the topology optimization techniques that are engulfed in the query plan for a certain MCS with the distributed evaluation of DMCS yielded DMCSOPT.

Abstracting from low-level implementation details, the idea of the algorithm is as follows: Given a query plan, Π_r, with respect to context C_r, we start with context C_k and traverse Π_k by expanding each outgoing edge, like in a depth first search, at each context. The expansion takes

Algorithm 1: DMCSOPT(c : context id of predecessor) at $C_k = (L_k, kb_k, br_k)$

Data: Π_r: query plan w.r.t. starting context C_r and label v, $cache(k)$: cache
Output: set of accumulated partial belief states

(a) **if** $cache(k)$ *is not empty* **then** $\mathcal{S} := cache(k)$ **else**
  $\mathcal{T} := \{(\epsilon, \ldots, \epsilon)\}$
(b) **foreach** $(k, i) \in E(\Pi_r)$ **do** $\mathcal{T} := \mathcal{T} \bowtie C_i$.DMCSOPT$(k)$ // neighbor beliefs
(c) **if** *there is* $i \in In(k)$ *s.t.* $(k, i) \notin E(\Pi_r)$ *and* $T_i = \epsilon$ *for* $T \in \mathcal{T}$ **then**
    $\mathcal{T} := \mathsf{guess}(v(c, k)) \bowtie \mathcal{T}$ // guess for removed dependencies in Π_r
(d) **foreach** $T \in \mathcal{T}$ **do** $\mathcal{S} := \mathcal{S} \cup \mathsf{lsolve}(T)$ // get local beliefs w.r.t. T
  $cache(k) := \mathcal{S}$
(e) **if** $(c, k) \in E(\Pi_r)$ *(i.e., C_k is non-root)* **then return** $\mathcal{S}|_{v(c,k)}$ **else return** $\mathcal{S}$

Algorithm 2: lsolve(S: partial belief state) at $C_k = (L_k, kb_k, br_k)$

Output: set of locally acceptable partial belief states
$\mathbf{T} := \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, S)\})$
return $\{(S_1, , \ldots, S_{k-1}, T_k, S_{k+1}, \ldots, S_n) \mid T_k \in \mathbf{T}\}$

place till a leaf context is reached. If C_i is a leaf that contains $(j{:}p)$ in the bodies of its bridge rules and there is no context C_j along the query plan to visit, then this means that a cycle was broken and thus an edge was removed. Subsequently, all the possible truth assignments for the import interface of C_j are considered. For any context , C_i, the result is the set of partial belief states, which comes from the join of the local belief sets and the neighbor results. The final result of the system is computed at C_k. A cache is used to store the partial belief states at each context, in order to minimize the re-computations.

According to Bairakdar et al. [2010a], the steps of DMCSOPT (Algorithm 1) are explained as follows:

(a) + (b) check the cache, if empty get neighbor contexts from the query plan, request partial belief states from all neighbors and join them;

(c) if there are $(i : p)$ in the bridge rules br_k such that $(k, i) \notin E(\Pi_r)$, and no neighbor delivered the belief sets for C_i in step (b) (i.e., $T_i = \epsilon$), we have to call guess on the interface $v(c, k)$ and join the result with $\mathcal{T}$: intuitively, this happens when edges had been removed from cycles;

(d) compute local belief states given the imported partial belief states collected from neighbors; and

(e) return the locally computed belief states and project to the variables in $v(c, k)$ for non-root contexts; this is the point were we mask out parts of the belief states that are not needed in contexts the lie in a different block of $T(G_M)$.

DMCSOPT utilizes two subroutines:

- lsolve(S) (Algorithm 2): This module augments the knowledge base kb of the current context with the heads of applicable bridge rules in br with respect to partial belief state S. This is achieved by computing the local belief sets using the $\mathbf{ACC}$ function, then merging them S and eventually returning the resulting set of partial belief states.

- guess(V): This is a unary function, that guesses all the possible truth assignments for the interface variables V.

In order to grasp the difference between DMCS and DMCSOPT, let us consider the following example. It illustrates the control flow during the execution of our running example (Example 6) with DMCSOPT:

Example 13 Invoking DMCSOPT on C_1 from $M = (C_1, \ldots, C_6)$ in Example 6 with Π_1 as the formal representation of Figure 2.5. The initial call would be with C_1.DMCSOPT(0), this would eventually amount to invoking all the contexts once, as the control flow would be:

$$C_1, C_2, C_3, C_4, C_5, C_6$$

At the end, C_1.DMCSOPT(0) computes two partial equilibria for M:
- $S = (\{train_1\}, \{train_2\}, \{train_3\}, \{train_4\}, \emptyset, \emptyset)$;and
- $T = (\{car_1\}, \emptyset, \{car_3\}, \{car_4\}, \emptyset, \emptyset)$.

Observe that in Example 13 there are two equilibria, while Example 10 produced three. This is because DMCSOPT utilizes the minimal interface projection which is part of the query plan. As a result, within the block that contains C_1, which is the context that we used to start the evaluation, the information regarding C_5 and C_6 are deemed irrelevant and are discarded at the respective cut vertices. Additionally, the import interface between the contexts has been restricted according to the query plan so that it conveys only the join-relevant information.

Although DMCSOPT produced exemplary results when compared to its predecessor DMCS, optimization is still possible. Despite the addition of the query plan notion and the improvements added to context interface variables, some local pruning techniques are still possible.

2.3 Association Rules

Given a set of data, one of the key approaches in the data mining field that provides descriptive information about the general properties within that data is *association rule mining* (Rabuñal et al. [2009]). Acquiring descriptive knowledge about data is a highly beneficial process, depending on the field where that data originated the usage will be different, as an example in case of the business field of customer relationship management, such insight helps to concentrate the efforts on customers that have a high likelihood to respond to offers.

In the sequel, we recall the notion of association rules as well as some of the related statistical metrics. Association rules were first introduced by Agrawal et al. [1993] in order to address the classical problem of "market basket analysis". This problem is clearly visible in stores and

supermarkets where stores aim at optimizing the product placement strategy. This is done by gathering information regarding customers buying habits from the daily transactions, where frequently jointly purchased products are discovered and thus they are placed in neighboring shelves accordingly. For example, it is almost always the case that when a customer buys chips, he buys some kind of beverage. Thus, if supermarkets place the chips next to the beverages shelves, they have a high probability to increase their revenue, as the customers' psyche will unconsciously remind them when they are buying chips to buy some kind of beverage from the neighboring shelf.

Prior to recalling the formal definition of association rules, one has to recall some other basic notions.

Definition 18 *A* transaction *is an n-ary tuple, where the first element is a unique transaction identifier. The remaining elements of the tuple are called an* itemset*. They are the items from the available binary data such that no item appears twice.*

Definition 19 (Agrawal et al. [1993]) *Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n binary attributes called* items*. Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of* unique transactions *called the* database*, where each transaction in D contains a subset of the items in I. An* association rule *is defined as an implication of the form $X \Rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$.*

In the previous definition X and Y are both itemsets and are called *antecedent* (left-hand-side, LHS) and *consequent* (right-hand-side, RHS) of the rule respectively. One can also denote them as head and body, respectively. Generally, one is interested in a sub class of itemsets, the *Frequent itemsets*, which are the itemsets satisfying a minimum support threshold.

The process of association rule extraction is usually guided by some statistical parameters (Maimon and Rokach [2005]), the most commonly used are *minimum support* and *confidence*.

- Support is the portion of transactions in the data set which contain the itemset. Let n be the total number of available transactions and a the number of transaction that contain X, then $supp(X) = \frac{a}{n}$.
- Confidence is an estimate of the probability $P(Y \mid X)$, which is the probability of finding the consequent of the rule, Y, in the transactions under condition that these transactions also contain the antecedent, X. Formally it can be represented as $conf(r) = \frac{supp(X \cup Y)}{supp(X)}$,

Example 14 Assume that we are working at a supermarket. We are tasked with the extraction of some association rules for the following set of items, $I = \{milk, bread, butter, beer\}$, where the relevant set of transactions are in Table 2.1.

Each row corresponds to a transaction. Each cell, apart from those in the transaction ID column, represents an item within a transaction, a "1" represents the existence of that item, and vice versa for "0". Some of the possible association rules would be:

a) $\{milk, bread\} \Rightarrow \{butter\}$: which means that whenever a customer buys milk and bread, we expect him to buy butter as well. The confidence of this rule is 0.5, which means that for 50% of the transactions that contain $milk$ and $bread$, this rule applies.

Transaction ID	milk	bread	butter	juice
1	1	1	0	0
2	0	1	1	0
3	0	0	0	1
4	1	1	1	0
5	0	1	0	0

Table 2.1: Supermarket transactions

b) $\{milk\} \Rightarrow \{bread\}$: which means that if milk is bought, customers also buy bread. The confidence of this rule is 1.0, which means that this rule is always valid.

c) $\{butter\} \Rightarrow \{bread\}$: which means that if butter is bought, customers would most likely buy bread. The confidence of this rule is 1.0, which means that this rule is always valid.

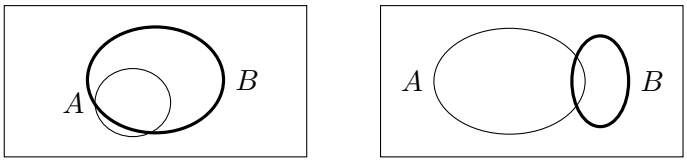
d) $\{bread\} \Rightarrow \{butter\}$: which means that if someone buys bread, he/she will probably buy butter. Unlike the previous rule, the confidence of this one is 0.5, despite of using the same itemsets.

In Example 14, we have assumed that all the investigated association rules were statistically significant, i.e., above the designated threshold for the support, or rather exceeded the minimum support. However, the value of confidence varied greatly among the chosen rules.

Consider Figure 2.6, each rectangular area represents some itemsets. We are interested in two specific itemsets; namely, A and B. The minimum support is graphically represented by $A \cup B$, while the intersection between each two sets signifies the confidence level when combined with a set's area. In Figure 2.6a, $A \Rightarrow B$ can be classified as a strong rule as it is almost always the case that when A is true B holds. However, the converse is not true, $B \Rightarrow A$ holds sometimes, thus it has a low confidence level. Consider Figure 2.6b, $A \Rightarrow B$ and $B \Rightarrow A$ are both considered association rules with a low confidence level. Conversely, in Figure 2.6d, $A \Rightarrow B$ and $B \Rightarrow A$ are both considered association rules with a very high confidence level. Thus, one can observe that both sides of any association rule are rarely interchangeable without having great ramifications on the association rule's confidence level.

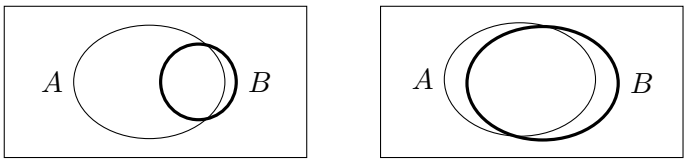
Practical applications usually contain thousands or millions of transactions, and an association rule usually requires a support of hundreds of transactions before being deemed statistically relevant.

The basic idea behind any association rule mining algorithm is to identify from the given data base, set of transactions, consisting of itemsets, whether the occurrence of specific items, implies also the occurrence of other items with a relatively high probability. Generally, this is a two step process. First, the minimum support is applied to find all frequent itemsets in a database. second, these frequent itemsets and the minimum confidence constraints are used to form the rules. There are several algorithms which handle this harvesting process, *Apriori* which was introduced by Agrawal and Srikant [1994], the *Partition* algorithm by Savasere et al. [1995], Zaki [2000] introduced six algorithms, KDCI developed by Orlando et al. [2003] among others.



(a) $conf(A \Rightarrow B)$ high
$conf(B \Rightarrow A)$ low

(b) $conf(A \Rightarrow B)$ low
$conf(B \Rightarrow A)$ low



(c) $conf(A \Rightarrow B)$ high
$conf(B \Rightarrow A)$ high

(d) $conf(A \Rightarrow B)$ high
$conf(B \Rightarrow A)$ high

Figure 2.6: Possible confidence levels

We shall not delve into the inner workings of association rule mining algorithms, as they will be utilized later on as inter-changeable black box elements in our work. We consider them as having an identical structure with respect to input and output parameters.

Chapter 3

Local Constraint Pushing for Distributed Multi-Context Systems

In this chapter, we present the formalization for the idea of local optimization of MCS via constraint pushing. To this end, we present the intuition behind our approach, adapt the notion of association rules and association rule extraction to the framework of MCS. Afterwards, we present a distributed evaluation algorithm for MCS, which we provide a proof for its soundness and completeness. Additionally, we present several parameterized optimization strategies and ideas for association rule extraction procedure.

3.1 Motivation and Intuition

Motivated by the previous chapters, we build upon the foundations of multi-context systems under the framework of Brewka and Eiter [2007]. We utilize the decomposition technique from Bairakdar et al. [2010a].

Informally speaking, the idea behind our work is to utilize the available information about each context effectively, which has been otherwise neglected. Upon examination of the control flow of the both previous approaches DMCS and DMCSOPT, we notice that each context invokes its neighbors, combines their results and then performs the local solving. Now the question comes up what would happen if the local computation took place before checking for the neighbors? If we just swap these parts, in either algorithms, we will end up with the same result eventually, which is expected. The key question that we bring up in our work is can we utilize some of the knowledge that was generated from the local computation at a each context to aid the computation at its neighbors?

During a distributed MCS evaluation with DMCSOPT, it might be the case that some context C_i, has a common atom a, among all the set of its partial belief states. This a might be a bridge atom from another context C_j. Using the standard computation methodology of DMCSOPT, a will only be discovered to be a common atom when all of the neighbors of C_i will return their respective sets of partial belief states and all the joining would have occurred. During the local

computation of C_j, it might be the case that the search space would have been pruned greatly if a it was known that the invoker of C_j is looking only for the literal a, thus eliminating the need for computing the models for $\neg a$. In other words, if C_i was able to push some of the information that it already possesses via its bridge rules to its neighbors as a set of facts, clauses or rather models, then in principle the overall computation time and search space of the neighboring atoms could decrease significantly.

Example 15 Given an MCS $M = (C_1, C_2)$, where all L_i are ASP logics and

- $kb_1 = \{\bot \leftarrow \text{not } a_1\}$ and
  $br_1 = \{a_1 \leftarrow (2 : b_2)\}$
- $kb_2 = \left\{ \begin{array}{l} c_2 \leftarrow \text{not } b_2; \\ b_2 \leftarrow \text{not } c_2 \end{array} \right\}$
  $br_2 = \emptyset$

Now we aim to find the equilibrium for M, utilizing DMCSOPT with C_1 as a root context. Tracing the execution we get:

a) The client initiates a call to C_1, checks for neighbors, finds C_2 and queries it. Context C_2 checks for neighbors, discovers that it is a leaf node, computes its local belief sets, transforms them into partial belief states, projects them to the interface $v(1, 2)$, which is $\{b_2\}$, and finally returns the result to its invoker, C_1:

   - $U = (\{\}, \{\text{not } b_2\})$
   - $T = (\{\}, \{b_2\})$.

b) Afterwards, C_1 computes its local belief sets and joins them with the results from C_2, which leads to the following equilibrium for M:

   - $S = (\{a_1\}, \{b_2\})$.

In Example 15, we observe that during the local evaluation for C_2, unnecessary belief states were generated. In fact, the belief state T was discarded entirely during the join operation in C_1. Upon closer examination for T, one notices that it could have lead to an inconsistent system as a_1 would be false. Our idea is to find the proper means by which one can push forward the information that C_1 is expecting the value of b_2 is going to be false. By pushing forward, we mean propagating the constraint that b_2 is true to all the neighbors of C_1, in this case C_2. This should help C_2 during local soiling, as it would prune away the search space for lsolve. Additionally, it will reduce the amount of partial belief states transmitted between contexts.

We have just introduced a simple example to show the effect of constraint pushing. Imagine a bigger setup, where the pruning effect will not be just one single belief state, but rather an exponential number of belief states. Even better, imagine a case where such constraint pushing techniques would readily lead to an inconsistency within the system, thus eliminating the need for useless computation along the expected control flow.

Pushing the information forward is not an easy task. As we are dealing with heterogeneous nonmonotonic MCS, a unified generic formalization had to be chosen. After investigating several fields, we noticed that looking for important information within a given set of data or rather extracting and formalizing descriptive knowledge is an idea that has been well exhausted in the field of data mining. We utilize the association rules introduced in Section 2.3 to handle the constraint pushing scenario.

3.2 Association Rules for Multi-Context Systems

In this section, we tailor the notion of association rules that was introduced in Section 2.3 so that it is compatible with heterogeneous nonmonotonic MCS.

Association rules are mainly characterized by their support and confidence values. Recall the statistical notions from Section 2.3, where the support represents the portion of transactions that contain a certain itemset and the confidence factor that is responsible for the strength of the rule as it is a measure of its validity. Soundness and completeness are the two corner stones of our work. To this end, we will always maintain the value of the confidence at "1.0", so that we consider association rules that are always true for a certain set of transactions. This is to ensure that the equilibrium of the system will not be contaminated. On the other hand, we have to ensure that the threshold for the minimum support will be as low as possible. This constraint is added to make sure that during the association rule mining process, no rule will go unchecked and thus jeopardizing the constraint pushing system as a whole.

The first amendment to the original definition (Definition 19), is to remove the transaction unique identifier from the definition of transaction. Thus a transaction is now a set of itemsets. This coincides perfectly with our notion of belief states. Thus one can consider there is a one-to-one correspondence between a (partial) belief state and a transaction, where the belief sets are indeed the item sets. As a result, one can consider the set of belief states as a set of transactions.

Notice that the belief sets, and by extension the belief states, contain only atoms that are assigned the value true; However, this is not the case for transactions. Transforming each belief state into a transaction means that all the positive atoms are considered as well as negative atoms. Additionally, all the unknown belief sets, marked with ϵ, are simply ignored.

Example 16 Consider the running example (Example 6) where C_4 computes the following set of partial belief states:

$$\mathcal{S} = \left\{ \begin{array}{l} (\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon) \\ (\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \epsilon) \\ (\epsilon, \epsilon, \epsilon, \{train_4\}, \{sooner_5\}, \epsilon) \end{array} \right\}.$$

Transforming $\mathcal{S}$ into a set of transactions, it would lead to the inclusion of the negative literals, which are a hidden part of $\mathcal{S}$. Thus, the set of transactions that correspond to $\mathcal{S}$ is $\mathcal{S}_t$,

represented as:

$$\mathcal{S}_t = \left\{ \begin{array}{l} (\{car_4, \text{not } train_4\}, \{\text{not } sooner_5\}) \\ (\{\text{not } car_4, train_4\}, \{\text{not } sooner_5\}) \\ (\{\text{not } car_4, train_4\}, \{sooner_5\}) \end{array} \right\}.$$

Definition 20 *Let $\mathcal{A}$ be a finite alphabet of atomic propositions. An* association rule *for an MCS $M = (C_1, \ldots, C_n)$ has the following form:*

$$l_0 \Leftarrow l_1, \ldots, l_m \tag{3.1}$$

for $n \geq 0$, where $l_0, l_1, \ldots, l_m$ are literals, such that and $l_i \neq (\neg) l_j$ for $i \neq j$, $0 \leq i, j \leq m$. Additionally, $\mathcal{A} = \bigcup_{i=0}^{n} \Sigma_i$, where Σ_i is the alphabet of C_i.

Comparing the adapted association rules Definition 20 with the original Definition 19, one can observe that the set of binary items I has been replaced by $\mathcal{A}$ and that l_0 is the consequent Y of the original association rule definition, but with the cardinality restricted to 1 and that $l_1, \ldots, l_m$ are X. Additionally, note that the direction of rule has been reversed.

Given an association rule r of the form 3.1, let the head of the rule be defined as $H(r) = \{l_0\}$, the body of the rule as $B(r) = \{l_1, \ldots, l_n\}$ and $at(r) = \{l_0, l_1, \ldots, l_n\}$ returns a set of all the atoms in the rule.

