Decomposition of Distributed Nonmonotonic Multi-Context Systems[⋆]

Seif El-Din Bairakdar, Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{bairakdar,dao,eiter,fink,tkren}@kr.tuwien.ac.at

Abstract. Multi-Context Systems (MCS) are formalisms that enable the interlinkage of single knowledge bases, called contexts, via bridge rules. Recently, a fully distributed algorithm for evaluating heterogeneous, nonmonotonic MCS was described in [7]. In this paper, we continue this line of work and present a decomposition technique for MCS which analyzes the topology of an MCS. It applies pruning techniques to get economically small representations of context dependencies. Orthogonal to this, we characterize minimal interfaces for information exchange between contexts, such that data transmissions can be minimized. We then present a novel evaluation algorithm that operates on a query plan which is compiled with topology pruning and interface minimization. The effectiveness of the optimization techniques is demonstrated by a prototype implementation, which uses an off-the-shelf SAT solver and shows encouraging experimental results.

1 Introduction

In the last years, there has been increasing interest in systems comprising multiple knowledge bases. The rise of distributed systems and the World Wide Web fostered this development, and to date, several formalisms are available that accommodate multiple, possibly distributed knowledge bases. One formalism are Multi-Context Systems (MCS) consisting of several theories (the contexts) that are interlinked with bridge rules which allow to add knowledge to a context depending on knowledge in other contexts. E.g., the bridge rule $a \leftarrow (2:b)$ of a context C_1 means that C_1 should conclude a if context C_2 believes b. MCS have applications in various areas, such as argumentation, data integration, or multi-agent systems. There, contexts may model the beliefs of an agent while the bridge rules model an agent's perception of the environment, i.e., other contexts.

Among the various MCS proposals (e.g., [10–12]), the general MCS framework of [5] is of special interest, as it generalizes previous approaches in contextual reasoning and allows for *heterogeneous and nonmonotonic* MCS, 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 MCS interlinking monotonic context logics are possible. This MCS framework can conveniently capture the following scenario, which we use as a running example.

[⋆] This research has been supported by the Austrian Science Fund (FWF) project P20841 and by the Vienna Science and Technology Fund (WWTF) project ICT 08-020.

Example 1. 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.

Similar scenarios have already been investigated in the realm of multi-agent systems (see, e.g., [6] on social answer set programming). We do not aim at introducing a new semantics for such scenarios; our example is meant to be a plain showcase application of MCS. We stress that MCS have potential as a host for KR formalisms, just like answer set programs have; however, in this paper we concentrate on efficient MCS evaluation.

The distributed algorithm introduced in [7], called DMCS, computes the semantics of an MCS, which is given in terms of equilibria. Roughly, an equilibrium is a collection of local models (belief sets) for the individual contexts that is compatible with the bridge rules. The principle of the algorithm is, starting from context C_k (the root), that models will be processed at each context. Bridge rules, which access beliefs in other contexts, implicitly span belief import dependencies between contexts. This relationship is used to navigate the system, and models returned from invoked neighbors are combined with the local beliefs and passed back to the invoking contexts. DMCS uses a parameter for projecting models to relevant variables to reduce data payload.

Experiments for an instantiation of DMCS with answer set programming contexts revealed some scalability issues which can be tracked down to the following problems:

(1) contexts are unaware of context dependencies in the system beyond their neighbors, and thus treat each neighbor in a generic way. Specifically, cyclic dependencies remain undetected until a context, seeing the invocation chain, requests models from a context in the chain. Furthermore, a context C_k does not know whether a neighbor C_i already requests models from another neighbor C_j which then would be passed to C_k; hence, C_k makes possibly a superfluous request to C_j.

(2) a context C_i returns the combination of its local models with the models received from all neighboring contexts. As contexts may have multiple models, the number of models can become huge as the size of the system respectively neighbors increases. In fact, this is one of the main performance obstacles.

In this work, we address the issue of optimization; there is an urgent need for this in order to increase the scalability of distributed MCS evaluation. Resorting to methods from graph theory, we aim at decomposing, pruning, and improved cycle breaking for dependencies in multi-context systems. Focusing on (1), we describe a decomposition method using biconnected components of inter-context dependencies. Based on this we can break cycles and prune acyclic parts before evaluating the system and create an acyclic query plan. To address (2), we foster a partial view of the system, which is often sufficient to reach a satisfactory answer. In Ex. 1, e.g., we could mask out the beliefs of Mr. 5 and Ms. 6 to compute a partial equilibrium within the scientist group. This way we can make a compromise between partial information and performance. We thus define a set of variables for each import dependency in the system to project the models in each context to the bare minimum such that they continue to be meaningful. In this manner, we can omit needless information and circumvent excessive model combinations.