Given a set of association rules R and an MCS $M = (C_1, \ldots, C_n)$, where $(\Sigma_1, \ldots, \Sigma_n)$ represents the tuple of local alphabets, then the *applicable* set of association rules from R with respect to Σ_i is defined as $R|_{\Sigma_i} = \{r \in R \mid at(r) \subseteq \Sigma_i\}$. Thus, computing the applicable set of association rules for a context, would yield all the possible rules that are relevant for that certain context, i.e., *context specific filtration*.

Example 17 Consider the MCS M from the running example (Example 6) and assume that C_1 extracted association rules via a module called the association rules miner. Then, C_1 would push forward some association rules, which include:

$$\begin{array}{rcl} \neg peanuts_3 & \Leftarrow & \\ \neg nuts_1 & \Leftarrow & \\ train_1 & \Leftarrow & \neg car_1 \\ \neg car_1 & \Leftarrow & train_1 \end{array}$$

where they will be utilized by C_2 and other contexts to prune their local search space.

One of the benefits that we gain from such adaptation for the association rules is the ability to pass along clauses not only facts as described in Example 17.

After adjusting the association rules for MCS, we move to the actual association rule mining process, which is discussed in the next section.

3.3 Association Rules Harvesting

There is a variety of schools and algorithms, when it comes to association rules extraction. However, none of them is of relevance to this section as they are treated as inter-changeable black boxes. Nonetheless, we are interested in the syntax and semantics of the input and output of such procedure, as we have to ensure their compatibility with the MCS framework.

Generally, an association rule mining function requires a minimum of several parameters as its input, namely the set of transactions, the minimum support and confidence. However, as we have discussed in the previous section, the values for the confidence and minimum support are fixed. Additionally, recall the analogy between transactions and the set of (partial) belief states. Thus, the association rule mining function requires only one input parameter to operate, a set of (partial) belief states. To this end, we denote the association rule mining function as a unary function, called $mineARules$.

The function $mineARules$ has one input parameter which is a set of partial belief states restricted to a certain set of atoms $\mathcal{S}|_V$, each following Definition 11, while the output is a set of association rules R, such that each association rule follows Definition 20. The generated association rules have to be sound and have only one single literal for the head. The rules bodies can contain as many positive or negative literals as needed provided that none of them is the head of the given rule. Naturally, as the $mineARules$ function operates on a certain set of input, the signature $\mathcal{A}$ (Definition 20) has to be restricted to the some set of the atoms available in the set of partial belief states, $\mathcal{S}$.

We reduce the set of extracted association rules by restricting the set of belief states to a minimal set of atoms such that extracted association rules are only for literals that are required by other contexts. This is because the association rule extraction process is expensive in terms of time and processing power, so we eliminate the generation of association rules that are going to be ignored by other contexts.

Given an MCS $M = (C_1, \ldots, C_n)$. Let C_r be the root context for a query. Then by Definition 14, the minimum recursive import interface for M w.r.t. C_r is $V^*(r)$. This interface represents all the atoms that are required at some point by some context $C_i \in M$ such that $i \in IC(r)$. Thus, restricting the mining process to the set of atoms that appear in $V^*(r)$, where C_r is a root context of an MCS M would yield the desired reduction.

Notation: Let $\mathcal{S}$ be a set of partial belief states, and $V \subseteq \Sigma$ be a set of atoms for the MCS M. Then $\bigcup S|_V := \{\bigcup_i S_i \mid S_i \in \mathcal{S}|_V \wedge S_{i_j} \neq \epsilon\}$, denotes the set of all atoms that appear in all the partial belief states excluding the belief sets that have not been evaluated yet ($S_{i_j} \neq \epsilon$). Moreover, $\overline{S|_V} := \neg.(\Sigma|_V \setminus \bigcup S|_V)$ denotes the set of all remaining negated atoms computed by collecting the atoms in the global alphabet that do not appear in $\mathcal{S}|_V$ for non-empty belief sets, and then negating them. Thus, $\bigcup S|_V \cup \overline{S|_V}$ denotes all the possible literals from non-empty belief sets for a certain set of partial belief states.

Definition 21 *Let X be a set of literals, a be a literal such that $a \notin X$, and $\mathcal{S}$ be a set of partial belief states. Then the* support *of X w.r.t. $\mathcal{S}$, is denoted by $\sigma(X, \mathcal{S}) := |\{S \in \mathcal{S} \mid X \subseteq \bigcup S\}|$. Moreover, the* confidence *which is the estimate for $P(\{a\} \mid X)$, is denoted by $\gamma(X, a, \mathcal{S}) := \frac{\sigma(X \cup \{a\}, \mathcal{S})}{\sigma(X, \mathcal{S})}$.*

Definition 22 *Let $\mathcal{S}$ be a set of partial belief states. Then,*

$$\begin{aligned} mineARules(\mathcal{S}|_V) \quad = \quad & \{a \Leftarrow X \mid S \in \mathcal{S}|_V \wedge a \in \bigcup S|_V \cup \overline{S|_V} \wedge \\ & (X \subseteq \bigcup S|_V \cup \overline{S|_V} \setminus \{a, \neg a\}) \wedge \gamma(X, a, \mathcal{S}|_V) = 1\} \end{aligned} \quad (3.2)$$

Note that in the previous definition, the confidence level of all the rules is maintained at 1.0 by forcing $\gamma(X, a, \mathcal{S}) = 1.0$. Additionally, note that the minimum support was not part of equation 3.2, which automatically sets it to zero.

The following example presents a direct application of the association rules miner on one of the contexts in our running example.

Example 18 Consider the running example (Example 6) where C_4 computes the following set of partial belief states:

$$\mathcal{S} = \left\{ \begin{array}{l} (\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon) \\ (\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \epsilon) \\ (\epsilon, \epsilon, \epsilon, \{train_4\}, \{sooner_5\}, \epsilon) \end{array} \right\}.$$

Invoking $mineARules(\mathcal{S}|_{V^*(1)})$ would yield a total of 15 rules, which include the following:

$$R = \left\{ \begin{array}{rcl} car_4 & \Leftarrow & \neg train_4 \\ \neg train_4 & \Leftarrow & car_4 \\ \neg sooner_5 & \Leftarrow & \neg train_4 \\ \neg sooner_5 & \Leftarrow & car_4 \\ \neg car_4 & \Leftarrow & sooner_5 \\ train_4 & \Leftarrow & sooner_5 \\ train_4 & \Leftarrow & \neg car_4 \\ \neg car_4 & \Leftarrow & train_4 \\ \neg sooner_5 & \Leftarrow & car_4, \neg train_4 \end{array} \right\}.$$

Note that the association rules are only treated at the syntactic level, and do not incorporate any semantics at all.

In the sequel, we extend an existing distributed algorithm for equilibrium evaluation for multi-context systems by augmenting it with constraint pushing strategy that makes use of the association rule extraction process. We will discuss the merits, drawbacks, and investigate its full potential.

3.4 Evaluation Algorithm using Association Rules Pushing

In this section, we formalize the ideas presented in the previous sections into an algorithm called DMCS-SLIM. Our formalization is based on DMCSOPT (Algorithm 1) by Bairakdar et al. [2010a]. This is due to the assumption that the topology for a set of given context has been

optimized and that there is a given query plan for a certain root context. Additionally, we utilize the data mining techniques introduced in Section 2.3.

Abstracting from low-level implementation issues, the idea is as follows: Given a query plan that starts with a root context C_r, Π_r. We start with context C_k, and a set of association rules R, we guess for all the possible truth assignments for all the $(j : p)$ in the bodies of C_k bridge rules while complying to the relevant constraints provided by R, and store the result in partial belief state format. Afterwards, we compute C_k's local belief sets with the aid of all the applicable association rules from R with respect to C_k, and transform all the belief sets into partial belief states. In the next step, we apply a mining function to mine for all the possible association rules from these partial belief states and append them to R, giving us Q. Then, we traverse the given query plan, Π_r, by expanding the outgoing edges of that plan at each context, like in a depth-first search, until a leaf context is reached. During the traversal, R is always updated and passed on to the next context as a new Q, so that it contains all the mined association rules along the path from C_k to its invoker. A leaf context C_i simply performs the local computation of it belief states and returns the results as partial belief states to its parent, without mining for association rules.

Algorithm 3: DMCS-SLIM(c, R) at $C_k = (L_k, kb_k, br_k)$

Input: c: context identifier of a direct predecessor, R: a set of association rules for M
Data: Π_r: query plan w.r.t. starting context C_r and label v, $cache(k, R)$: cache
Output: set of accumulated partial belief states

(a) **if** $cache(k, R)$ *is not empty* **then** $\mathcal{S} := cache(k, R)$
else
(b) $\quad \mathcal{T} := \{(\epsilon, \ldots, \epsilon)\}$
$\quad$ **foreach** $i \in In(k)$ **do**
$\quad\quad \mathcal{T} := \mathsf{guess}(v(k, i), R|_{\Sigma_i}) \bowtie \mathcal{T}$
(c) $\quad \mathcal{S} := \emptyset$
$\quad$ **foreach** $T \in \mathcal{T}$ **do** $\mathcal{S} := \mathcal{S} \cup \mathsf{lsolve}(T, R|_{\Sigma_k})$
(d) $\quad$ **if** $\mathcal{S} \neq \emptyset \wedge \exists (k, i) \in E(\Pi_r)$ **then**
(e) $\quad\quad Q = R \cup mineARules(\mathcal{S}|_{V^*(r)})$
(f) $\quad\quad \mathcal{T} := \{(\epsilon, \ldots, \epsilon)\}$
$\quad\quad$ **foreach** $(k, i) \in E(\Pi_r)$ **do**
$\quad\quad\quad \mathcal{T} := \mathcal{T} \bowtie C_i.\mathsf{DMCS\text{-}SLIM}(k, Q)$
(g) $\quad\quad \mathcal{S} := \mathcal{S} \bowtie \mathcal{T}$
(h) $\quad cache(k, R) := \mathcal{S}$
(i) **if** $(c, k) \in E(\Pi_r)$ **then** $\mathcal{S} := \mathcal{S}|_{v(c,k)}$ // C_k is non-root
return $\mathcal{S}$

Algorithm 4: lsolve(S, R) at $C_k = (L_k, kb_k, br_k)$

Input: S: partial belief state, R: a set of association rules R
Output: set of locally acceptable partial belief states
$\mathbf{T} := \{T \in \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, S)\}) \mid T \models R\}$;
return $\{(S_1, , \ldots, S_{k-1}, T_k, S_{k+1}, \ldots, S_n) \mid T_k \in \mathbf{T}\}$;

The steps of Algorithm 3 can be explained as follows:

(a) Check the cache for an appropriate partial belief state;

(b) Guess for all interface $v(k, i)$ of the import neighborhood of context k with respect to the relevant set of association rules,$R|_{\Sigma_i}$, which is the standard R projected to context i. Join the all the partial belief states together;

(c) Compute local belief states given the guessed partial belief states of the neighbors, taking into consideration the set of applicable association rules, $R|_k$; this is the standard R projected to the current context, k;

(d) Check the partial belief states for consistency; in case of inconsistency there would be no partial belief states. In addition, check if it is not a leaf context. If it was indeed a leaf context, then the computation of $\mathcal{S}$ has been finalized;

(e) Apply the association rule miner function on the projected partial belief states to gather as much information as possible about the expected neighbors' models;

(f) For each neighbor to context k, that is available in the root's query plan, $E(\Pi_r)$, invoke said neighbor and request its partial belief states. Join all the results from all neighbors;

(g) Filter irrelevant models from the current set of partial belief states by joining the previously computed local models with the combined partial belief states collected from all neighbors;

(h) Store the value of $\mathcal{S}$ in the relevant $cache$ position; and

(i) In case of a non-root context, project the partial belief state to the interface variables in $v(c, k)$. Otherwise, return the full belief state unaltered.

DMCS-SLIM utilizes several subroutines. Some of them are adaptations of others, previously used in DMCSOPT and/or DMCS, while some are new:

- $cache(k, R)$: This is a binary function that accepts an integer denoting the context id k, and a set of association rules as input parameters R. This is simply a lookup table where the unique key is the pair (k, R) and stored data are the partial belief states at context C_k. If a hit is found, then there is no need for any computations. Thus, minimizing the re-computations. It behaves similarly to the unary $cache$ function in DMCSOPT.

- guess(v, R): This binary function that takes a set of interface variables v between two contexts as the first parameter and a set of relevant association rules R as the second. The result is the same as the unary guess function utilized by DMCSOPT, as it guesses all the possible truth assignments for the interface variables $V = v(k, i)$. However, the enhancement to the previous version is that it takes the set of applicable association rules into account. As such, the return value of this function is all the possible truth assignments of the interface variables while adhering to the constraints imposed by the set of relevant association rules.

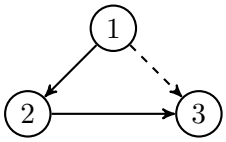


Figure 3.1: Graphical representations of Example 20

- lsolve(S, R): This is a binary function which takes a set of partial belief states $\mathcal{S}$ as a first parameter and a set of association rules R as the second and last parameter. The call of this function occurs at a certain context $C_k = (L_k, kb_k, br_k)$. The role of the function is the same as that utilized within DMCSOPT, although the other is a unary function and a less sophisticated one. The goal of this function is to supplement $\mathcal{S}$ with the augmentation of kb_k as well as all heads from bridge rules br_k such that $\mathcal{S}$ is still valid. The key difference in our approach is the extra restriction that has been placed prior to the result of the join of the augmented kb_k with the given $\mathcal{S}$, where the augmented kb_k must model all the association rules available in R. The function returns a set of partial belief states adheres to the previously mentioned limitations.
- $mineARules(\mathcal{S})$: This is a unary function which accepts a projected set of partial belief states $\mathcal{S}$ as its input. It is specific to DMCS-SLIM and has not been used in either DMCSOPT or DMCS, as it is specific to the model pushing strategy. A detailed explanation of this function input and output parameters is provided in the previous section (Section 3.3).

Consider the following excerpt from invoking DMCS-SLIM on our running example, to aid understanding our approach.

Example 19 Invoking DMCS-SLIM on C_1 from $M = (C_1, \ldots, C_6)$ in Example 6 with Π_1 as the formal representation of Figure 2.5. The initial call would be C_1.DMCS-SLIM$(0, \{\})$. When the execution flow reaches C_3, then at step (c),

$$R|_{\Sigma_3} = \left\{ \begin{array}{rcl} \neg peanuts_3 & \Leftarrow & \\ \neg coke_3 & \Leftarrow & \neg car_3 \\ train_3 & \Leftarrow & \neg car_3 \\ train_4 & \Leftarrow & \neg car_3 \\ \neg coke_3 & \Leftarrow & \neg car_4 \\ train_3 & \Leftarrow & \neg car_4 \\ train_4 & \Leftarrow & \neg car_4 \\ car_4 & \Leftarrow & \neg train_3 \\ car_3 & \Leftarrow & \neg train_3 \\ car_4 & \Leftarrow & \neg train_4 \\ car_3 & \Leftarrow & \neg train_4 \\ car_4 & \Leftarrow & \neg coke_3 \\ car_3 & \Leftarrow & \neg coke_3 \end{array} \right\}.$$

Thus, after the execution of the lsolve in the loop at step (c),

$$\mathcal{S} = \left\{ \begin{array}{l} (\epsilon, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon) \\ (\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3\}, \{train_4\}, \epsilon, \epsilon) \\ (\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3\}, \{car_4, train_4\}, \epsilon, \epsilon) \\ (\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3, urgent_3\}, \{train_4\}, \epsilon, \{sick_6\}) \\ (\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3, urgent_3\}, \{car_4, train_4\}, \epsilon, \{sick_6)\}) \end{array} \right\}.$$

Note that compared to DMCSOPT, $\mid \mathcal{S} \mid$ has been reduced from to 34 to 5.

The full trace of the running example is available in Appendix A. Next we consider a relatively small example, to highlight all the interesting execution steps.

Example 20 Consider the following MCS $M = (C_1, C_2, C_3)$, where all L_i are ASP logics and

- $kb_1 = \{a_1 \leftarrow \text{not } b_1; \quad \bot \leftarrow c_1\}$ and
  $br_1 = \{b_1 \leftarrow (2 : b_2), (3 : b_3); \quad c_1 \leftarrow (3 : c_3)\}$;
- $kb_2 = \{\bot \leftarrow \text{not } a_2, \text{not } b_2\}$ and
  $br_2 = \{a_2 \leftarrow (3 : a_3); \quad b_2 \leftarrow (3 : b_3)\}$;
- $kb_3 = \{a_3 \vee b_3 \leftarrow; \quad c_3 \vee d_3 \leftarrow b_3\}$ and
  $br_3 = \emptyset$

The representation of M's query plan is depicted in Figure 3.1, where the dashed line represents a removed edge (see Section 2.2.3).

Following a the control flow of DMCS-SLIM applied to M, we notice the following:

- (f) at C_1: after applying the mining operation, there is a total of 21 association rules, such as $\neg c_1 \Leftarrow$,$\neg c_3 \Leftarrow$ and $\neg a_1 \Leftarrow b_1$.
- (b) at C_2: we have a non-empty set of association rules. Applying the applicability relation (filtering relation) to this set with respect to C_3, we get a set of one rule only, namely $\neg c_3 \Leftarrow$. This leads to the pruning away half of the set of the partial belief states that were normally going to be computed.
- (f) at C_2: $mineARules$ generates 9 association rules. The union removes any duplicates.
- (b) at C_3: there are 3 relevant association rules, $\neg c_3 \Leftarrow$, $b_3 \Leftarrow \neg a_3$ and $a_3 \Leftarrow \neg b_3$.
- (c) at C_3: the size of the computed set of partial belief states has been reduced from 3 to 2.

Thus, the computed equilibria for M are: $S = (\{a_1\}, \emptyset, \{a_3\})$ and $T = (\{b_1\}, \{b_2\}, \{b_3\})$.

Recall that at step (d), we have a check for inconsistency that might arise during lsolve or guess. If the system is consistent, then the execution proceeds normally. Otherwise, there is no need to continue the evaluation of the remaining contexts, as the result is already available. This property is illustrated in the next example.

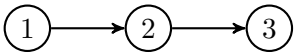


Figure 3.2: Graphical representations of Example 21

Example 21 Consider the following relatively small MCS $M = (C_1, C_2, C_3)$, whose topology is shown in Figure 3.2, where all L_i are ASP logics and

- $kb_1 = \{a_1 \leftarrow \text{not } b_1; \quad \bot \leftarrow a_1\}$ and
  $br_1 = \{b_1 \leftarrow (2 : b_2)\}$;
- $kb_2 = \{\bot \leftarrow b_2\}$ and
  $br_2 = \{a_2 \leftarrow (3 : a_3)\}$;
- $kb_3 = \{a_3 \vee b_3 \leftarrow\}$ and
  $br_3 = \emptyset$.

  This MCS is inconsistent. Using DMCSOPT, the inconsistency will not be detected until all contexts have been invoked and returned their results. However, DMCS-SLIM realizes the inconsistency earlier and terminates any further processing at step (d) in C_2. This is because C_1 generated the association rule, $b_2 \Leftarrow$ at step (e). The result of augmenting this rule during the lsolve operation at C_2, was the generation of the empty set. Thus, an inconsistency within the system have been created and caught by the algorithm at the next step of execution.

In the previous examples, we have demonstrated the effect of search space pruning during local computations. In theory, this reduces both transmission time and cost, but the overhead incurred from association rules transmission might overwhelm this reduction. Needless to say, there might be some cases where there are no benefits from using DMCS-SLIM over DMCSOPT at all. Further investigation is provided in Chapter 5.

3.5 Soundness and Completeness of DMCS-SLIM

In this section, we formally prove the soundness and a completeness of DMCS-SLIM.

We start by recalling Lemma 7 from Dao-Tran et al. [2010].

Lemma 1 (Dao-Tran et al. [2010]) *For any context C_k and partial belief state S of an MCS $M = (C_1, \ldots, C_n)$, $app(br_k, S) = app(br_k, S|_V)$ for all $V \subseteq V^*(k)$*

We do not recall the proof for this lemma, instead we refer the interested reader to the original paper.

For Lemmas 2 and 3, we let C_k and C_c be contexts of an MCS $M = (C_1, \ldots, C_n)$, where $k \in IC(c)$. Let $T(G_M) = (\mathcal{B} \cup \mathcal{C}, \mathcal{E})$ be the block tree graph of G_M, let $B_j \in \mathcal{B}$, and let $C_k \in B_j$. Additionally, let R be a set of association rules computed at C_c.

In the following lemma, we prove that the set of partial belief states computed by the guess function in DMCS-SLIM is a subset of the set of partial belief states computed by the guess function in DMCSOPT.

Lemma 2 *Let $\mathcal{S}$ be the set of belief states at C_k computed from $\bowtie_{i \in In(k)}$ guess$(v(k,i))$. Let $\mathcal{S}^{'}$ be the set of belief states at C_k computed from $\bowtie_{i \in In(k)}$ guess$(v(k,i), R|_{\Sigma_i})$. Then the computations of the guess function from DMCS-SLIM $\mathcal{S}^{'}$, are all a subset of the computations of the guess function from DMCSOPT $\mathcal{S}$, i.e., $\mathcal{S}^{'} \subseteq \mathcal{S}$.*

Proof Let $S \in \mathcal{S}$, as $\bowtie_{i \in In(k)}$ guess$(v(k,i)) = \bowtie_{i \in In(k)} 2^{(v(k,i))}$, then $S \in \bowtie_{i \in In(k)} 2^{(v(k,i))}$.

Let $S^{'} \in \mathcal{S}^{'}$, as $\bowtie_{i \in In(k)}$ guess$(v(k,i), R) = \bowtie_{i \in In(k)} 2^{(v(k,i))} \models R$, then $S^{'} \in \bowtie_{i \in In(k)} 2^{(v(k,i))} \models R$.

Thus by construction of S and $S^{'}$, if $S^{'} \in \mathcal{S}^{'}$ then $S^{'} \in \mathcal{S}$. Thus, $\mathcal{S}^{'} \subseteq \mathcal{S}$.

In the following lemma, we prove that the set of partial belief states computed by the lsolve function in DMCS-SLIM is a subset of the set of partial belief states computed by the lsolve function in DMCSOPT.

Lemma 3 *Let U be the set of belief states from $\bowtie_{i \in In(k)}$ guess$(v(k,i), R|_{\Sigma_i})$. Let $\mathcal{S} =$ lsolve(U) and $\mathcal{S}^{'} =$ lsolve(U,R) at C_k. Then the computations of the lsolve function from DMCS-SLIM $\mathcal{S}^{'}$, are all a subset of the computations of the lsolve function from DMCSOPT $\mathcal{S}$, i.e., $\mathcal{S}^{'} \subseteq \mathcal{S}$.*

Proof Let $S \in \mathcal{S}$, then by definition of lsolve(U), $S = (U_1, \ldots, U_{k-1}, T_k, U_{k+1}, \ldots, U_n)$, such that $T_k \in \mathbf{T}$, where $\mathbf{T} := \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, U)\})$.

Similarly let $S^{'} \in \mathcal{S}^{'}$, then by definition of lsolve(U,R), $S^{'} = (U_1, \ldots, U_{k-1}, T_k^{'}, U_{k+1}, \ldots, U_n)$, such that $T_k^{'} \in \mathbf{T}^{'}$, where $\mathbf{T}^{'} := \{T^{'} \in \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, U)\}) \mid T^{'} \models R\}$.

If $R = \emptyset$, then $\mathbf{T}^{'} = \mathbf{T}$, otherwise if $R \neq \emptyset$, then by construction of $\mathbf{T}^{'}$, $\mathbf{T}^{'} \subseteq \mathbf{T}$ and by extension $T_k^{'} \subseteq T_k$.

Thus by construction of S and $S^{'}$, if $S^{'} \in \mathcal{S}^{'}$ then $S^{'} \in \mathcal{S}$. Thus, $\mathcal{S}^{'} \subseteq \mathcal{S}$.

In order to prove soundness and completeness of DMCS-SLIM, we present two propositions, where Propositions 4 and 5 give us soundness and completeness, respectively.

For Propositions 4 and 5, we let C_k be a context of an MCS M, where $M = (C_1, \ldots, C_n)$. Let $T(G_M) = (\mathcal{B} \cup \mathcal{C}, \mathcal{E})$ be the block tree graph of G_M, let $B_j \in \mathcal{B}$, and let $C_k \in B_j$ such that C_k is the root context. Let Π_k be a query plan for C_k, let $V = \{p \in v(k,j) \mid (k,j) \in E(\Pi_k)\}$.

Proposition 4 *For each $S' \in C_k$.DMCS-SLIM$(0, \emptyset)$, there exists a partial equilibrium S of M w.r.t. C_k such that $S' = S|_V$.*

Proof If this is not the first round of queries for C_k, then at step (a), the $cache$ will have a hit where we retrieve the stored value of $\mathcal{S}$. Then, we jump to step (i), where for $S' \in \mathcal{S}|_V$, there exists a partial equilibrium S of M w.r.t. C_k such that $S' = S|_V$. For the remainder of the proof, we assume it is the first time to query the system.

Let $S' \in C_k.\mathsf{DMCS\text{-}SLIM}(c, R)$. We show now that there exists a partial equilibrium S of M w.r.t. C_k such that $S' = S|_V$. We proceed by structural induction on the topology of MCS M.

Base case: C_k is a leaf with $In(k) = \emptyset$, then $br_k = \emptyset$. Let C_c be the direct predecessor of C_k in Π_r, so R is the set of association rules computed at C_c. As C_k is a leaf, this means that (b) is not executed and $\mathcal{T} = \{(\epsilon, \ldots, \epsilon)\}$. At step (c), $\mathcal{S} = \emptyset$ and lsolve runs exactly once on $(\epsilon, \ldots, \epsilon)$ and R. Thus the set of all belief states $\mathcal{S} = \mathsf{lsolve}((\epsilon, \ldots, \epsilon), R) = \{(\epsilon, \ldots, \epsilon, T_k, \epsilon, \ldots, \epsilon) \mid T_k \in \{T \in ACC_k(kb_k) \mid T \models R\}\}$. By Lemma 3, if there is a partial equilibrium of M w.r.t. C_k, then $\mathcal{S}$ is a partial equilibrium for M w.r.t. C_k. At step (d), as C_k is a leaf node, so $\neg\exists(k, i) \in E(\Pi_r)$, thus the control flow jumps to step (h). We store the value of $\mathcal{S}$ in the cache under the entry for $cache(c, R)$. At step (i) we get that $S' \in \mathcal{S}|_V$. Towards a contradiction, assume that there is no partial equilibrium $S = (S_1, \ldots, S_n)$ of M w.r.t. C_k such that $S' = S|_V$. From $In(k) = \emptyset$, we get that $IC(k) = \{k\}$, thus the partial belief state $(\epsilon, \ldots, \epsilon, T_k, \epsilon, \ldots, \epsilon) \in S$ is a partial equilibrium of M w.r.t. C_k. Contradiction.

Induction step: assume that the import neighborhood of context C_k is $In(k) = \{i_1, \ldots, i_m\}$ for $m \geq 1$, Q_k is the set of association rules computed at C_k and

$$\begin{aligned} \mathcal{S}^{i_1} &= C_{i_1}.\mathsf{DMCS\text{-}SLIM}(k, Q_k), \\ &\vdots \\ \mathcal{S}^{i_m} &= C_{i_m}.\mathsf{DMCS\text{-}SLIM}(k, Q_k), \end{aligned}$$

such that for every $S'^{i_j} \in \mathcal{S}^{i_j}$, there exists a partial equilibrium S^{i_j} of M w.r.t. C_{i_j} such that $S'^{i_j} = S'^{i_j}|_V$. Let the invoker for C_k be C_c and let R_c be the set of association rules which was passed as a parameter. Following the control flow for C_k, as $In(k) = \{i_1, \ldots, i_m\}$, step (b) will be executed, which yields

$$\mathcal{T} = \bowtie_{i \in In(k)} \mathsf{guess}(v(k, i), R|_{\Sigma_i}).$$

According to Lemma 2, $\mathcal{T}$ is a set of partial belief states w.r.t. C_k. At step (c), $\mathcal{S} = \emptyset$ and lsolve is invoked several times according to the cardinality of $\mathcal{T}$.

$$\mathcal{S} = \bigcup\{\mathsf{lsolve}(S, R|_{\Sigma_k}) \mid S \in \mathcal{T}\}.$$

Thus, by Lemma 3, $\mathcal{S}$ is the set of partial belief states containing the partial equilibria of C_k. At step (d), as we have $In(k) = \{i_1, \ldots, i_m\}$, then we have a case distinction over $\mathcal{S}$:

- $\mathcal{S} = \emptyset$: this implies that C_k is inconsistent, which in turn makes M inconsistent. The control flow jumps to step (h), where $\mathcal{S}$ is stored in the cache. Afterwards, step (i) is irrelevant as the projection of $\emptyset$ yields $\emptyset$. Thus, returning $\mathcal{S}$ as the result, which would make Proposition 4 trivially holds.

- $\mathcal{S} \neq \emptyset$: Thus, there is no inconsistency so far and we step inside the if condition. We continue the proof w.r.t. this case.

At step (e), $Q_k = mineARules(\mathcal{S}|_V)$ is computed. Then step (f) takes place, which yields

$$\mathcal{T} = \mathcal{S}^{i_1} \bowtie \cdots \bowtie \mathcal{S}^{i_m}$$

as a result of calling all DMCS-SLIM at $C_{i_1}, \ldots, C_{i_m}$. Since every DMCS-SLIM at $C_{i_1}, \ldots, C_{i_m}$ returns some of its partial equilibria w.r.t. C_{i_j} projected to V, we have every $T \in \mathcal{T}$ is a partial equilibria w.r.t. C_{i_j} projected to V. At step (g) $\mathcal{S}^{=}\mathcal{S} \bowtie \mathcal{T}$. Thus, we have that $\mathcal{S}$ will simply be filtered w.r.t. to the partial equilibria in $\mathcal{T}$, as per the join operation (Definition 13). Thus, as we have visited all the contexts from $In(k)$, $\mathcal{S}$ contains the partial equilibria of M w.r.t. C_k. At step (h), We store the value of $\mathcal{S}$ in the cache under the entry for $cache(c, R)$. By Lemma 1, every $T \in \mathcal{T}$ preserves applicability of the bridge rules. Thus at step (i), we eventually get that $S' \in \mathcal{S}|_V$, and that there exists a partial equilibrium S of M w.r.t. C_k such that $S' = S|_V$.

Proposition 5 *For each partial equilibrium S of M w.r.t. C_k, there exists an $S' \in C_k$.DMCS-SLIM$(0, \emptyset)$, such that $S' = S|_V$.*

Proof If this is not the first round of queries for C_k, then at step (a), the $cache$ will have a hit where we retrieve the stored value of $\mathcal{S}$. Then, we jump to step (i), where for $S' \in \mathcal{S}|_V$, there exists a partial equilibrium S of M w.r.t. C_k such that $S' = S|_V$. For the remainder of the proof, we assume it is the first time to query the system.

Let S be a partial equilibrium of M w.r.t. C_k such that $S' = S|_V$. We show that $S' \in C_k$.DMCS-SLIM$(0, \emptyset)$. Let $C_k \in B_j$ be the root context for B_j. We a have case distinction on $In(k)$.

- $In(k) = \emptyset$: At step (b), $\mathcal{T} = \bowtie_{i \in In(k)} \mathsf{guess}(v(k,i), R|_{\Sigma_i})$. As $R = \emptyset$ and $In(k) = \emptyset$, then $\mathcal{T} = \emptyset$. This means that C_k is a leaf node and does not have any neighbors at all. At step (c), $\mathcal{S}$ should represent a superset for the partial equilibria of M w.r.t. C_k. However, as $In(k) = \emptyset$, then $br_k = \emptyset$, so $\mathcal{S}$ represents the set of all the partial equilibria of M w.r.t. C_k. Thus, $\mathcal{S} = \bigcup\{\mathsf{Isolve}(T, R|_{\Sigma_i}) \mid T \in \mathcal{T}\}$. As $R = \emptyset$, we have $\mathcal{S} = \bigcup\{\mathsf{Isolve}(T, \emptyset) \mid T \in \mathcal{T}\}$. Let $\hat{S} \in \mathcal{S}$, thus $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_{k-1}, T_k, \hat{S}_{k+1}, \ldots, \hat{S}_n)$, such that $T_k \in \mathbf{T}$, where $\mathbf{T} := \{T \in \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, S)\}) \mid T \models R\}$. Thus, $\mathbf{T} := \mathbf{ACC}_k(kb_k)\})$. As a result, for each $\hat{S} \in \mathcal{S}$, $\hat{S}$ is a partial equilibria of M w.r.t. C_k. As $In(k) = \emptyset$, then we jump to step(h) at step (d) as C_k is a leaf node. We store the value of $\mathcal{S}$ in the cache under the entry for $cache(0, \emptyset)$. Afterwards, at step (i), we just have to show that for each $S \in \mathcal{S}$, $S' = S|_V$. Since $C_k \in B_j$, $V = \{p \in v(k,j) \mid (k,j) \in E(\Pi_k)\}$, and $br_k = \emptyset$. Thus, $\mathcal{S}$ is the set of partial equilibria for C_k that will be returned by the C_k as a response to the initial call. As a result for each $S \in \mathcal{S}$, we get that $S' = S|_V$, which means that $S' \in C_k$.DMCS-SLIM$(0, \emptyset)$.

- $In(k) = \{i_1, \ldots, i_m\}$ where $m \geq 1$: At step (b), $\mathcal{T} = \bowtie_{i \in In(k)} \mathsf{guess}(v(k,i), R|_{\Sigma_i})$, which is the set of all interpretations for all the bridge rule atoms w.r.t. C_k. As $R = \emptyset$ and $In(k) = \{i_1, \ldots, i_m\}$, then $\mathcal{T} = \bowtie_{i \in In(k)} \mathsf{guess}(v(k,i), \emptyset) = \bowtie_{i \in In(k)} 2^{(v(k,i))} \models \emptyset$, then $\mathcal{T} = \bowtie_{i \in In(k)} 2^{(v(k,i))}$. We continue the proof w.r.t. this case.

At step (c), $\mathcal{S}$ should represent a superset for the partial equilibria of M w.r.t. C_k. Thus, $\mathcal{S} = \bigcup\{\mathsf{Isolve}(T, R|_{\Sigma_i}) \mid T \in \mathcal{T}\}$. As $R = \emptyset$, we have $\mathcal{S} = \bigcup\{\mathsf{Isolve}(T, \emptyset) \mid T \in \mathcal{T}\}$. Let $\hat{S} \in \mathcal{S}$,

thus $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_{k-1}, T_k, \hat{S}_{k+1}, \ldots, \hat{S}_n)$, such that $T_k \in \mathbf{T}$, where $\mathbf{T} := \{T \in \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, \hat{S})\}) \mid T \models R\}$. Thus, $\mathbf{T} := \{T \in \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, \hat{S})\}) \mid T \models \emptyset\}$. Thus, $\mathbf{T} := \mathbf{ACC}_k(kb_k \cup \{head(r) \mid r \in app(br_k, \hat{S})\})$. At this point, for some $\hat{S} \in \mathcal{S}$, it is the case that $\hat{S}|_V$ is a partial equilibrium of M w.r.t. C_k. At step (d), as we have $In(k) = \{i_1, \ldots, i_m\}$, then we have a case distinction over $\mathcal{S}$:

- $\mathcal{S} = \emptyset$: this implies that C_k is inconsistent, which in turn makes M inconsistent. Thus, Proposition 5 trivially holds.
- $\mathcal{S} \neq \emptyset$: Thus, there is no inconsistency and we step inside the if condition. We continue the proof w.r.t. this case.

At step (e), $Q_k = mineARules(\mathcal{S}|_V)$, where by the mechanism association rule mining (Definition 22), for all $r \in Q_k$ and for each $\hat{S} \in \mathcal{S}$, it is the case that for $\hat{S} = (\hat{S}_1, \ldots, \hat{S}_n)$, $\hat{S}_i \models r$. At step (f), since $In(k) = \{i_1, \ldots, i_m\}$, let the results collected from the neighbors be:

$$\begin{aligned} \mathcal{S}^{i_1} &= C_{i_1}.\textsf{DMCS-SLIM}(k, Q_k), \\ &\vdots \\ \mathcal{S}^{i_m} &= C_{i_m}.\textsf{DMCS-SLIM}(k, Q_k). \end{aligned}$$

where each $\mathcal{S}^{i_j}$ represents all the partial equilibrium for M w.r.t. C_j. Let $\bar{S}$ be S restricted to $IC(j)$ where $\bar{S} = (\bar{S}_1, \ldots, \bar{S}_n)$ such that for each $y \in IC(j)$, $\bar{S}_y = S_y$, otherwise $\bar{S}_i = \epsilon$. Towards a contradiction, assume that $\bar{S} \notin \mathcal{S}^{i_j}$.

(i) From step (c), we already established that $\mathcal{S}$ represents a superset for all the possible partial equilibria of M w.r.t. C_k, thus $\bar{S} \in \mathcal{S}$. Since $Q_k = mineARules(\mathcal{S}|_V)$, thus for each $\hat{S} \in \mathcal{S}$, it is the case that for each $r \in Q_k$, $\hat{S} \models r$. As S is a partial equilibrium of M w.r.t. C_k, then for each $r \in Q_k$, $\bar{S} \models r$.

(ii) As each $C_{i_j}.\textsf{DMCS-SLIM}(k, Q_k)$ uses $Q_k|_{\Sigma_j}$ at the lsolve and guess operations. Thus, for each $S^{i_j} \in \mathcal{S}^{i_j}$, it is the case that for each $r \in Q_k|_{\Sigma_j}$, $S^{i_j} \models r$. As $\bar{S} \notin \mathcal{S}^{i_j}$, then there exists $r \in Q_k|_{\Sigma_j}$ such that $\bar{S} \not\models r$.

Since $Q_k|_{\Sigma_j} \subseteq Q_k$, then from (i) we get that for each $r \in Q_k|_{\Sigma_j}$, $\bar{S} \models r$, which clearly contradicts with (ii) where there exists $r \in Q_k|_{\Sigma_j}$ such that $\bar{S} \not\models r$. Thus (i) and (ii) can not both hold, contradiction. Therefore, $\bar{S} \in \mathcal{S}^{i_j}$, and thus all the partial equilibria computed at any neighbor C_{i_j} for C_k, which are partial equilibria for M w.r.t. C_k, are not pruned away.

At the end of step (f), the result from joinin all the partial belief states is

$$\mathcal{T} = \mathcal{S}^{i_1} \bowtie \cdots \bowtie \mathcal{S}^{i_m},$$

where $\mathcal{T}$ represents all the partial equilibrium for M w.r.t. to all of C_k's neighbors. At step (g), the locally computed partial belief states are pruned with the aid of $\mathcal{T}$, thus $\mathcal{S}'' = \mathcal{S} \bowtie \mathcal{T}$. By the join operation (Definition 13) and the generation of each $\hat{S} \in \mathcal{S}$ (step (b) +(c)), no new partial belief states can be generated. In fact it can only be the case that either $\mathcal{S}$ does not change or is partially pruned. Thus, for each $S'' \in \mathcal{S}''$, it is the case that $S'' \in \mathcal{S}$. Thus, $\mathcal{S}'' \subseteq \mathcal{S}$. At

the next step (step (h)), we store the value of $\mathcal{S}$ in the cache under the entry for $cache(0, \emptyset)$. Afterwards, at step (i), for each $S \in \mathcal{S}''$, we have to show that $S' = S|_V$. Since $C_k \in B_j$ and $V = \{p \in v(k, j) \mid (k, j) \in E(\Pi_k)\}$, we get by Lemma 1 that the applicability of bridge rules in br_k are preserved. Thus, $\mathcal{S}''$ is the set of partial equilibria for C_k that will be returned by the C_k as a response to the initial call. As a result for each $S \in \mathcal{S}''$, we get that $S' = S|_V$, which means that $S' \in C_k.\mathsf{DMCS\text{-}SLIM}(0, \emptyset)$.