Based on these ideas, we have designed a new evaluation algorithm DMCSOPT, which intertwines decomposition and pruning with variable projection. For evaluation, we adapted our DMCS prototype and ran some experiments. The results show a major improvement compared to DMCS; here we can handle systems with up to 600 contexts. This demonstrates that our optimization techniques are effective and bring MCS closer to applications.

2 Preliminaries

We recall some basic notions of heterogeneous nonmonotonic multi-context systems [5].

A *logic* is, viewed abstractly, a tuple $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$, where

- $\mathbf{KB}_L$ is a set of well-formed knowledge bases, each being a set (of formulas),
- $\mathbf{BS}_L$ is a set of possible belief sets, each being a set (of formulas), and
- $\mathbf{ACC}_L \colon \mathbf{KB}_L \to 2^{\mathbf{BS}_L}$ assigns each $kb \in \mathbf{KB}_L$ a set of acceptable belief sets.

This covers many (non-)monotonic KR formalisms like description logics, default logic, answer set programs, etc. For example, a (propositional) *ASP logic* L may be 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, and $\mathbf{ACC}_L$ assigns each $kb \in \mathbf{KB}_L$ the set of all its answer sets (see [9] for details).

Definition 1. *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 *of the form*

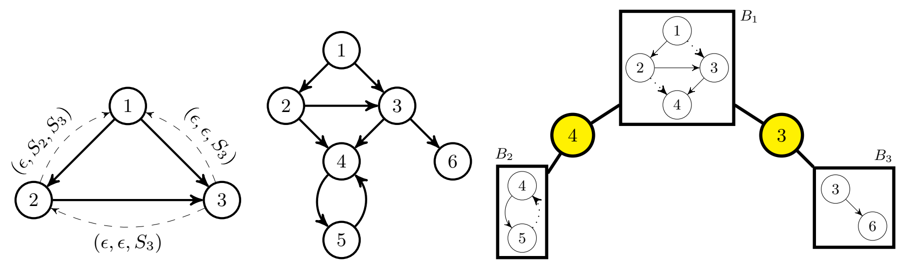
$$s \leftarrow (c_1 : p_1), \ldots, (c_j : p_j), \text{not}\,(c_{j+1} : p_{j+1}), \ldots, \text{not}\,(c_m : p_m) \tag{1}$$

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* $kb \cup \{s\} \in \mathbf{KB}_i$ *for each* $kb \in \mathbf{KB}_i$*.*

Informally, bridge rules allow to modify the knowledge base by adding s, depending on the beliefs in other contexts.

The semantics of an MCS M is defined in terms of particular *belief states*, which are sequences $S = (S_1, \ldots, S_n)$ of belief sets $S_i \in \mathbf{BS}_i$. Intuitively, S_i should be a belief set of the knowledge base kb_i; however, also the bridge rules br_i must be respected. To this end, kb_i is augmented with the conclusions of all $r \in br_i$ that are applicable.

Formally, r of form (1) is *applicable in* S, if $p_i \in S_{c_i}$, for $1 \leq i \leq j$, and $p_k \notin S_{c_k}$, for $j + 1 \leq k \leq m$. Let $app(R, S)$ denote the set of all bridge rules $r \in R$ that are



(a) Triangle (b) Diamond-ring (c) Diamond-ring block tree

Fig. 1: Topologies and Decomposition of Scientist Group Example

applicable in S. Furthermore, $head(r)$ denotes the part s, and $B(r) = \{(c_k : p_k) \mid 1 \leq k \leq m\}$, for any r of form (1).

Definition 2. *A belief state $S = (S_1, \dots, S_n)$ of a multi-context system M is an* equilibrium *iff for all $1 \leq i \leq n$, $S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$.*

In the rest of this paper, we assume that contexts C_i have finite belief sets S_i that are represented by truth assignments $v_{S_i} \colon \Sigma_i \to \{0, 1\}$ to a finite set Σ_i of propositional atoms such that $p \in S_i$ iff $v_{S_i}(p) = 1$ (as in [5], 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 $\Sigma = \bigcup_i \Sigma_i$.