3.6 Optimizations for Association Rules Mining

In the previous sections, we have presented a generic approach for distributed MCS evaluation where local optimization is performed based on the concept of constraint pushing. Here, we present some optimization strategies that offer more control over the association rule mining operation.

3.6.1 Subsumption of Association Rules

Association rule miners work only on the syntactic level, thus it might be the case that the miner produces association rules that provide the same information as other association rules during a mining operation. The only difference between these rules is that one of them is more effective and general than the other.

Definition 23 *Let r_1 and r_2 be two association rules. Then* r_1 subsumes r_2 *iff* $H(r_1) = H(r_2)$ *and* $B(r_1) \subset B(r_2)$.

Thus in order to generate such prime implicants, we can utilize the following function.

Definition 24 *Let R be a set of association rules. Then,*

$$nonSubsumption(R) = \{r \in R \mid \textit{there is no } r' \in R \textit{ s.t. } r' \textit{ subsumes } r, \textit{ and } r \neq r'\}$$

Example 22 Consider the MCS M from the running example (Example 6). During the association rule extraction procedure at C_1, the following rules are produced:

$$R = \left\{ \begin{array}{lcl} \neg peanuts_3 & \Leftarrow & \\ \neg peanuts_3 & \Leftarrow & \neg car_1 \\ \neg peanuts_3 & \Leftarrow & train_1 \\ \neg peanuts_3 & \Leftarrow & \neg car_3 \end{array} \right\}.$$

Applying the subsumption property during extraction, leads to the following reduction:

$$R = \left\{ \begin{array}{lcl} \neg peanuts_3 & \Leftarrow & \end{array} \right\}.$$

In order to make DMCS-SLIM compatible with the subsumption checking strategy, simply replace step (e) by $Q = nonSubsumption(R \cup mineARules(\mathcal{S}|_{V^*(r)}))$.

In theory, enforcing subsumption should improve the amount of association rules exchange among contexts.

3.6.2 Association Rules Size

Adjusting the size of the generated association rules offers a significant amount of control of the extraction process. Of course, depending on the set of belief states under consideration, there is an upper limit on the size of the association rules. Otherwise the extracted association rules will start exhibiting the "bias" effect, i.e., they will contain information too much specific for current context under consideration and thus they will not contribute to other contexts and just induce an extra unnecessary overhead.

Formally, if r is an association rule, then $|r| := |at(r)|$.

Example 23 Consider the MCS M from the running example (Example 6). During the association rule extraction procedure at C_1, the following rules are produced:

- association rule size = 1 :

$$R = \left\{ \begin{array}{rcl} \neg peanuts_3 & \Leftarrow & \\ \neg nuts_1 & \Leftarrow & \end{array} \right\}.$$

- association rule size = 2:

$$R = \left\{ \begin{array}{lcl} train_3 & \Leftarrow & train_1 \\ train_2 & \Leftarrow & train_1 \end{array} \right\}.$$

- association rule size = 3:

$$R = \left\{ \begin{array}{rcl} \neg peanuts_3 & \Leftarrow & \neg train_1, \neg nuts_1 \\ \neg nuts_1 & \Leftarrow & car_1, \neg train_3 \end{array} \right\}.$$

Definition 25 *Let $\mathcal{S}$ be a set partial belief states. Let n be a natural number. Then,*

$$mineARules(\mathcal{S}, n) = \{r \in mineARules(\mathcal{S}) \mid |H(r) \cup B(r)| \leq n\}$$

In order to use DMCS-SLIM while adopting the upper threshold for association rules size technique, simply replace $mineARules(\mathcal{S}|_{V^*(r)}))$ by $mineARules(\mathcal{S}|_{V^*(r)}, n)$ and add an extra input parameter n,to DMCS-SLIM where n represents the maximum allowed association rule size.

The lower the size limit, the faster the extraction process. However, less information is pushed forward.

3.6.3 Symmetric Association Rules

As the association rule extraction process does not involve any semantical aspects, and despite the fact that the mining process produces unique association rules, it might be the case the the information portrayed by some rules be duplicated.

Definition 26 *Let r_1 and r_2 be association rules, such that $r_1 \neq r_2$. Then, r_1 and r_2 are called* symmetrical *($r_1 \hat{\equiv} r_2$) iff:*

$$\neg.B(r_1) \cup H(r_1) = \neg.B(r_2) \cup H(r_2)$$

Example 24 Consider the MCS M from the running example (Example 6). During the association rule extraction procedure at C_2, the following rules are produced:

$$R = \left\{ \begin{array}{rcl} \neg coke_3 & \Leftarrow & \neg car_3 \\ car_3 & \Leftarrow & coke_3 \end{array} \right\}.$$

Note that these two rules are symmetrical and only one of them should be considered.

Definition 27 *Let R be a set of association rules. Then,*

$$nonSymmetric(R) = \{r \in R \mid \textit{there is no } r' \in R \textit{ s.t., } r' \hat{\equiv} r\}$$

Thus given a set of association rules, one could remove all the symmetrical rules while keeping one of them at random by $filterSymmetric$ function.

Definition 28 *Let R be a set of association rules. Then $filterSymmetric(R)$ returns R', such that $R' \subseteq R$ is an asymmetric variant of R where:*

- $nonSymmetric(R')$,
- *R' is maximal with respect to $\subseteq$.*

In order to enforce the usage of the symmetrical removal strategy with DMCS-SLIM, simply replace step (e) by $Q = filterSymmetric(R \cup mineARules(\mathcal{S}|_{V^*(r)}))$.

If any two rules are symmetrical this means we are introducing redundancy. Although, no erroneous evaluations are procured, it is still the case that one is introducing an overhead both at the lsolve and inter-context transmissions levels.

3.7 Summary

In this chapter, we elaborated upon the motivation for establishing a new approach for distributed equilibrium evaluation for nonmonotonic heterogeneous multi-context systems. We argued that knowledge contained within each context is not utilized efficiently. One could extract information from such knowledge in the form of constraints, and push them forward to other contexts with the aim of pruning their search space for local evaluation.

We discussed one of the possible ways to formally capture such information: association rules. We have integrated the association rules extraction technique with the multi-context system terminology. Afterwards, we presented our new approach to optimize local evaluation by providing a distributed algorithm for equilibrium evaluation for MCS with constraint pushing. We provided an extensive explanation of the algorithm as well as concrete examples to ease the understanding. We proved the soundness and completeness of our approach.

Finally, we discussed some possible techniques that could improve the behavior of our approach, by augmenting the mining module with additional properties and characteristics.

Chapter 4

Implementation

In order to make the constraint pushing strategy to improve local evaluation for MCS practically available and to evaluate it empirically, we provide a prototypical implementation of DMCS-SLIM.

In this chapter, we first approach the system architecture on a modular level, after that we discuss how querying the system takes place. We also provide an example on how to invoke the system on an existing MCS.

The idea for this distributed system was not created from scratch. In fact, Bairakdar et al. [2010b] provided a prototypical implementation of DMCSOPT and DMCS approaches for distributed evaluation for MCS. The system was called DMCS system. We base our implementation on that system, thus enhancing it with the constraint pushing capabilities.

DMCS is a SAT-based implementation of DMCS-SLIM, DMCSOPT and DMCS restricted to ASP logics, it is written in C++ and utilizes several off-the-shelf components. The system is available at

`http://www.kr.tuwien.ac.at/research/systems/dmcs/`.

4.1 System Architecture

The DMCS system is a sophisticated system, as it is a distributed system with the capabilities of evaluating multi-context systems using different approaches (DMCS-SLIM, DMCSOPT and DMCS). It is comprised of a number of modules.

In the sequel, we list all of the utilized modules within DMCS, and how they interact with each other to yield the system architecture. Additionally, we present some of the implemented optimization strategies that are relevant for DMCS-SLIM.

4.1.1 Modules

DMCS is a distributed system, and has a client-server architecture. It has the following modules:

- **Association Rules Miner**: This is a statistical data mining module, tasked with association

rule extraction. Given a set of (partial) belief states, it transforms them into the transaction format and extracts all the possible association rules that have a confidence level of 1.0, i.e., it is the implementation of Definition 22. Conceptually, different versions of association rule miners can be utilized. Nonetheless, an off-the-shelf association rule extraction algorithm (Hahsler et al. [2011, 2005] based on GNU R 2.11.1 by R Development Core Team [2011]) is used in this module. It produces as an end result a set of association rules.

- **DMCS**: This is the core module for our implementation. Based on the desire of the user, it handles the evaluation of the multi-context system as indicated in DMCS, DMCSOPT or DMCS-SLIM. It adjusts the control flow, data flow, and modules interactions, accordingly.

- **dmcsc**: As this is a distributed system, a client/server architecture is essential. This module is considered to be the front-end of the system. It handles the querying of the multi-context system by the user.

- **dmcsm**: This module is responsible for optimizing the topological structure of the MCS. Given the location of each context in the network, their local alphabet, and their bridge rules, it builds up the topology of the system, applies optimization techniques and generates a query plan for a specific root context. It employs the decomposition and optimization techniques discussed in Bairakdar et al. [2010a]. In our implementation version, dmcsm is not actually implemented, instead we utilize a file based version of the optimized topology.

- **Loop Formula**: This module is responsible for transforming the local knowledge base and bridge rules within a certain context into a SAT theory, denoted by $\pi(C_k)$ (Section 2.2.2). It accepts the local knowledge base and bridge rules as input and produces the propositional SAT theory in DIMACS CNF format (Gebser et al. [2007a]) to match the requirements of the SAT solver.

- **SAT Solver**: Due to the restriction to ASP logics, this module is responsible for the combined functions of the lsolve and guess sub-routines. Its main input parameter is a propositional SAT theory in DIMACS format. If DMCS-SLIM evaluation technique is chosen, it accepts as another input parameter a set of association rules, where the association rules are reformatted into propositional SAT theory, and merged with the other parameter. It employs an off-the-shelf SAT solver (Clasp 1.3.7 , Gebser et al. [2007b]) and returns all the models in belief states format.

- **Network Interface**: This is a standard client-server module. It is responsible for redirecting queries to their correct destination, whether it is client/server or server/server.

4.1.2 Module Interaction

The DMCS system architecture is outlined in Figure 4.1. It has the following main components:

(i) a front-end dmcsc for querying the multi-context system.

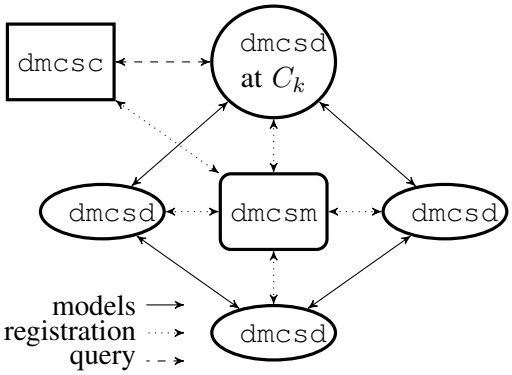


Figure 4.1: DMCS system architecture

(ii) daemons `dmcsd`, where each of them represents a context and interacts with the others. Each daemon consists of several modules, namely Loop Formula, SAT Solver, Association Rule Miner, DMCS, and Network Interface. Their interaction is illustrated in Figure 4.2.

(iii) a component `dmcsm` holding meta information about the MCS that has been collected from each context, which is basically a configuration file.

Our choice of the association rule mining algorithm was the Apriori algorithm. This stems mainly from the comparison performed on several association rule mining algorithms by Zheng et al. [2001], which showed that the lead performance on the conducted comparisons was by the Apriori algorithm given that synthetic data was utilized. Given that our benchmarking is comprised of synthetic data as well, in addition to the free-source available implementations for association rule extraction algorithms, we went ahead with aforementioned algorithm. Apriori utilizes a breadth-first search strategy to count the support of itemsets.

4.1.3 Optimization Strategies

Several optimization strategies have been incorporated into the DMCS system, all of them are related to the DMCS-SLIM technique.

- **Association Rules Subsumption**: Recall Definition 24, which only allows the extracted association rules that are not subsumed by any other association rule to be passed along. We augment the association rule miner with this additional property. However, it is only activated upon user request.

- **Association Rules Size**: This property offers control over the size of the association rules that are extracted in the association rules miner module, as described in Definition 25. Similar to the subsumption checking property, the user has full control over using this property.

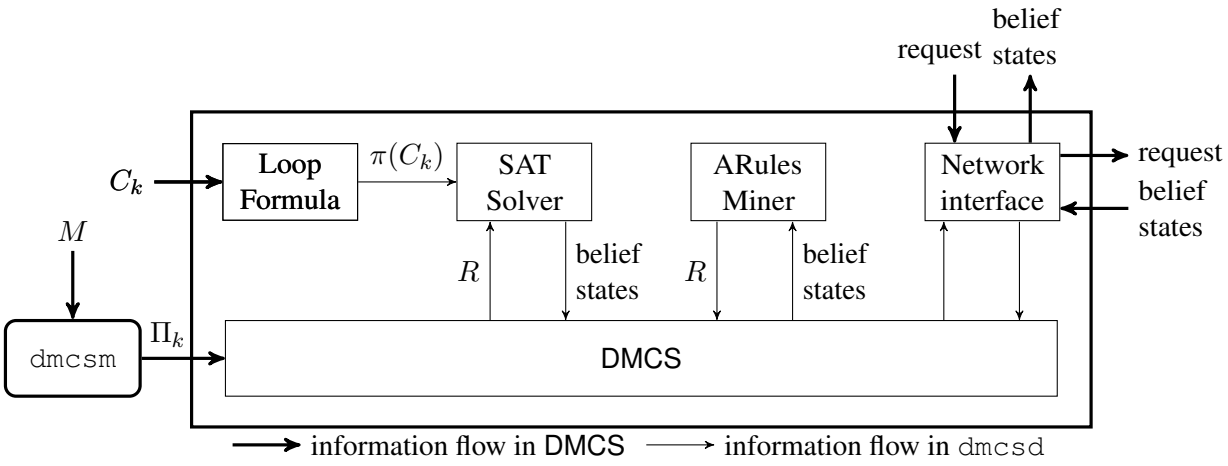


Figure 4.2: `dmcsd` architecture

- **Symmetrical Association Rules**: By activating this property, any symmetrical rules that convey the same information are eliminated as per Definition 28. As is the case with all additional properties, the user has full control on its activation.
- **Inconsistency Detection**: In addition to the inconsistency check at step (d) in DMCS-SLIM that occurs before querying the neighbors (if any), the execution halts automatically during neighbor calls, if any of the neighbors returned an inconsistent result. The rationale behind this optimization technique is as follows: if one of the neighbors for a certain context is inconsistent, there is no need to call the other neighbors and waste time with their computations, thus that context simply terminates its computation and flags the inconsistency for its invoker (if detected).

4.2 Querying the System

Any query directed to the DMCS system must go through `dmcsc`, the client. Naturally, this means that all the servers, `dmcsd`, must be up and running at that point. To this end, we explain the process of system startup and query processing.

4.2.1 System Startup

Each `dmcsd`, which represents one context in the given multi-context system, has the following set of parameters:

- `--context` represents the context id that the daemon is representing.
- `--kb` provides the context's knowledge base in a file.
- `--br` provides the context's bridge rules in a file.

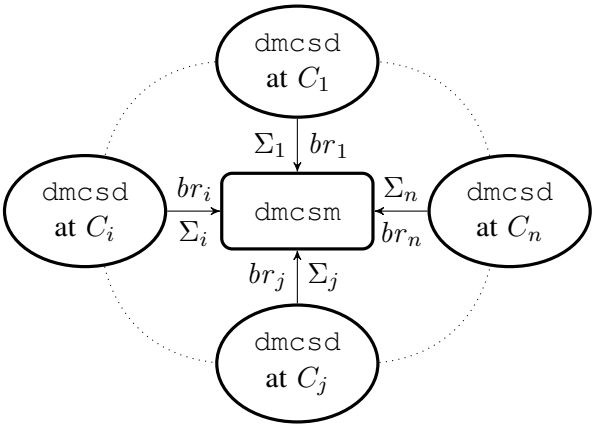


Figure 4.3: System startup

- `--manager` provides a file with the topological structure of the multi-context system. When `dmcsm` is properly implemented, the parameter will refer to its location in the form of `HOST:PORT`.

If the `dmcsm` module would have been available, then each `dmcsd` process registers at the `dmcsm`, provides its own set of bridge rules, alphabet as well as its port and host name (Figure 4.3) upon startup. With this information, the `dmcsm` component identifies the topology of the system and gets ready to answer any question regarding this meta knowledge. As `dmcsm` has not been implemented yet, this information is currently stored and accessed via file.

Upon initialization, each `dmcsd` utilizes the Loop Formula module to transform its local knowledge base and bridge rules into a SAT theory denoted by $\pi(C_k)$ in DIMACS format. Simultaneously, it activates all modules that utilize off-the-shelf components to avoid any time delay required for their startup. Then, it starts listening for incoming requests from other daemons, or from queries of `dmcsc`.

4.2.2 Processing a Query

Whenever a user wants to query the system, `dmcsc` is to be used. There are several mandatory parameters that the user has to provide in order for any query to be properly formulated.

- `--context` represents the root context that will handle the query.
- `--manager` provides a file with the topological structure of the multi-context system. When `dmcsm` is properly implemented, the parameter will refer to its location in the form of `HOST:PORT`.
- `--technique` provides the desired evaluation technique. There are only three permissible options `SLIM`, `OPT` and `STANDARD`. If none is provided, the default is `STANDARD`.

In addition to the previous set of required input parameters, there are several other optional parameters that the user can provide:

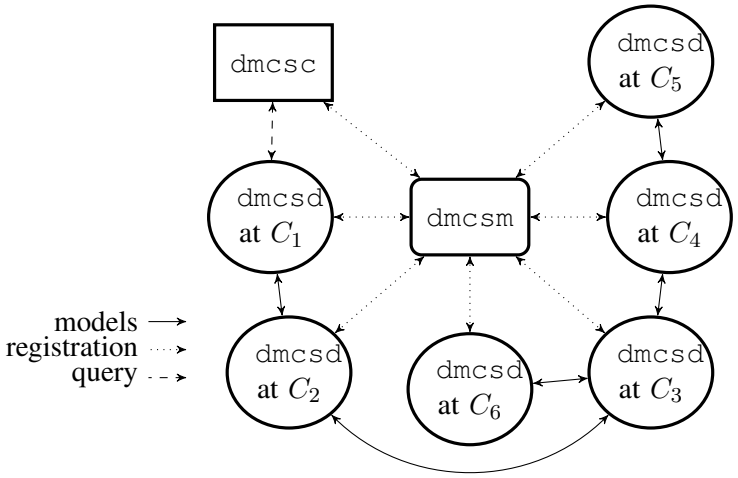


Figure 4.4: Evaluating Example 6 by DMCS

- `--query_variables` provides a set of variables of interest to evaluate.
- `--subsumption` turns the subsumption checking property on. It takes effect only when the chosen technique is `SLIM`.
- `--symmetrical` turns the symmetrical removal property on. It takes effect only when the chosen technique is `SLIM`.
- `--arules_size` provides an upper limit on the size of the association rules to be extracted. It takes effect only when the chosen technique is `SLIM`.
- `--extra_arules` provides an extra set of association rules to the root context to consider and pass along. It takes effect only when the chosen technique is `SLIM`.

When the query posed by `dmcsc` reaches the `dmcsd` representing the starting context C_k, the daemon will start its partial belief states computation depending on the chosen evaluation technique. However, the gist behind all of the evaluation techniques is the computation of the local partial belief states w.r.t. interface variables, followed by the projection of unwanted variables. If C_k needs beliefs from neighboring contexts, it sends each neighbor a request and awaits their belief states, which will be consistently combined with the local beliefs of C_k. Essentially, those requests look just as queries sent from `dmcsc`, and every `dmcsd` will process them in a uniform manner. After all neighbors have been addressed, C_k returns the partial equilibria to the client, who presents them to the user.

For a concrete usage scenario of DMCS, we consider our running example.

Example 25 Invoking DMCS on $M = (C_1, \ldots, C_6)$ from Example 6 yields the following set up Figure 4.4. We start by initializing all the daemons.

```
$ dmcsd --context=1 --kb=C1.kb --br=C1.br --manager=scientists.topology
$ dmcsd --context=2 --kb=C2.kb --br=C2.br --manager=scientists.topology
$ dmcsd --context=3 --kb=C3.kb --br=C3.br --manager=scientists.topology
$ dmcsd --context=4 --kb=C4.kb --br=C4.br --manager=scientists.topology
$ dmcsd --context=5 --kb=C5.kb --br=C5.br --manager=scientists.topology
$ dmcsd --context=6 --kb=C6.kb --br=C6.br --manager=scientists.topology
```

To compute the partial equilibria of M w.r.t. context C_1, we initiate a query via `dmcsc`. We want to find the total number of equilibria using the DMCS-SLIM approach. Additionally, we want to use the checking for subsumption property, the symmetrical removal property and we want to limit the size of the computes association rules to 3. Thus, we issue the follow query:

```
$ dmcsc --context=1 --manager=scientists.topology --technique=SLIM
--subsumption=ON --symmetrical=ON --arules_size=3
```

After `dmcsd` at C_1 finishes computation of the query, it delivers the result back to `dmcsc`. A list of the partial equilibria is then enumerated to the user:

```
( {train1,train2,train3,train4}, {car1,car3,car4} )
Total Number of Equilibria: 2
```

Chapter 5

Evaluation

In the previous chapters, we provided a new approach for distributed equilibrium computation for heterogeneous nonmonotonic multi-context systems. In this chapter, we evaluate DMCS-SLIM by empirically evaluating the experimental results. We consider DMCS-SLIM with respect to different runtime parameters, and compare it to the other mentioned distributed algorithms, namely DMCSOPT and DMCS.

In the sequel, we start by describing the setup of the system and then we provide the details of the experiments. Afterwards, we interpret the experimental results and finally present a brief summary of our findings.

5.1 System Setup

Establishing a non-biased benchmark that captures the provides a neutral point of view for understanding the strengths and weaknesses of one's approach is a tricky subject. To this end, we present several setups that we utilized in evaluating our approach.

In this section, we discuss the structure of the knowledge bases and bridge rules for each context. Afterwards, we illustrate the utilized query plans. Finally, we recall the technical aspects of our benchmark host.

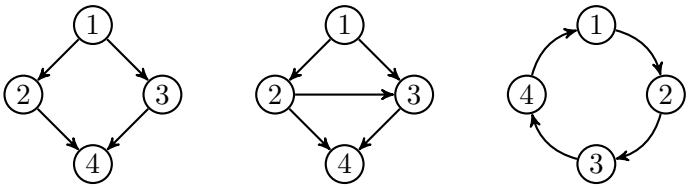
5.1.1 Intra-Contexts

Given an MCS $M = (C_1, \ldots, C_n)$, Definition 7 states that each context is characterized by three components, $C_i = (L_i, kb_i, br_i)$. Restricting each L_i to ASP Logic, leaves two components as parameters.

We utilize the parameter setting (n, s, b, r, p), which specifies

(i) the number n of contexts,

(ii) the local alphabet size $|\Sigma_i| = s$, where each C_i has a random ASP program on s atoms with 2^k answer sets, $0 \leq k \leq s/2$. Let a_i, b_i and $c_i \in \Sigma_i$, then the schema for each ASP program is: $a_i \leftarrow \text{not } b_i$, and there is a 50% chance of having $b_i \leftarrow \text{not } a_i$ or $b_i \leftarrow \text{not } c_i$.



(a) Diamond (b) Zig-Zag (c) Ring

Figure 5.1: Topologies

(iii) the maximum interface size b (number of atoms exported), and

(iv) the maximum number r of bridge rules per context, each having at most 2 body literals,

(v) $p \in [0, 1]$ is the probability of adding random constraints to the random ASP program. Each constraint can contain up to two literals.

5.1.2 Topologies

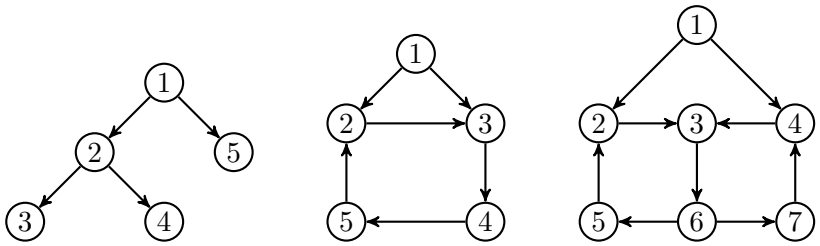
The topological representation of any MCS is an interesting factor for equilibrium evaluation via the query plan. Consequently, we consider the same topologies used in Dao-Tran et al. [2010], Bairakdar et al. [2010a], and introduce a new one. We present the structure of the topology as well as a discussion of its optimization strategies:

- Diamond, D: A diamond topology is represented in 5.1a. A stack of diamonds is a combination of multiple diamonds in a row, in other words, stacking m diamonds in a tower of $3m + 1$ contexts. Applying the decomposition techniques, we notice that we can not remove any edges, as the topologies are equal to their transitive reductions. However, the import interface at each sub-diamond is refined as every fourth context is a cut vertex, thus the partial belief states eventually returned by the root context just contain entries for the first four contexts.

- Zig-Zag, Z: A zig-zag diamond is an ordinary diamond with a connection between the two middle contexts, as depicted in Figure 5.1b. Similarly to diamond, a stack of diamonds is a combination of multiple diamonds in a row. The block optimization techniques remove two edges per block to obtain the transitive reduction and update the recursive import interface accordingly.

- Ring, R: A ring topology is a cyclic topology that contains only one cycle as depicted in Figure 5.1c. Applying the decomposition techniques, we realize in the query plan that we only removed the last edge closing the cycle to context C_1. As the resulting topology is a spanning tree, the refinement of the import interface is restricted to neighboring contexts including the import interface of the removed edge.

- Binary Tree, B: Binary trees grow balanced, i.e., every level is complete except for the last level, which grows from the left-most context (Figure 5.2a). Similar to the diamond, no



(a) Tree (b) House (c) Mongolian tent

Figure 5.2: Topologies cont.

edges are removed in the query plan generation. However, the refinement of the import interface is more drastic, as every non-leaf context is a cut vertex, and the import interfaces are restricted to those between two neighboring contexts.

- House, H: A house consists of 5 nodes with 6 edges. The ridge context has directed edges to the two middle contexts which form, with the two base contexts, a cycle with 4 edges (Figure 5.2b). House stacks are subsequently built up by using the basement nodes as ridges for the next houses (thus, m houses have $4m + 1$ contexts). Applying the decomposition techniques, one notices that house stacks have a triangle as a roof and a ring as walls, thus two connections are pruned which results in chains of contexts. As the two basement contexts are cut vertices, the final belief states contain only belief sets for the 5 contexts in the top-most house.

- Mongolian Tent, M: Let P_m and P_n be two path graphs. Formally, a mongolian tent graph is defined as the graph obtained from the graph Cartesian product $P_m \times P_n$ for an odd n by adding an extra vertex above the graph and joining every other vertex of the top row to the additional vertex. We increase the cyclicity of the graph by enforcing the edges directionality, where each 4 contexts located at the intersection of rows i and $i + 1$ with columns j and $j + 1$ form a cycle, as depicted in Figure 5.2c. Applying the decomposition techniques, one notices that the monogolian tent is comprised of one block only. Thus, there exists no cut vertices. Transitive reduction and cycle breaking techniques leads to the removal of all the edges from the root context except the first one, as well as the removal of at least one edge per each 4 contexts (2 contexts lie consecutively on the same row and their counter parts on the next row). The resulting graph is an unbalanced binary tree, where the recursive import interface is updated accordingly.

5.1.3 Experimental Setup

Our evaluation is based on the C++ implementation of DMCS system described in Chapter 4. The host system was using an Intel Xeon 3.00GHz quad-core processor with 16GB RAM and running Ubuntu 10.10 as the operating system.

As for the off-the-shelf solvers we used:

- SAT solver: We used *clasp* 1.3.7 as a SAT solver, Gebser et al. [2007a].

- association rule miner: We used the association rule extraction algorithm Hahsler et al. [2011, 2005] based on GNU R by R Development Core Team [2011], version 2.11.1 (2010-5-31), available at
  `http://r-forge.r-project.org/projects/arules/`.

Additionally, we set a timeout for the running processes to 200 seconds and 1000 MB as a memory-limit.

The randomly generated test instances were based on the following set of parameters (n, s, b, r, p), where n, s, b, r, p have the same functionality as described in Section 5.1.1.

DMCS-SLIM was evaluated using the standard algorithm as well as some of the additional properties introduced in Section 3.6, namely subsumption, association rules size and removing symmetrical association rules.

5.2 Experiments

We ran an exhaustive set of benchmarks under the experimental setup described in Section 5.1. Based on some initial testing while varying all the experimental parameters, we decided that for the parameter tuple $P = (n, s, b, r, p)$, we vary the following variables accordingly:

- n was chosen based on the topology, T, under consideration. As $T \in \{D, Z, R, B, H, M\}$, then for:
  - D: $n \in \{10, 13, 25\}$.
  - Z: $n \in \{10, 13, 49\}$.
  - R: $n \in \{10, 13, 50\}$.
  - T: $n \in \{20, 35, 50\}$.
  - H: $n \in \{9, 13, 41\}$.
  - M: $n \in \{7, 10, 16\}$.
- we fixed the variables for s, b, r to either $7, 3, 3$ or $10, 5, 5$, respectively.
- p was either 0 or 0.2. This is because it was noticed that increasing the value for p greatly increases the chance of inconsistency even for relatively small systems.

Thus, for each topology there were a total of 12 different possible parameter settings. For each setting, we created 10 random test instances, where the distributed algorithms were invoked on each set for empirical data evaluation. In total we ran over 700 experiments.

Based on preliminary results from initial testing, we restricted the investigation of DMCS-SLIM to the size of the association rules and the subsumption property. Thus DMCS-SLIM$_{a,subs}$ denotes invoking DMCS-SLIM while limiting the size of the association rules to a and only applying subsumption $subs$ was available.

- DMCS-SLIM$_1$: DMCS-SLIM is used while fixing the size of extracted association rules to 1, and without using the subsumption property. Note that activating the subsumption property while restricting the size of the association rules to 1 is redundant, as no fact subsumes another fact.
- DMCS-SLIM$_2$: DMCS-SLIM is invoked while fixing the size of extracted association rules to 2, and without using the subsumption property.
- DMCS-SLIM$_{2,subs}$: With this setting, DMCS-SLIM is using the subsumption property in addition to limiting the size of extracted association rules to 2.
- DMCS-SLIM$_3$: DMCS-SLIM extracts association rules with a maximum size of 3 under this setting. It does not utilize the subsumption property.
- DMCS-SLIM$_{3,subs}$: DMCS-SLIM is invoked while fixing the size of extracted association rules to 3, and it uses the subsumption property as well.

Additionally, we decided to remove the association rules symmetrical removal property from the investigation as its affect on the performance was negligible. This could be attributed to the nature of the randomly generated knowledge base and bridge rules as well as the implementation details of this property.

In the sequel, we report our observations on the gathered data. We present a sample of gathered statistical data in a tabular format in the next section.

5.3 Observations and Interpretations

Evaluating the results collected from the benchmarking, led to the conclusion that there are several issues that should be investigated in depth.

A sample of the experimental results are displayed in Tables 5.1 and 5.2. The results are data set from the parameter setting $P = (n, 10, 5, 5, 0)$. Each subtable $X \in \{R, B, D, Z, H, M\}$ shows runs with one of the predetermined number of contexts and corresponds to a benchmark topology from Section 5.1.2. Each row displays the average total running over ten generated instances, Σ, for one of the different runtime parameters of DMCS-SLIM or DMCSOPT. For each context in the instances the column x, for $x \in \{\phi, \phi_{\#}, \bowtie, \leftrightarrow, F, F_{\#}, E, E_{\#}, S, S_{\#}\}$ shows the average over the

- total time spent in the SAT solver (ϕ),
- total number of partial belief states computed at the SAT solver ($\phi_{\#}$),
- total time needed to join the belief states ($\bowtie$),
- total time used to transfer the belief states between the contexts ($\leftrightarrow$),
- total time spent for association rules filtration (F),
- total number of context specific association rules ($F_{\#}$),

	n		ϕ	$\phi_\#$	$\bowtie$	$\leftrightarrow$	F	$F_\#$	E	$E_\#$	S	$S_\#$	Σ (σ)	# (σ)
R	13	DMCS-SLIM$_1$	0.08	411.13	0.0	0.0	0.0	0.38	0.88	8.5	0.0	0.0	1.01 (0.03)	5.13 (4.58)
		DMCS-SLIM$_2$	0.08	399.63	0.0	0.0	0.0	4.38	0.93	155.75	0.0	0.0	1.06 (0.02)	5.13 (4.58)
		DMCS-SLIM$_{2,subs}$	0.08	399.63	0.0	0.0	0.0	2.88	0.94	155.75	0.0	75.13	1.06 (0.02)	5.13 (4.58)
		DMCS-SLIM$_3$	0.08	399.63	0.0	0.0	0.01	10.88	1.08	1025.0	0.0	0.0	1.3 (0.11)	5.13 (4.58)
		DMCS-SLIM$_{3,subs}$	0.08	399.63	0.0	0.0	0.0	2.88	1.08	1025.0	0.02	902.25	1.24 (0.06)	5.13 (4.58)
		DMCSOPT	0.07	412.63	0.0	0.0							0.1 (0.02)	5.13 (4.58)
B	20	DMCS-SLIM$_1$	1.04	698.5	0.0	0.0	0.0	0.0	6.01	5.88	0.0	0.0	7.25 (4.21)	13.25 (5.95)
		DMCS-SLIM$_2$	1.14	698.5	0.0	0.0	0.0	0.0	6.6	162.38	0.0	0.0	7.98 (4.62)	13.25 (5.95)
		DMCS-SLIM$_{2,subs}$	1.2	698.5	0.0	0.0	0.0	0.0	6.95	162.38	0.0	83.0	8.44 (3.97)	13.25 (5.95)
		DMCS-SLIM$_3$	1.27	698.5	0.0	0.02	0.15	0.0	9.49	1980.75	0.0	0.0	12.0 (5.61)	13.25 (5.95)
		DMCS-SLIM$_{3,subs}$	1.07	698.5	0.0	0.01	0.0	0.0	8.77	1980.75	1.79	1808.12	11.98 (6.9)	13.25 (5.95)
		DMCSOPT	0.14	698.5	0.0	0.0							0.19 (0.03)	13.25 (5.95)
D	13	DMCS-SLIM$_1$	0.76	3968.38	0.0	0.0	0.0	0.0	4.02	35.38	0.0	0.0	4.93 (1.35)	23.75 (13.46)
		DMCS-SLIM$_2$	0.74	3968.38	0.0	0.0	0.01	0.0	4.22	654.88	0.0	0.0	5.18 (1.43)	23.75 (13.46)
		DMCS-SLIM$_{2,subs}$	0.73	3968.38	0.0	0.0	0.0	0.0	4.29	654.88	0.01	428.38	5.22 (1.45)	23.75 (13.46)
		DMCS-SLIM$_3$	0.75	3968.38	0.0	0.02	0.14	0.0	5.4	5371.12	0.0	0.0	7.75 (1.88)	23.75 (13.46)
		DMCS-SLIM$_{3,subs}$	0.73	3968.38	0.0	0.0	0.0	0.0	5.43	5371.12	0.3	4983.38	6.82 (1.96)	23.75 (13.46)
		DMCSOPT	0.7	3968.38	0.0	0.0							0.82 (0.19)	23.75 (13.46)

Table 5.1: Runtimes for $P = (n, 10, 5, 5, 0)$ DMCS-SLIM and DMCSOPT for all 6 topologies

	n		ϕ	$\phi_\#$	$\bowtie$	$\leftrightarrow$	F	$F_\#$	E	$E_\#$	S	$S_\#$	$\Sigma\,(\sigma)$	$\#\,(\sigma)$
Z	13	DMCS-SLIM$_1$	0.6	3507.25	0.0	0.0	0.0	0.0	19.36	7.63	0.0	0.0	20.23 (27.17)	29.0 (32.9)
		DMCS-SLIM$_2$	0.6	3507.25	0.0	0.0	0.0	0.0	19.62	234.25	0.0	0.0	20.51 (27.21)	29.0 (32.9)
		DMCS-SLIM$_{2,subs}$	0.61	3507.25	0.0	0.0	0.0	0.0	20.03	234.25	0.0	127.0	20.94 (28.47)	29.0 (32.9)
		DMCS-SLIM$_3$	0.63	3507.25	0.0	0.01	0.03	0.0	21.34	2766.38	0.0	0.0	22.59 (29.03)	29.0 (32.9)
		DMCS-SLIM$_{3,subs}$	0.63	3507.25	0.0	0.0	0.0	0.0	20.52	2766.38	0.25	2587.75	21.7 (28.36)	29.0 (32.9)
		DMCSOPT	0.6	3507.25	0.0	0.0							0.87 (0.68)	29.0 (32.9)
H	13	DMCS-SLIM$_1$	0.13	759.13	0.0	0.0	0.0	0.63	2.45	8.63	0.0	0.0	2.63 (1.83)	0.0 (0.0)
		DMCS-SLIM$_2$	0.15	755.63	0.0	0.0	0.0	5.5	2.58	224.25	0.0	0.0	2.79 (1.93)	0.0 (0.0)
		DMCS-SLIM$_{2,subs}$	0.14	755.63	0.0	0.0	0.0	3.25	2.59	224.25	0.0	145.0	2.78 (1.93)	0.0 (0.0)
		DMCS-SLIM$_3$	0.14	755.63	0.0	0.01	0.02	11.88	3.29	2619.5	0.0	0.0	3.76 (2.22)	0.0 (0.0)
		DMCS-SLIM$_{3,subs}$	0.14	755.63	0.0	0.0	0.0	3.25	3.12	2619.5	0.41	2475.75	3.83 (2.34)	0.0 (0.0)
		DMCSOPT	0.15	912.0	0.0	0.0							0.21 (0.07)	0.0 (0.0)
M	7	DMCS-SLIM$_1$	0.14	749.25	0.03	0.01	0.0	2.13	2.55	7.25	0.0	0.0	4.13 (2.99)	331.75 (367.12)
		DMCS-SLIM$_2$	0.13	709.75	0.02	0.0	0.0	22.25	2.65	234.25	0.0	0.0	2.91 (1.81)	331.75 (367.12)
		DMCS-SLIM$_{2,subs}$	0.16	709.75	0.02	0.0	0.0	11.0	2.7	234.25	0.0	150.75	3.08 (1.86)	331.75 (367.12)
		DMCS-SLIM$_3$	0.13	709.75	0.02	0.01	0.03	67.13	3.18	3241.12	0.0	0.0	3.67 (1.66)	331.75 (367.12)
		DMCS-SLIM$_{3,subs}$	0.16	709.75	0.02	0.0	0.0	11.0	3.28	3241.12	0.79	3126.62	4.48 (1.61)	331.75 (367.12)
		DMCSOPT	0.12	773.43	0.03	0.02							2.32 (2.92)	214.57 (170.55)

Table 5.2: Runtimes for $P = (n, 10, 5, 5, 0)$ DMCS-SLIM and DMCSOPT for all 6 topologies cont.

- total time spent for association rule extraction (E),
- total number of extracted association rules ($E_{\#}$),
- total time spent for checking the association rules for subsumption (S), and
- total number association rules that have been removed due to the subsumption property ($S_{\#}$).

The $\#$ columns show the median of numbers of projected partial equilibria computed at C_1, which is initiated by querying C_1 with a fixed query plan Π_1. Entries in parenthesis (σ) show the standard deviation of Σ and $\#$. Note that only the first four columns and the last two columns are relevant to DMCSOPT.

In the sequel, we evaluate DMCS-SLIM against itself, to check the effect of certain properties. Afterwards, we compare the DMCS-SLIM runtime parameter that yielded the best performance against the other distributed approaches; namely DMCSOPT and DMCS.