Example 2. The scenario in Ex. 1 can be encoded as an MCS $M = (C_1, \dots, C_6)$, where all L_i are ASP logics (t_i and c_i represent train and car in C_i, resp.) and

- $kb_1 = \{c_1 \leftarrow \text{not } t_1; \quad \bot \leftarrow nuts_1\}$ and
  $br_1 = \{t_1 \leftarrow (2 : t_2), (3 : t_3); \quad nuts_1 \leftarrow (3 : peanuts_3)\}$;
- $kb_2 = \{\bot \leftarrow \text{not } c_2, \text{not } t_2\}$ and
  $br_2 = \{c_2 \leftarrow (3 : c_3), (4 : c_4); \quad t_2 \leftarrow (3 : t_3), (4 : t_4), \text{not } (3 : coke_3)\}$;
- $kb_3 = \{c_3 \vee t_3 \leftarrow; t_3 \leftarrow urgent_3; salad_3 \vee peanuts_3 \leftarrow t_3; coke_3 \vee juice_3 \leftarrow t_3\}$ and $br_3 = \{urgent_3 \leftarrow (6 : sick_6); \ t_3 \leftarrow (4 : t_4)\}$;
- $kb_4 = \big\{c_4 \vee t_4 \leftarrow\big\}$ and $br_4 = \big\{t_4 \leftarrow (5 : sooner_5)\big\}$;
- $kb_5 = \big\{sooner_5 \leftarrow soon_5\big\}$ and $br_5 = \big\{soon_5 \leftarrow (4 : t_4)\big\}$;
- $kb_6 = \big\{sick_6 \vee fit_6 \leftarrow\big\}$ and $br_6 = \emptyset$.

The context dependencies of M are shown in Fig. 1b. M has three equilibria:

- $S = (\{t_1\}, \{t_2\}, \{t_3, urgent_3, juice_3, salad_3\}, \{t_4\}, \{soon_5, sooner_5\}, \{sick_6\})$;
- $T = (\{t_1\}, \{t_2\}, \{t_3, juice_3, salad_3\}, \{t_4\}, \{soon_5, sooner_5\}, \{fit_6\})$; and
- $U = (\{c_1\}, \{c_2\}, \{c_3\}, \{c_4\}, \emptyset, \{fit_6\})$.

Partial Equilibria. We recall partial equilibria [7], which informally are equilibria of a sub-MCS generated by a context C_k.

Definition 3 (Import Closure). *Let $M = (C_1, \dots, C_n)$ 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 $j \in S$, $In(j) \subseteq S$.*

Based on the import closure, we then define:

Definition 4 (Partial Belief States and Equilibria). *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 $1 \leq i \leq n$. A partial belief state $S=(S_1,\ldots,S_n)$ of 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 $1 \leq i \leq n$.*

For instance, in our running example $(\epsilon, \epsilon, \{c_3, coke_3, peanuts_3\}, \{t_4\}, \{soon_5, sooner_5\}, \{fit_6\})$ is a partial equilibrium w.r.t. context C_3.

For combining partial belief states $S=(S_1,\ldots,S_n)$ and $T=(T_1,\ldots,T_n)$, we define their *join* $S \bowtie T$ 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 $1 \leq i \leq n$. Note that $S \bowtie T$ is void, if some S_i, T_i are from $\mathbf{BS}_i$ but different. The *join* of two sets $\mathcal{S}$ and $\mathcal{T}$ of partial belief states is then naturally defined as $\mathcal{S} \bowtie \mathcal{T} = \{S \bowtie T \mid S \in \mathcal{S}, T \in \mathcal{T}\}$.

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 $S = (S_1|_V, \ldots, S_n|_V)$, where $S_i|_V = S_i \cap V$ if $S_i \neq \epsilon$, and $\epsilon|_V = \epsilon$; for a set of (partial) belief states $\mathcal{S}$, we let $\mathcal{S}|_V = \{S|_V \mid S \in \mathcal{S}\}$.

Definition 5. *The* import interface *of context C_k in M is $V(k) = \{p \mid (c:p) \in B(r), r \in br_k\}$, and its* recursive import interface *is $V^*(k) = V(k) \cup \{p \in V(j) \mid j \in IC(k)\}$.*

As an example, the recursive import interface of C_1 in M from Ex. 2 is $V^*(1) = \{c_3, c_4, peanuts_3, coke_3, sick_6, t_2, t_3, t_4, sooner_5\}$.