5.3.1 Subsumption

Theoretically, the enforcement of the subsumption property for the association rules at each context should dramatically decrease the number of extracted association rules. In fact, this was exactly the case as evident in Figure 5.3. One notices that the reduction of the number of extracted association rules is always apparent, regardless of the topology under consideration. When DMCS-SLIM checks that all the extracted rules satisfy the subsumption property, this reduces the number of available association rules for inter-context transmission by at least 50% and up to 95% in some cases.

Utilizing such property involves some costs, as depicted in Figure 5.4. One can see that the overall time required to discover the partial equilibria of the system increases whenever the subsumption check is enforced. The temporal increase can range from just 5% up to 25% w.r.t. the extraction time. On the other hand, not utilizing the subsumption check increases the time required for association rules filtration about 2% as well as the transfer time between contexts.

Thus, one may infer that given a proper network setup, one can ignore the need to enforce the subsumption property among the extracted association rules. Otherwise, if it is the case that network resources are scarce and expensive, it pays off to enforce the subsumption check.

5.3.2 Limiting the Size of Association Rules

During the mining process, increasing the threshold imposed on the maximum size of association rules to be extracted increases their number exponentially, as depicted in Figure 5.5.

It is only natural that the higher the number of available association rules, the more information is available at each context. Thus, the higher the probability to improve the local solving at each context with respect to the number of locally computed belief states. Figure 5.6, shows that the number of belief states computed at the SAT solver decreases when the threshold for the size of association rules increases. However, the improvement is sensibly visible in case of going from size 1 to 2, and it reaches an average of up to 5%. According to the benchmarking, any

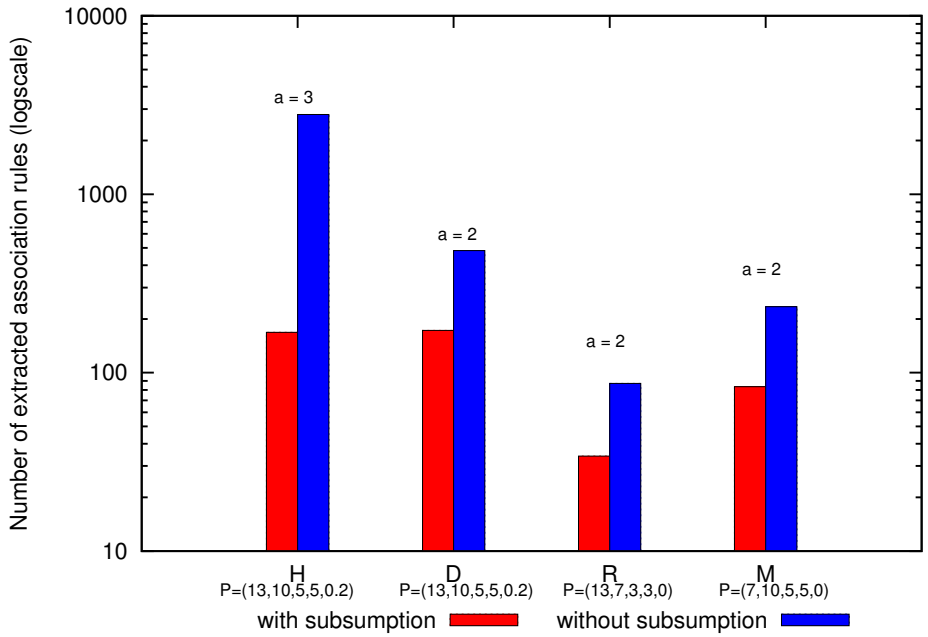


Figure 5.3: Number of extracted association rules at DMCS-SLIM with and without subsumption

association rule size bigger than two does not yield any significant improvement for the local solving process. In theory, a fixed point is reached with respect to the improvement in local solving versus the number of extracted association rules. This is a direct result of the fact that the bigger the rules get, the more specific they become as they will only represent the context that they were extracted from and will not contribute any information to others. Given the structure of our knowledge bases and bridge rules, the optimum was reached at size two.

The overhead incurred from increasing the size limit for the extracted association rules is depicted in Figure 5.7. The increase in the extraction time can reach up to 25% for association rules with size 3 w.r.t. those of size 1. However, the extra time for size 2 is only up to 5% w.r.t. size 1.

Given the experiments performed on the limit on the size of association rules that will be extracted, one may infer that the optimum size of the upper threshold is 2 literals per rule, as it does not vary much with respect to needed time when compared to size 1. Additionally, it is almost always the case that the optimum regarding improvement in the local solving is reached when the size is set to 2.

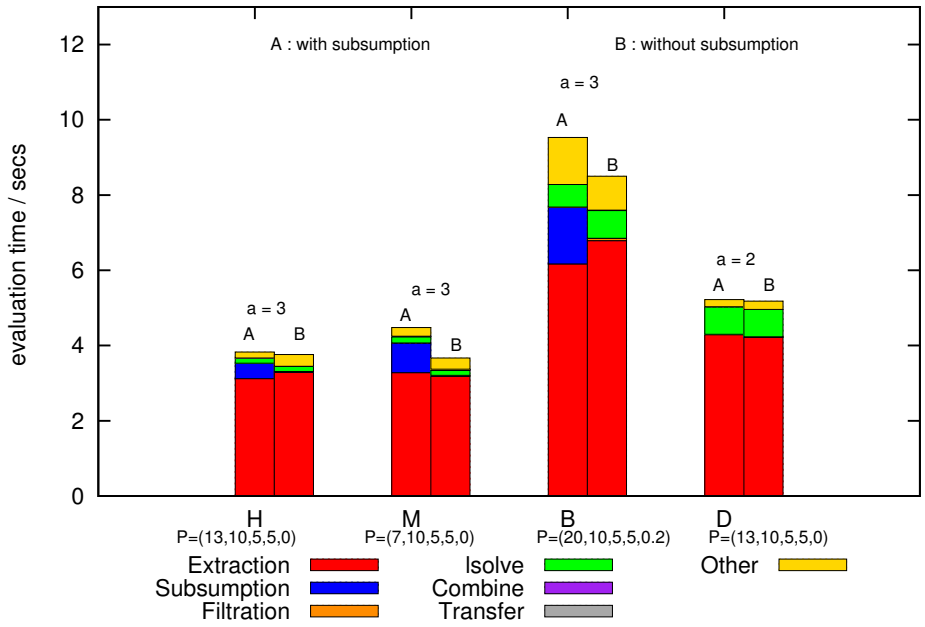


Figure 5.4: Total evaluation time for DMCS-SLIM with and without subsumption

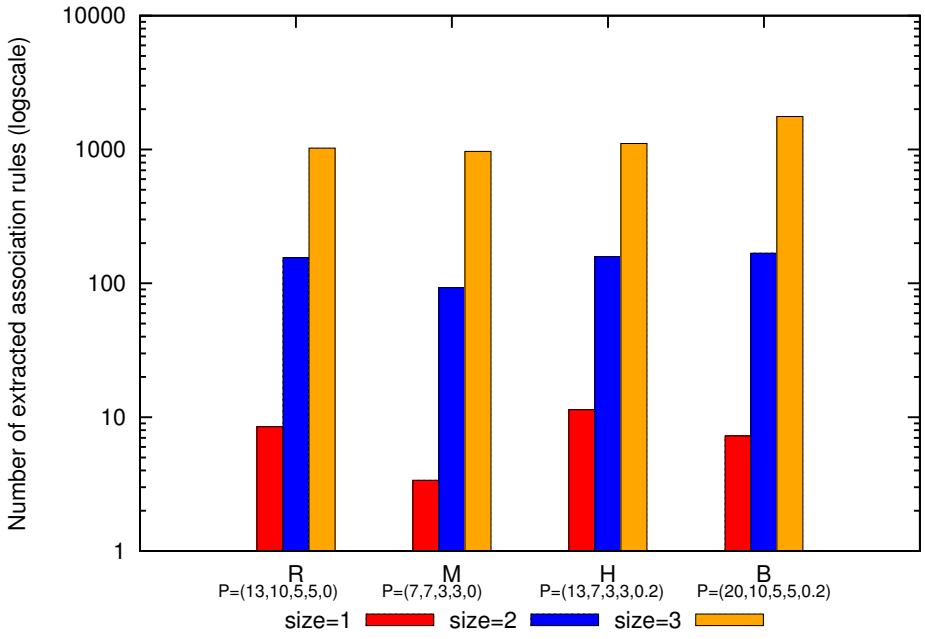


Figure 5.5: Number of extracted association rules at DMCS-SLIM for different sizes

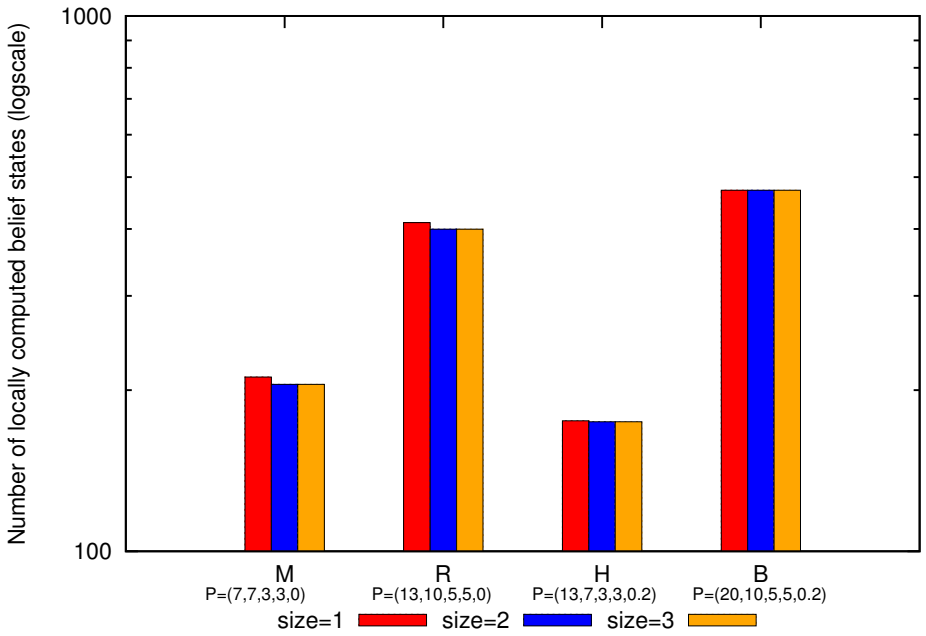


Figure 5.6: Number of locally computed partial belief states at DMCS-SLIM with different sizes

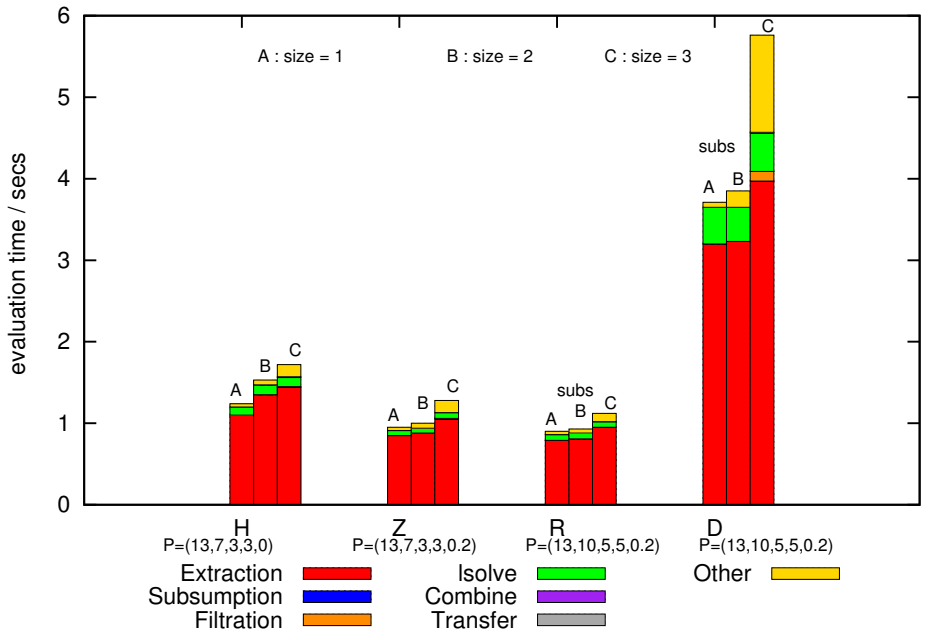
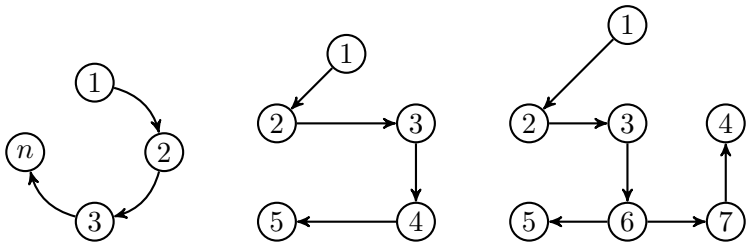


Figure 5.7: Total evaluation time for DMCS-SLIM with different sizes



(a) Ring (b) House (c) Mongolian tent

Figure 5.8: Cyclic topologies after decomposition

5.3.3 Effects of MCS Topology

It has been well established that the topological structure of the multi-context system greatly influences the evaluation algorithm. Recall that DMCS-SLIM operates on a query plan that is generated after applying some decomposition techniques that break cycles, apply decomposition techniques and update the interface variables.

Recall that parameter p controls the probability of adding random constraints to the knowledge base. In theory, these extra constraints reduce the number of equilibria for the multi-context system as they automatically force certain values for some literals.

We noticed during the experimentation that when $p = 0$, the improvement with respect to the number of locally computed belief states is always present in case of cyclic topologies and almost non-existent in the rest. However, in case of $p = 0.2$, the pruning of the local search space is visible for all topologies.

There are only three topologies in our benchmark that contain any kind of cycle. The fixed query plans generated for these topologies are depicted in Figure 5.8.

Consider the case where $p = 0$ and the ring topology R_n, where the fixed query plan is illustrated in Figure 5.8a. As a result of the applied decomposition techniques, the cycle has been broken and the last edge has been removed; namely between C_n and C_1. Consequently, the interface variables for all the other contexts have been updated to compensate for the removed edge. Consider applying DMCS-SLIM to such multi-context system. Then at each context, except the last, the process of extracting the association rules takes place. Given the randomness nature of our knowledge base and bridge rules, at C_i, the generated association rules will most likely provide information regarding C_i itself and no information for C_{i+1} at all. However, in case of C_1, the extracted association rules contain information that will be utilized by C_n, as C_n had originally an edge connecting to C_1.

This behavior is also apparent in the house topology, specifically at C_5 (Figure 5.8b) and in the mongolian tent topology at C_4 and C_5 Figure (5.8c). The sample results presented in Tables 5.1 and 5.2 show that in case of diamond, zig-zag diamond or tree topology, there was no improvement over the number of extracted association rules during the local solving. However, in very rare cases which are also attributed to the randomness of the knowledge bases and bridge rules, it was the case that acyclic topologies exhibited an improvement in the local solving during evaluation with DMCS-SLIM.

On the other hand, when $p = 0.2$, the randomly added constraints affect the results of the local solving regardless of the adopted topological structure.

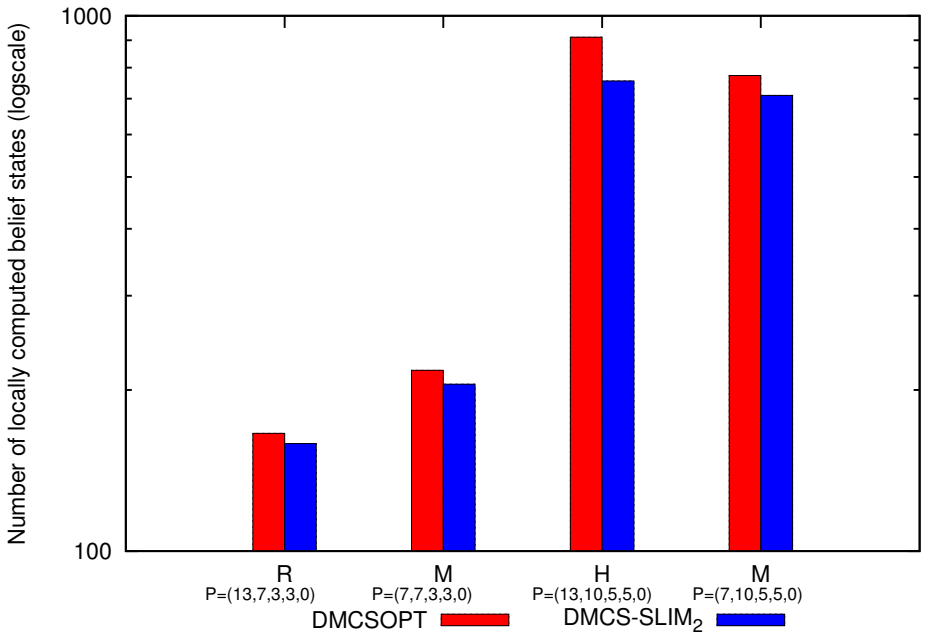


Figure 5.9: Number of extracted association rules DMCSOPT vs. DMCS-SLIM$_2$

5.3.4 DMCSOPT vs. DMCS-SLIM

Up to this point, we have investigated the behavior of DMCS-SLIM under different conditions. In the sequel, we compare the behavior of DMCS-SLIM with respect to DMCSOPT. We refrain from providing a full comparison between DMCS and DMCS-SLIM as initial tests revealed that DMCS-SLIM is on a different level than DMCS with respect to scalability and that it should be compared by its direct predecessor as they share a common ground, namely the decomposition techniques that yield the query plan.

In our comparison we utilized DMCS-SLIM$_2$, as we have already established that the best performance for DMCS-SLIM was achieved while enforcing that the upper limit for the size of the association rule to be extracted was 2. Additionally, we ignore the subsumption checking property as the benchmarking was performed on one machine so the networking overhead incurred from the extra association rules transmission is insignificant compared to the amount of time invested in enforcing the subsumption property.

Recall, the connection that we established in Section 5.3.3 between the parameter p for the test instances and the topology based improvement. We adhere to the same case distinction during this comparison.

For $p = 0$, a graphical representation of the number of belief states computed at local solving with DMCS-SLIM$_2$ and DMCSOPT is displayed in Figure 5.9. We observed through all of our benchmark results that DMCS-SLIM$_2$ achieved a reduction rate on the search space up to 20% when compared to DMCSOPT w.r.t. the results of the SAT solver utilizing cyclic topologies. For acyclic topologies, it was also the case that there were no improvements at all, as the number of

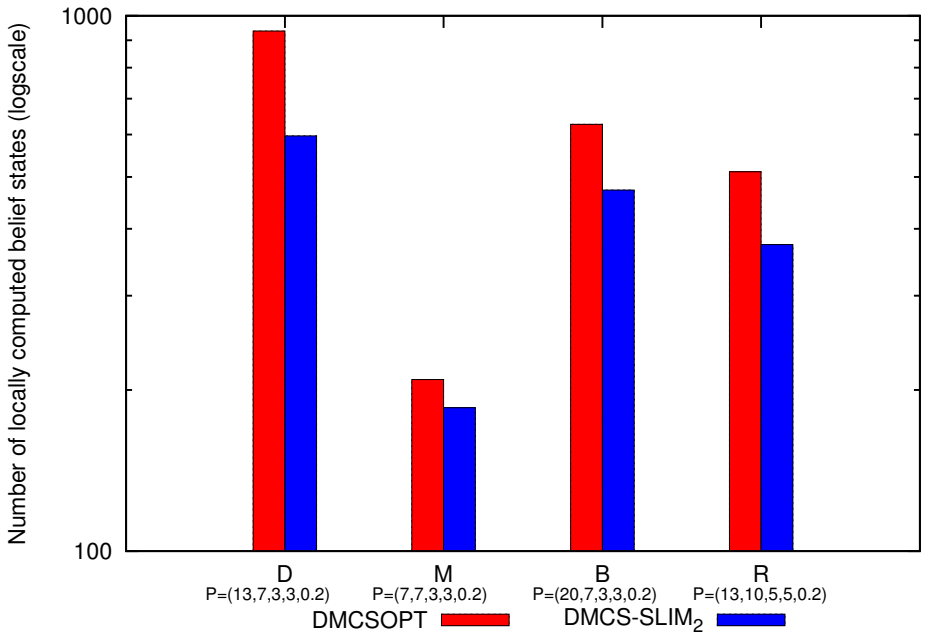


Figure 5.10: Number of extracted association rules DMCSOPT vs. DMCS-SLIM$_2$ cont.

computed belief states stayed the same. Additionally, there were some cases where the acyclic topologies showed a search space reduction of 2%, which can be attributed to the randomness nature of the intra-context data generation.

On the other hand, for $p = 0.2$, DMCS-SLIM$_2$ showed an improvement over DMCSOPT for all the topologies, a graphical comparison is depicted in Figure 5.10. The search space reduction rate on the number of computed belief states reached 40% with DMCS-SLIM$_2$ compared to DMCSOPT. We also observed that the introduction of random constraints might lead to inconsistent multi-context systems. In fact, with relatively high number of contexts, ≥ 40, it is almost guaranteed that the randomly generated multi-context system will be inconsistent. Similar to $p = 0$, it was some times the case that a very small reduction is apparent which only reached the neighborhood of 3%; even though this was rare.

Unfortunately, the overhead induced by DMCS-SLIM$_2$ is relatively high compared to DMCSOPT as shown in Figure 5.11. The total time utilized by both systems is presented as one bar in a histogram where each colored block represents a certain part the DMCS system. One can easily spot that the bottleneck is the extraction time. As the time block entails, the extraction time represents the amount of time invested by the DMCS system in the association rule extraction process.

In attempt to improve the total time needed to evaluate the system with DMCS-SLIM, we proceeded by testing the external off-the-shelf association rule miner separately. We found a relatively efficient implementation of an association rule miner by Borgelt [2003]. Initial comparison with the GNU R association rule extraction library proved that the implementation of

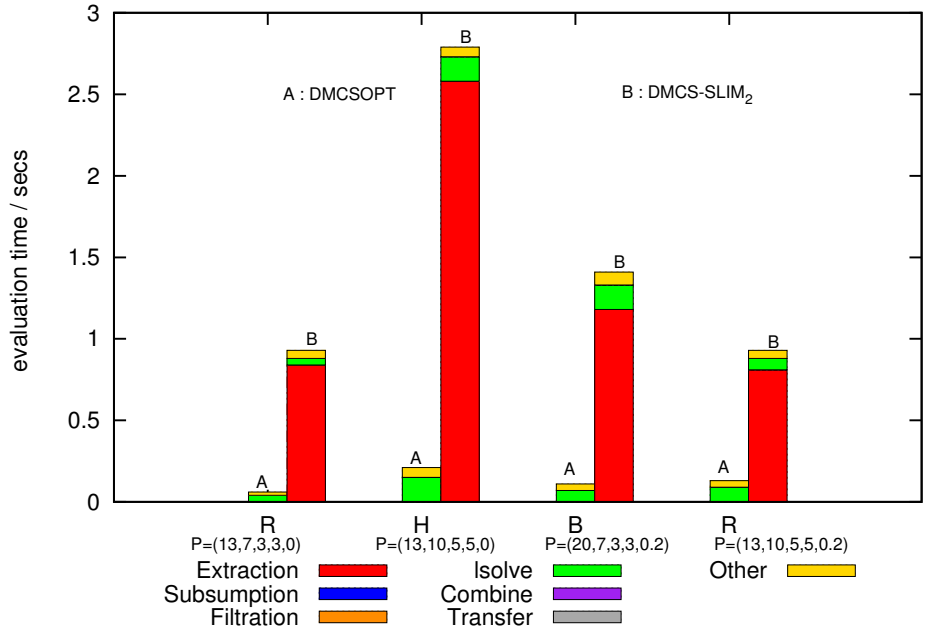


Figure 5.11: Total evaluation time for DMCSOPT vs. DMCS-SLIM$_2$

GNU R is not efficient. Thus, the extra time spent extracting the association rules by DMCS-SLIM could greatly be reduced, if a different library is used. Unfortunately, due to the thesis time constraints, running the experiments with any other libraries was not possible.

5.4 Summary

In this chapter, we attempted to evaluate our implementation of DMCS-SLIM against the other distributed algorithms; namely DMCSOPT and DMCS. In order to achieve that goal, we performed a considerable number of experiments on DMCS-SLIM.

We explained the system setup used in the benchmarking process with respect to the inner structure of the contexts, the topological representation of the system, and the actual experimental setup utilized (hardware and software). Afterwards, we explained the test instances parameters, the randomly generated test instances and the chosen DMCS-SLIM variations.

Comparing several versions of DMCS-SLIM together, as well as DMCS-SLIM with DMCSOPT yielded the following results:

- The effect of removing symmetrical rules is negligible.
- Checking for subsumption is only effective in case of very expensive transmission costs. Otherwise, it can be ignored on the expense of the time needed for filtration.
- The optimum upper limit for the size of association rules is 2.

- The effect of DMCS-SLIM is always visible in cyclic topologies.
- Acyclic topologies are not a good candidate for DMCS-SLIM unless random constraints exist in the knowledge base.
- For inconsistent systems, the local solving overhead is greatly reduced with respect to time and computed results.
- Given an efficient implementation of an association rule miner, DMCS-SLIM might match up against DMCSOPT.

Chapter 6

Conclusion

Multi-context systems emerged as a unifying formalism for integrating knowledge and information for independent units of reasoning. The framework of Brewka and Eiter [2007] was one of the generalizing approaches that allowed for heterogeneous and nonmonotonic multi-context systems. Our work falls under this framework.

Their initial approach was to encode the MCS M as whole into a HEX-program P_M, where the answer sets correspond to the equilibria of M. Afterwards, distributed approaches emerged, of which the first was DMCS. Despite of its innovative approach to compute the MCS equilibrium, it has scalability issues. DMCSOPT improved it, as it reduced the transmission data between contexts and pruned the context dependencies. However, all of the distributed approaches failed to address local optimization.

We developed DMCS-SLIM which is an extension of DMCSOPT as it relies on its decomposition techniques with respect to the topological structure of the multi-context system. The goal behind this development was to optimize local solving at each context using constraint pushing, which gathers information at one context and passes it along to the next context in order to prune its local solving search space. Our approach fulfilled its main objective by utilizing the notion of association rules to represent constraints.

We provided a formal adaptation for association rules with respect to the chosen framework of multi-context systems. Additionally, we presented a customized approach to handle association rules extraction where their soundness is guaranteed. We proved DMCS-SLIM to be sound and complete.

In order to evaluate our system empirically, we provided a prototypical implementation for DMCS-SLIM that was based on the existing DMCS system. We performed exhaustive experiments on several instantiations of DMCS-SLIM, as well as comparing it with DMCS and DMCSOPT.

After interpreting the data gathered from the experimental results, we came to several conclusions. The best version, or rather runtime parameter for DMCS-SLIM, would be the one that includes subsumption checking if the costs for network transmissions are very high. As the optimum size for the extracted association rules is 2, one can set it as an upper limit during the extraction process. The topology of the multi-context system is a very important factor, as

DMCS-SLIM indeed demonstrates its potential in case of cyclic topologies. Finally, one can say that given an efficient implementation of an association rules miner, DMCS-SLIM has the potential to surpass DMCSOPT.

Future Work

There are several open issues with regards to the theory behind DMCS-SLIM. One could try to improve the association rules extraction process by utilizing disjunctive association rules as described in Nanavati et al. [2001], as they already include the concept of subsumption. Another issue would be querying the neighbors of a certain context in parallel instead of the current sequential approach.

It would be interesting to discover sub-classes of multi-context systems where it is always guaranteed that DMCS-SLIM would have the upper hand compared to DMCSOPT.

With regards to system implementation and benchmarking, there is a huge room for improvement. A proper research on fast association rule miner should be performed, where proper benchmark experimentation should be carried on to assess the possibility of DMCS-SLIM exceeding DMCSOPT and by which margin (if any). Additionally, one could investigate the effect of extracting association rules using other association rules extraction algorithms, such as *Partition* algorithm by Savasere et al. [1995] or *Eclat* by Zaki [2000].

Another interesting issue to explore would be evaluating the system using a setup that is based on multiple machines. Thus, one can simulate a real network that acquires and analyzes realistic information with regards to data transfer and transmission costs.

A different approach could be to provide an encoding for our association rules mining process as an ASP program using the approach by Järvisalo [2011], thus enumerating all the answer sets, which could be considered as another form of association rules.

Appendix A

Full trace of scientists example using DMCS-SLIM

In this thesis, we utilized a running example, where the word problem is stated in Example 5, a formulation using ASP logics is presented in Example 6, and it is depicted graphically in Figure 2.3.

In this appendix, we present a full trace of the running example, using our new approach for distributed evaluation for nonmonotonic heterogeneous multi-context systems, DMCS-SLIM. We provide the trace using the subsumption check (Definition 24), and setting the limit for association rule extraction to 4 (Definition 25).

In the sequel, to track variables easier, we augment all the variables with indices illustrating the invoker and the current context under consideration. Thus, $\mathcal{S}_{12}$ means the set of partial belief states in C_2 where the invoker was C_1. Similarly, R_{01} is the set of association rules in C_1 where the invoker was the user, i.e. C_1 is the root context.

The full trace of Example 6 using DMCS-SLIM is provided in Table A.1. We will only comment on some of the trace steps, as most of them are straight forward application of the algorithm:

- (b) at C_1: recall that the value for any $v(k, i)$ is the updated recursive import interface. This is the reason why $v(1, 2)$ contains car_4 and $train_4$, although there is no direct connection between C_1 and C_4.

- (c) at C_1: at this stage, as there are no applicable association rules, the set of computed partial belief states is the same as if we have used DMCSOPT. If the invoker, or rather the user have provided extra set of constraints during the initial call, then this stage might have been changed.

- (e) at C_1: upon reviewing the set of all the extracted association rules. We assert that the generated rules are of different sizes. The only fact that we have seen so far is: $(\neg peanuts_3 \Leftarrow)$, this means that whenever a context is encountered with $peanuts_3$ as part of its alphabet, then it must be the case that the value assigned to this variable is false.

- (b) at C_2: we have a non-empty set of association rules, R_{12}. Applying the applicability relation to this set with respect to C_3, we get a set of one rule only, where it prunes away half of the set of the partial belief state that were normally going to be computed. As a result, all the generated partial belief states have the value of $peanuts_3$ as false, and never as true.
- (c) at C_2: application of lsolve without considering the set of applicable association rule would have lead to doubling the size of the set of partial belief states (22 instead of 11). This is because belief sets with the value of $peanuts_3$ as true would have been considered.
- (e) at C_2: mining at this context leads to the generation of some association rules that are already passed on from the invoker. However, this does not constitute a problem as the $\cup$ function eliminates duplicates.
- (c) at C_3: the size of $\mathcal{S}_{23}$ is 5, which have been reduced from 34 in case of DMCSOPT.
- (c) at C_4: at this context, the applicable association rules have no effect on the resulting set of partial belief states. This is because of the structure of kb_4 and br_4. In any case, one can notice that the applicable rules: $train_4 \Leftarrow \neg car_4$ and $car_4 \Leftarrow \neg train_4$ are in fact symmetrical rules (Section 28).
- (c) at C_5: the size of $\mathcal{S}_{45}$ is 2, which would have been doubled in case of running DMCSOPT.
- (d) at C_5: this is a leaf context, so there is no need to mine for any association rules, thus the direct jump to the cache storing step.
- (i) at C_5: DMCS-SLIM returns from C_5 with only 2 possible partial belief states which are in equilibrium. In case of DMCSOPT, it would have been 4.
- (i) at C_4: As the applicable association rules had no effect on $\mathcal{S}_{34}$, the size of the return set does not change.
- (c) at C_6: no optimization occurs in case of $\mathcal{S}_{36}$ as there are no applicable association rules.
- (i) at C_6: The size of the return set is the same in either DMCS-SLIM or DMCSOPT as $\mathcal{S}_{36}$ is the same.
- (i) at C_3: the size of the return set is dramatically decreased at C_3 from 9 partial belief states to a mere 2.
- (i) at C_2: one of the ramifications induced by the reduction on C_3's partial equilibria appear here, where C_2 returns only 2 partial belief states instead of 3.
- (i) at C_1: it is only natural that the result of C_1 does not show any improvement at all with respect to the number of equilibria. This is a direct consequence of the fact that both DMCSOPT and DMCS-SLIM rely on the same query plan,Π_1, that was generated by the same optimization technique.

Table A.1: Full trace for the running example (Example 6) by DMCS-SLIM

Execution step	**Trace**	**Comments**
(a) at C_1	$cache(1, \emptyset)$ is empty	Initial Call: C_1.DMCS-SLIM$(0, \emptyset)$
(b) at C_1	$\mathcal{T}_{01} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $In(1) := \{2, 3\}$ $v(1, 2) := \{train_2, car_3, train_3,$ $peanuts_3, coke_3, car_4,$ $train_4\}$ $v(1, 3) := \{train_3, peanuts_3,$ $car_3, coke_3\}$ $R_{01}\|_2 := \emptyset$ $R_{01}\|_3 := \emptyset$ after loop: $\mathcal{T}_{01} :=$ omitted for triviality	$\mathcal{T}_{01}$ is the standard binary truth table for all the 5 variables from $v(1, 2) \cup v(1, 3)$ in partial belief state format, total size is 32
(c) at C_1	$\mathcal{S}_{01} := \emptyset$ $R_{01}\|_1 := \emptyset$ after loop: $\mathcal{S}_{01} = \{(\{car_1\}, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \epsilon, \{train_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \epsilon, \{train_3, coke_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \epsilon, \{train_3, car_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \epsilon, \{train_3, coke_3, car_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \epsilon, \{coke_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \epsilon, \{car_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \epsilon, \{coke_3, car_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \{train_2\}, \epsilon, \epsilon, \epsilon, \epsilon)$ $(\{train_1\}, \{train_2\}, \{train_3\}, \epsilon, \epsilon, \epsilon)$ $(\{train_1\}, \{train_2\}, \{train_3, coke_3\}, \epsilon, \epsilon, \epsilon)$ $(\{train_1\}, \{train_2\}, \{train_3, car_3\}, \epsilon, \epsilon, \epsilon)$ $(\{train_1\}, \{train_2\}, \{train_3, coke_3, car_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \{train_2\}, \{coke_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \{train_2\}, \{car_3\}, \epsilon, \epsilon, \epsilon)$ $(\{car_1\}, \{train_2\}, \{coke_3, car_3\}, \epsilon, \epsilon, \epsilon)\}$	
(d) at C_1	$\mathcal{S}_{01} \neq \emptyset \wedge \exists (1, i) \in E(\Pi_1)$	
		Continued on next page

Table A.1 – continued from previous page

Execution Step	Trace	Comments
(e) at C_1	$R_{01} = \emptyset$ $Q_{01} := R_{01} \cup \{\neg peanuts_3 \Leftarrow$ $\neg nuts_1 \Leftarrow$ $train_1 \Leftarrow \neg car_1$ $\neg car_1 \Leftarrow train_1$ $train_2 \Leftarrow \neg car_1$ $train_3 \Leftarrow \neg car_1$ $train_2 \Leftarrow train_1$ $train_3 \Leftarrow train_1$ $\neg train_1 \Leftarrow \neg train_2$ $car_1 \Leftarrow \neg train_2$ $\neg train_1 \Leftarrow \neg train_3$ $car_1 \Leftarrow \neg train_3$ $car_1 \Leftarrow \neg train_1$ $\neg train_1 \Leftarrow car_1$ $\neg car_1 \Leftarrow train_2, train_3$ $train_1 \Leftarrow train_2, train_3$ $\neg train_3 \Leftarrow \neg train_1, train_2$ $\neg train_3 \Leftarrow car_1, train_2$ $\neg train_2 \Leftarrow \neg train_2, train_3$ $\neg train_2 \Leftarrow car_1, train_3\}$	$V^*(1) =$ $\{train_2, car_3, train_3,$ $peanuts_3, coke_3, car_4,$ $train_4, sooner_5, sick_6\}$, is constant throughout the run
(f) at C_1	$\mathcal{T}_{01} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $E(\Pi_1)\|_1 := \{(1, 2)\}$ In Loop: for $(1, 2)$: invoke C_2.DMCS-SLIM$(1, Q_{01})$	According to the query plan Π_1, there is only one neighbor C_2
(a) at C_2	$cache(2, R_{12})$ is empty	R_{12} is Q_{01}
(b) at C_2	$\mathcal{T}_{12} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $In(2) := \{3, 4\}$ $v(2, 3) := \{car_3, train_3, coke_3,$ $peanuts_3, car_4, train_4\}$ $v(2, 4) := \{car_4, train_4\}$ $R_{12}\|_3 := \{lnotpeanuts_3 \Leftarrow\}$ $R_{12}\|_4 := \emptyset$ after loop: $\mathcal{T}_{12} :=$ omitted for triviality	cT_{12} standard binary truth table for all the 5 variables from $v(1, 2) \cup v(1, 3) -$ $\{peanuts_3\}$ in partial belief state format, total size is 32. The variable $peanuts_3$ is only considered with negative values as instructed by the applicable rules

Continued on next page

Table A.1 – continued from previous page

Execution Step	Trace	Comments
(c) at C_2	$\mathcal{S}_{12} := \emptyset$ $R_{12}\|_2 := \{\{\neg peanuts_3\} \Leftarrow \{\}\}$ after loop: $\mathcal{S}_{12} = \{(\epsilon, \{train_2\}, \{train_3\}, \{car_4, train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{train_2\}, \{train_3\}, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3\}, \{car_4, train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3, coke_3\}, \{car_4, train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3, coke_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2, train_2\}, \{car_3, train_3\}, \{car_4, train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{train_2\}, \{car_3, train_3\}, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3, train_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3, train_3, coke_3\}, \{car_4, train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3, train_3, coke_3\}, \{car_4\}, \epsilon, \epsilon)\}$	$R_{12}\|_2$ should prune the search space for lsolve at C_2
(d) at C_2	$\mathcal{S}_{12} \neq \emptyset \wedge \exists(2, i) \in E(\Pi_1)$	
(e) at C_2	$Q_{12} := R_{12} \cup \{\neg peanuts_3 \Leftarrow$ $train_2 \Leftarrow \neg car_3$ $\neg coke_3 \Leftarrow \neg car_3$ $train_3 \Leftarrow \neg car_3$ $train_4 \Leftarrow \neg car_3$ $train_2 \Leftarrow \neg car_4$ $\neg coke_3 \Leftarrow \neg car_4$ $train_3 \Leftarrow \neg car_4$ $train_4 \Leftarrow \neg car_4$ $\neg coke_3 \Leftarrow train_2$ $train_3 \Leftarrow train_2$ $train_4 \Leftarrow train_2$ $\neg train_2 \Leftarrow \neg train_3$ $car_4 \Leftarrow \neg train_3$ $car_3 \Leftarrow \neg train_3$ $\neg train_2 \Leftarrow \neg train_4$ $car_4 \Leftarrow \neg train_4$ $car_3 \Leftarrow \neg train_4$ $\neg train_2 \Leftarrow coke_3$ $car_4 \Leftarrow coke_3$	