There are two extremal cases: 1. $V = V^*(k)$. Then, partial equilibria projected to V can be basically used for consistency-checking on the import closure of C_k. 2. $V = \Sigma$. Here, the projection to V yields partial equilibria w.r.t. C_k. By providing a fixed interface V such that $V^*(k) \subseteq V \subseteq \Sigma$, problem-specific knowledge (e.g. query variables) and infrastructure information can be exploited to focus computations to relevant projections.

3 Decomposition of Nonmonotonic MCS

Reconsider our running example, with all contexts C_i and atoms referring to them removed, for $i > 3$. Then, C_1 has bridge rules with atoms of form $(2 : p_2)$ and $(3 : p_3)$ in the body, and C_2 with atoms $(3 : p_3)$. That is, C_1 depends on both C_2 and C_3, while C_2 depends on C_3 (see Fig. 1a). A straightforward approach to evaluate this modified MCS is to ask in C_1 for the belief sets of C_2 and C_3. But as C_2 also depends on C_3, we would need another query from C_2 to C_3 to evaluate C_2 w.r.t. the belief sets of C_3. This shows that there is some evident redundancy in this approach, as C_3 will need to compute its belief sets twice. Simple caching strategies could mellow out the second belief state building in C_3; nonetheless, when C_1 asks C_3, the context will transmit back its belief states, thus consuming network resources.

Moreover, when C_2 asks for the partial equilibria of C_3, it will receive a set of partial equilibria that covers the belief sets of C_3 and in addition all contexts in the import closure $IC(3)$. This is excessive from the view of C_1, as it only needs to know the truth of $(2 : p_2)$ and $(3 : p_3)$. However, C_1 needs the belief states of both C_2 and C_3 in reply of C_2: if C_2 only reports its own belief sets (which are consistent w.r.t. C_3), then C_1 has no chance to align the belief sets received from C_2 with those received from C_3. Realizing that C_2 also reports the belief sets of C_3, no call to C_3 must be made.

Based on this, we present an optimization strategy which 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.

Graph-Theoretic Concepts. We start with defining the topology of an MCS.

Definition 6. *The* topology *of an MCS $M = (C_1, \dots, C_n)$ is the digraph $G_M = (V, E)$, where $V = \{1, \dots, n\}$ and $(i, j) \in E$ iff some rule in br_i has an atom $(j{:}p)$ in the body.*

The first optimization technique is made up of three graph operations. We get a coarse view of the topology by splitting it into *biconnected components*, which form a *tree representation* of the MCS. Then, edge removal techniques yield acyclic structures.

In the sequel, we will use standard terminology from graph theory (see [4]); graphs are directed by default. For any graph G and set $S \subseteq E(G)$ of edges, we denote by $G \backslash S$ the subgraph of G that has no edges from S. For a vertex $v \in V(G)$, we denote by $G \backslash v$ the subgraph of G induced by $V(G) \backslash \{v\}$. 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.

Example 3. The topology G_M of M in Ex. 2 is shown in Fig. 1b. It has two cut vertices, viz. 3 and 4; thus the block tree $T(G_M)$ (Fig. 1c) 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.

Pruning. In acyclic topologies, like the triangle presented in the previous section, we can exploit a minimal graph representation to avoid unnecessary calls between contexts. Namely, the *transitive reduction* of the graph G_M; recall that the transitive reduction of a digraph G is the graph G^- with the smallest set of edges whose transitive closure equals the one of G. Note that G^- is unique if G is acyclic.

Another essential part of our optimization strategy is to break cycles by removing edges from topologies. To this end, we use ear decompositions of cyclic graphs. A block may have multiple cycles which are not necessarily strongly connected, thus we first decompose cyclic blocks into their strongly connected components. The topological sort of these components yield a sequence of nodes $r_1, \dots, r_s$ that are used as entry points to each component. The next step is to break cycles. An *ear decomposition* of a strongly connected graph G *rooted at a node* r is a sequence $P = \langle P_0, \dots, P_m \rangle$ of subgraphs of G such that (i) $G = P_0 \cup \dots \cup P_m$, (ii) P_0 is a simple cycle (i.e., has no repeated edges or vertices) with $r \in V(P_0)$, and (iii) each P_i ($i > 0$) is a non-trivial path (without cycles) whose endpoints are in $P_0 \cup \dots \cup P_{i-1}$, but the other nodes are not. Let $cb(G, P)$ be the set of edges containing (ℓ, r) from P_0 and the last edge (ℓ, t) from each P_i, $i > 0$.