Continued on next page

Table A.1 – continued from previous page

Execution Step	Trace	Comments
	$car_3 \Leftarrow coke_3$ $car_4 \Leftarrow \neg train_2$ $car_3 \Leftarrow \neg train_2$ $train_2 \Leftarrow \neg coke_3, train_3, train_4\}$	
(f) at C_2	$\mathcal{T}_{12} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $E(\Pi_1)\|_2 := \{(2, 3)\}$ In Loop: for $(2, 3)$: invoke C_3.DMCS-SLIM$(2, Q_{12})$	According to the query plan Π_1, there is only one neighbor C_3
(a) at C_3	$cache(3, R_{23})$ is empty	R_{23} is Q_{12}
(b) at C_3	$\mathcal{T}_{23} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $In(3) := \{4, 6\}$ $v(3, 4) := \{car_4, train_4\}$ $v(3, 6) := \{sick_6\}$ $R_{23}\|_4 := \{train_4 \Leftarrow \neg car4$ $car_4 \Leftarrow \neg train4\}$ $R_{23}\|_6 := \emptyset$ after loop: $\mathcal{T}_{23} = \{(\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \{sick_6\})$ $(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \{sick_6\})$ $(\epsilon, \epsilon, \epsilon, \{car_4, train_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{car_4, train_4\}, \epsilon, \{sick_6\})\}$	
(c) at C_3	$\mathcal{S}_{23} := \emptyset$ $R_{23}\|_3 := \{lnotpeanuts_3 \Leftarrow$ $\neg coke_3 \Leftarrow \neg car_3$ $train_3 \Leftarrow \neg car_3$ $train_4 \Leftarrow \neg car_3$ $\neg coke_3 \Leftarrow \neg car_4$ $train_3 \Leftarrow \neg car_4$ $train_4 \Leftarrow \neg car_4$ $car_4 \Leftarrow \neg train_3$ $car_3 \Leftarrow \neg train_3$ $car_4 \Leftarrow \neg train_4$ $car_3 \Leftarrow \neg train_4$	Search space of lsolve heavily reduced when compared to DMCSOPT

Continued on next page

Table A.1 – continued from previous page

Execution Step	Trace	Comments
	$car_4 \Leftarrow coke_3$ $car_3 \Leftarrow coke_3\}$ after loop: $\mathcal{S}_{23} = \{(\epsilon, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3\}, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3\}, \{car_4, train_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3, urgent_3\}$ $\{train_4\}, \epsilon, \{sick_6\})$ $(\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3, urgent_3\}$ $\{car_4, train_4\}, \epsilon, \{sick_6)\}\}$	
(d) at C_3	$\mathcal{S}_{23} \neq \emptyset \wedge \exists(3, i) \in E(\Pi_1)$	
(e) at C_3	$Q_{23} := R_{23} \cup \{\neg coke_3 \Leftarrow$ $\neg peanuts_3 \Leftarrow$ $\neg train_3 \Leftarrow \neg train_4$ $\neg train_4 \Leftarrow \neg train_3$ $\neg car_3 \Leftarrow \neg train_4$ $\neg train_4 \Leftarrow car_3$ $\neg sick_6 \Leftarrow \neg train_4$ $car_4 \Leftarrow \neg train_4$ $car_3 \Leftarrow \neg train_3$ $\neg train_3 \Leftarrow car_3$ $\neg sick_6 \Leftarrow \neg train_3$ $car_4 \Leftarrow \neg train_3$ $\neg sick_6 \Leftarrow car_3$ $train_4 \Leftarrow car_3$ $\neg car_3 \Leftarrow \neg car_4$ $train_3 \Leftarrow \neg car_4$ $train_4 \Leftarrow \neg car_4$ $\neg car_3 \Leftarrow sick_6$ $train_3 \Leftarrow sick_6$ $train_4 \Leftarrow sick_6$ $train_3 \Leftarrow \neg car_3$ $\neg car_3 \Leftarrow train_3$ $train_4 \Leftarrow \neg car_3$ $\neg car_3 \Leftarrow train_4$	

Continued on next page

Table A.1 – continued from previous page

Execution Step	Trace	Comments
	$train_4 \Leftarrow train_3$ $train_3 \Leftarrow train_4\}$	
(f) at C_3	$\mathcal{T}_{23} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $E(\Pi_1)\|_3 := \{(3,4),(3,6)\}$ In Loop: for $(3,4)$: invoke C_4.DMCS-SLIM$(3, Q_{23})$	According to the query plan Π_1, there are two neighbors C_4 and C_6
(a) at C_4	$cache(4, R_{34})$ is empty	R_{34} is Q_{23}
(b) at C_4	$\mathcal{T}_{34} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $In(4) := \{5\}$ $v(4,5) := \{car_4, train_4, sooner_5\}$ $R_{34}\|_5 := \emptyset$ after loop: $\mathcal{T}_{34} :=$ omitted for triviality	$\mathcal{T}_{34}$ standard binary truth table for the 5 variables from $v(4,5)$ in partial belief state format,total size is 2
(c) at C_4	$\mathcal{S}_{34} := \emptyset$ $R_{34}\|_4 := \{train_4 \Leftarrow \neg car_4$ $car_4 \Leftarrow \neg train_4\}$ after loop: $\mathcal{S}_{34} = \{(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \{sooner_5\}, \epsilon)\}$	DMCS-SLIM and DMCSOPT computed same results
(d) at C_4	$\mathcal{S}_{34} \neq \emptyset \wedge \exists (4, i) \in E(\Pi_1)$	
(e) at C_4	$Q_{34} := R_{34} \cup \{car_4 \Leftarrow \neg train_4$ $\neg train_4 \Leftarrow car_4$ $\neg want\_sooner_5 \Leftarrow \neg train_4$ $\neg want\_sooner_5 \Leftarrow car_4$ $\neg car_4 \Leftarrow want\_sooner_5$ $train_4 \Leftarrow want\_sooner_5$ $train_4 \Leftarrow \neg car_4$ $\neg car_4 \Leftarrow train_4\}$	
(f) at C_4	$\mathcal{T}_{34} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $E(\Pi_1)\|_4 := \{(4,5)\}$ In Loop: for $(4,5)$: invoke C_5.DMCS-SLIM$(4, Q_{34})$	According to the query plan Π_1, there is only one neighbor C_5
(a) at C_5	$cache(5, R_{45})$ is empty	R_{45} is Q_{34}
(b) at C_5	$\mathcal{T}_{45} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$	

Continued on next page

Table A.1 – continued from previous page

Execution Step	Trace	Comments
	$In(5) := \{4\}$ $v(5,4) := \{train_4, car_4\}$ $R_{45}\|_4 := \{train_4 \Leftarrow \neg car_4$ $car_4 \Leftarrow \neg train_4$ $\neg train_4 \Leftarrow car_4$ $\neg car_4 \Leftarrow train_4\}$ after loop: $\mathcal{T}_{45} = \{(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \}, \epsilon)\}$	
(c) at C_5	$\mathcal{S}_{45} := \emptyset$ $R_{45}\|_5 := \{train_4 \Leftarrow \neg car_4$ $car_4 \Leftarrow \neg train_4$ $\neg train_4 \Leftarrow car_4$ $sooner_5 \Leftarrow \neg train_4$ $sooner_5 \Leftarrow car_4$ $\neg car_4 \Leftarrow want\_sooner_5$ $train_4 \Leftarrow want\_sooner_5$ $\neg car_4 \Leftarrow train_4\}$ after loop: $\mathcal{S}_{45} = \{(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \{sooner_5, soon_5\}, \epsilon)\}$	DMCSOPT produced double the number of belief states w.r.t. DMCS-SLIM
(d) at C_5	$\mathcal{S}_{45} \neq \emptyset \wedge \exists(5,i) \notin E(\Pi_1)$	Skip to (h) directly
(h) at C_5	$cache(5, R_{45}) := \mathcal{S}_{45}$	
(i) at C_5	$(4,5) \in E(\Pi_1)$ $\mathcal{S}_{45} := \mathcal{S}_{45}\|_{v(4,5)} =$ $\{(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \{sooner_5, \}, \epsilon)\}$ return $\mathcal{S}_{45}$	Returned two partial equilibria
(f) at C_4 continuation	after loop $\mathcal{T}_{34} := \mathcal{S}_{45}$	
(g) at C_4	$\mathcal{S}_{34} := \{(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \{sooner_5\}, \epsilon)\}$	Filter peq that do not have $\{train_4\}$ and $\{sooner_5\}$
(h) at C_4	$cache(4, R_{34}) := \mathcal{S}_{34}$	
(i) at C_4	$(3,4) \in E(\Pi_1)$ $\mathcal{S}_{34} := \mathcal{S}_{34}\|_{v(3,4)} =$	Returned two partial equilibria
Continued on next page		

Table A.1 – continued from previous page

Execution Step	Trace	Comments
	$\{(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \epsilon)\}$ return $\mathcal{S}_{34}$	
(f) at C_3 cont.	$\mathcal{T}_{23} := \mathcal{S}_{34}$ for $(3, 6)$: invoke C_6.DMCS-SLIM$(3, Q_{23})$	Now invoke C_6
(a) at C_6	$cache(6, R_{36})$ is empty	R_{36} is Q_{23}
(b) at C_6	$\mathcal{T}_{36} := \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \epsilon)\}$ $In(6) := \emptyset$ loop ignored:	C_6 is a leaf node
(c) at C_6	$\mathcal{S}_{36} := \emptyset$ $R_{36}\|_6 := \emptyset$ after loop: $\mathcal{S}_{36} = \{(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \{sick_6\})$ $(\epsilon, \epsilon, \epsilon, \epsilon, \epsilon, \{fit_6\})\}$	
(d) at C_6	$\mathcal{S}_{36} \neq \emptyset \wedge \exists(6, i) \notin E(\Pi_1)$	Skip to (h) directly
(h) at C_6	$cache(6, R_{36}) := \mathcal{S}_{36}$	
(i) at C_6	$(3, 6) \in E(\Pi_1)$ $\mathcal{S}_{36} := \mathcal{S}_{36}\|_{v(3,6)}$ return $\mathcal{S}_{36}$	No change for $\mathcal{S}_{36}$
(f) at C_3 continuation	after Loop $\mathcal{T}_{23} := (\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{car_4\}, \epsilon, \{sick_6\})$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \epsilon, \{train_4\}, \epsilon, \{sick_6\})\}$	Represents all partial equilibria for all neighbors
(g) at C_3	$\mathcal{S}_{23} := (\epsilon, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3\}, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \{train_3, sandwiches_3, juice_3, urgent_3\}$ $\{train_4\}, \epsilon, \{sick_6\})\}$	Partial equilibria reduced to 3 from 5
(h) at C_3	$cache(3, R_{23}) := \mathcal{S}_{23}$	
(i) at C_3	$(2, 3) \in E(\Pi_1)$ $\mathcal{S}_{23} := \mathcal{S}_{23}\|_{v(2,3)} =$ $\{(\epsilon, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \epsilon, \{train_3\}, \{train_4\}, \epsilon, \epsilon)\}$ return $\mathcal{S}_{23}$	Returns two partial equilibria, DMCS-SLIM shows great improvement
(f) at C_2	after loop	Represents all partial

Continued on next page

Table A.1 – concluded from previous page

Execution Step	Trace	Comments
continuation	$\mathcal{T}_{12} := \mathcal{S}_{23}$	equilibria for neighbors
(g) at C_2	after loop $\mathcal{S}_{12} := (\epsilon, \{train_2\}, \{train_3\}, \{train_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \{car_2\}, \{car_3, coke_3\}, \{car_4\}, \epsilon, \epsilon)\}$	Partial equilibria reduced to 11 from 3
(h) at C_2	$cache(2, R_{12}) := \mathcal{S}_{12}$	
(i) at C_2	$(1, 2) \in E(\Pi_1)$ $\mathcal{S}_{12} := \mathcal{S}_{12}\|_{v(1,2)} =$ $\{(\epsilon, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\epsilon, \{train_2\}, \{train_3\}, \{train_4\}, \epsilon, \epsilon)\}$ return $\mathcal{S}_{12}$	returns two partial equilibria
(f) at C_1 continuation	after loop $\mathcal{T}_{01} := \mathcal{S}_{12}$	Represents all partial equilibria from neighbors
(g) at C_1	$\mathcal{S}_{01} = \{(\{car_1\}, \epsilon, \{car_3\}, \{car_4\}, \epsilon, \epsilon)$ $(\{train_1\}, \{train_2\}, \{train_3\}, \{train_4\}, \epsilon, \epsilon)\}$	Partial equilibria reduced to 16 from 2
(h) at C_1	$cache(1, R_{01}) := \mathcal{S}_{01}$	
(i) at C_1	$(0, 1) \notin E(\Pi_1)$ return $\mathcal{S}_{01}$	Return partial equilibria for M w.r.t. C_1

Bibliography

Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, Santiago, Chile, September 1994.

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, volume 22, pages 207–216, New York, NY, USA, June 1993. ACM. ISBN 0-89791-592-5. doi: 10.1145/170035.170072. URL `http://dx.doi.org/10.1145/170035.170072`.

Rajendra Akerkar and Priti Sajja. *Knowledge-Based Systems*. Jones and Bartlett Publishers, Inc., USA, 1st edition, 2009. ISBN 0763776475, 9780763776473.

Seif El-Din Bairakdar, Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner. Decomposition of Distributed Nonmonotonic Multi-Context Systems. In Tomi Janhunen and Ilkka Niemelä, editors, *12th European Conference on Logics in Artificial Intelligence (JELIA 2010), Helsinki, Finland, September 13-15, 2010*, volume 6341 of *LNAI*, pages 24–37. Springer, September 2010a. URL `http://www.kr.tuwien.ac.at/staff/tkren/pub/2010/jelia2010-decompmcs.pdf`.

Seif El-Din Bairakdar, Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner. The DMCS Solver for Distributed Nonmonotonic Multi-Context Systems. In Tomi Janhunen and Ilkka Niemelä, editors, *12th European Conference on Logics in Artificial Intelligence (JELIA 2010), Helsinki, Finland, September 13-15, 2010*, volume 6341 of *LNAI*, pages 352–355. Springer, September 2010b. doi: 10.1007/978-3-642-15675-5_30. URL `http://www.kr.tuwien.ac.at/staff/tkren/pub/2010/jelia2010-dmcs-system.pdf`.

Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve, and James B. Rothnie. Query processing in a system for distributed databases (sdd-1). *ACM Transactions on Database Systems*, 6:602–625, 1981.

Adrian Bondy and U. S. R. Murty. *Graph Theory*, volume 244 of *Graduate Texts in Mathematics*. Springer, 2008.

Christian Borgelt. Efficient implementations of apriori and eclat. In *Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations (FIMI 2003, Melbourne, FL). CEUR Workshop Proceedings 90*, page 90, 2003.

Gerhard Brewka and Thomas Eiter. Equilibria in heterogeneous nonmonotonic multi-context systems. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 1*, pages 385–390. AAAI Press, 2007. ISBN 978-1-57735-323-2. URL http://portal.acm.org/citation.cfm?id=1619645.1619707.

Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner. Distributed Nonmonotonic Multi-Context Systems. In Fangzhen Lin and Uli Sattler, editors, *12th International Conference on the Principles of Knowledge Representation and Reasoning (KR2010), Toronto, Canada, May 9-13, 2010*. AAAI Press, May 2010. URL http://www.kr.tuwien.ac.at/staff/tkren/pub/2010/kr2010-dmcs.pdf.

Premkumar Devanbu and Eric Wohlstadter. Managing evolution in distributed heterogeneous systems. In *NFS Workshop on New Visions for Software Design and Productivity: Research and Applications*, 2001.

Jean Dollimore, Tim Kindberg, and George Coulouris. *Distributed Systems: Concepts and Design (4th Edition) (International Computer Science Series)*. Addison Wesley, May 2005. ISBN 0321263545. URL http://www.worldcat.org/isbn/0321263545.

Thomas Eiter, Giovambattista Ianni, Roman Schindlauer, and Hans Tompits. A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *In Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 90–96. Professional Book, 2005.

Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. *Answer Set Programming: A Primer*, pages 40–110. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-03753-5. doi: 10.1007/978-3-642-03754-2_2. URL http://portal.acm.org/citation.cfm?id=1611703.1611705.

Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392. AAAI Press, 2007a. URL http://dli.iiit.ac.in/ijcai/IJCAI-2007/PDF/IJCAI07-060.pdf.

Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. clasp: A conflict-driven answer set solver. In *Proceedings of the 9th International Conference Logic Programming and Nonmonotonic Reasoning (LPNMR-2007), May 14–17, 2007,Tempe, AZ, USA*, pages 260–265. Springer, 2007b.

Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–385, 1991. doi: 10.1007/BF03037169. URL `http://www.cs.utexas.edu/~vl/mypapers/clnegdd.ps`.

Chiara Ghidini and Fausto Giunchiglia. Local models semantics, or contextual reasoning = locality + compatibility. *Artificial Intelligence*, 127(2):221–259, 2001.

Fausto Giunchiglia and Fausto Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, 345:345–364, 1992.

Fausto Giunchiglia and Luciano Serafini. Multilanguage hierarchical logics or: How we can do without modal logics. *Artificial Intelligence*, 65(1):29–70, 1994.

Michael Hahsler, Bettina Gruen, and Kurt Hornik. A computational environment for mining association rules and frequent item sets. *Journal of Statistical Software*, 14(15):1–25, October 2005. ISSN 1548-7660. URL `http://www.jstatsoft.org/v14/i15/`.

Michael Hahsler, Christian Buchta, Bettina Gruen, and Kurt Hornik. *arules: Mining Association Rules and Frequent Itemsets*, 2011. URL `http://CRAN.R-project.org/`. R package version 1.0-5.

Matti Järvisalo. Itemset Mining as a Challenge Application for Answer Set Enumeration. In *11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2011), Vancouver, BC, Canada, May 16-19, 2011*, LNAI. Springer, 2011. To appear.

Carl M. Kadie. Rational nonmonotonic reasoning. In *Proceedings of the Fourth Workshop on Uncertainty in Artificial Intelligence, Minneapolis, August*, 1988.

Gerhard Lakemeyer and Bernhard Nebel. Foundations of knowledge representation and reasoning. In *Foundation of Knowledge Representation and Reasoning [the book grew out of an ECAI-92 workshop]*, pages 1–12, London, UK, 1994. Springer-Verlag. ISBN 3-540-58107-3. URL `http://portal.acm.org/citation.cfm?id=645296.648478`.

Joohyung Lee and Vladimir Lifschitz. Loop formulas for disjunctive logic programs. In *ICLP'03*, pages 451–465. Springer, 2003.

Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.*, 157(1-2):115–137, 2004. doi: 10.1016/j.artint.2004.04.004.

Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 0387244352.

John McCarthy. Notes on formalizing context. In *International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 555–562, 1993.

Pamela McCorduck. *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligencetts*. AK Peters, 2004. ISBN 1-56881-205-1.

F. Najjar and Y. Slimani. The enhancement of semijoin strategies in distributed query optimization. In David Pritchard and Jeff Reeve, editors, *Euro-Par'98 Parallel Processing*, volume 1470 of *Lecture Notes in Computer Science*, pages 528–533. Springer Berlin / Heidelberg, 1998. URL `http://dx.doi.org/10.1007/BFb0057897`. 10.1007/BFb0057897.

Amit A. Nanavati, Krishna P. Chitrapura, Sachindra Joshi, and Raghu Krishnapuram. Mining generalised disjunctive association rules. In *Proceedings of the tenth international conference on Information and knowledge management*, CIKM '01, pages 482–489, New York, NY, USA, 2001. ACM. ISBN 1-58113-436-3.

Salvatore Orlando, Claudio Lucchese, Paolo Palmerini, Raffaele Perego, and Fabrizio Silvestri. kdci: a multi-strategy algorithm for mining frequent sets. In Bart Goethals and Mohammed J. Zaki, editors, *FIMI'03: Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, November 2003.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Juan R. Rabuñal, Julian Dorado, and Alejandro Pazos, editors. *Encyclopedia of Artificial Intelligence (3 Volumes)*. IGI Global, 2009. ISBN 9781599048499.

Raymond Reiter. A logic for default reasoning. *Artif. Intell.*, 13(1-2):81–132, 1980.

Floris Roelofsen, Luciano Serafini, and Alessandro Cimatti. Many Hands Make Light Work: Localized Satisfiability for Multi-Context Systems. In Ramon López de Mántaras and Lorenza Saitta, editors, *16th Eureopean Conference on Artificial Intelligence (ECAI'04)*, pages 58–62. IOS Press, 2004. URL `http://people.umass.edu/froelofs/professional/documents/ecai04_roelofsen_serafini_cimatti.pdf`.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

Ashok Savasere, Edward Omiecinski, and Shamkant Navathe. An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21st VLDB Conference*, pages 432–443, Zurich, Switzerland, 1995.

Mohammed J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, May/June 2000.

Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In F. Provost and R. Srikant, editors, *Proceedings of the ACM SIGKDD Intentional Conference on Knowledge Discovery in Databases and Data Mining (KDD-01)*, pages 401–406. ACM Press, 2001.