Example 4. Block B_1 of $T(G_M)$ is acyclic, and the transitive reduction gives B_1^- with edges $\{(1, 2), (2, 3), (3, 4)\}$. B_2 is cyclic, and $\langle B_2 \rangle$ is the only ear decomposition rooted at 4; removing $cb(B_2, \langle B_2 \rangle) = \{(5, 4)\}$, we obtain B_2' with edges $\{(4, 5)\}$. B_3 is acyclic and already reduced. Fig. 1c shows the final result (dotted edges are removed).

The graph-theoretic concepts introduced here, in particular the transitive reduction of acyclic blocks and the ear decomposition of cyclic blocks, are used to implement the first optimization of MCS evaluation outlined above. Intuitively, given the transitive reduction B^- of an acyclic block $B \in \mathcal{B}$, and a total order on $V(B^-)$ that extends B^-, one can evaluate the respective contexts in reverse order for computing partial equilibria at some context C_k: the first context simply computes its local belief sets which—represented as a set of partial belief states $\mathcal{S}_0$—constitutes an initial set of partial belief states $\mathcal{T}_0$. In any iterative Step i, $\mathcal{T}_{i-1}$ is updated by joining it with the local belief sets $\mathcal{S}_i$ of the context under consideration. Given $\mathcal{T}_k$ (after updating with $\mathcal{S}_k$) for context C_k, it holds that $\mathcal{T}_k|_{V^*(k)}$ is the set of partial equilibria at C_k (restricted to contexts in $V(B^-)$).

For cyclic blocks, one can in principle proceed as above; however, any context C_k accessing beliefs from a context C_i that precedes it in the given total order, has to temporarily consider all possible belief sets for C_i in $\mathcal{S}_k$. As a consequence, the above relation to partial equilibria can only be established after visiting all contexts that have been temporarily considered in previous steps.

Refined recursive import. Next, we define the second part of our optimization strategy which handles minimization of information needed for transmission between two neighboring contexts C_i and C_j. For this purpose, we refine the notion of recursive import interface in a context w.r.t. a particular neighbor, and a given (sub-)graph.

Definition 7. *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* $G\}$.

Intuitively, if a context is a cut vertex c in G_M, one can drop all entries S_i $(i \neq c)$ from the partial belief states computed at c, and pass this result to the parent block of c in $T(G_M)$, without compromising the computation of compatible (restricted) belief sets at the parent. Recursive import interfaces w.r.t. blocks in G_M reflect this property, which can be exploited for minimizing the information transmitted.

Algorithms. Alg. 1 and 2 combine the optimization techniques outlined above. Intuitively, OptimizeTree takes a block tree T as input together with parent cut vertex c_p and root cut vertex c_r. It traverses T in a DFS-way and calls OptimizeBlock on every block. The result of the latter calls are removed edges F; after all blocks have been processed, the final result of OptimizeTree is a pair of all edges removed from blocks in T, and a labelling v for the remaining edges. OptimizeBlock takes a graph G and calls subroutine CycleBreaker for cyclic G, which decomposes G into its strongly connected components, creates an ear decomposition P for each component G_c, and breakes cycles by removing edges $cb(G_c, P)$. For the resulting acyclic subgraph of G (or if G was already acyclic), OptimizeBlock computes the transitive reduction G^-. All edges removed from G are returned. OptimizeTree continues computing the labelling v for the remaining edges, building on the recursive import interface, but keeping relevant interface variables of child cut vertices and removed edges. It can be shown that:

Proposition 1. *For any context C_k in an MCS M,* OptimizeTree$(T(G_M), k, k)$ *returns a pair (F, v) such that (i) the subgraph G of $G_M \backslash F$ induced by $IC(k)$ is acyclic, and (ii) for all $(i, j) \in E(G)$, $v(i, j) = V^*(i, j)_G$.*

Algorithm 1: $\mathsf{OptimizeTree}(T = (\mathcal{B} \cup \mathcal{C}, \mathcal{E}), c_p, c_r)$

```
**Input**: $T$: block tree, $c_p$: identifies level in $T$, $c_r$: identifies level above $c_p$
**Output**: $F$: removed edges from $\bigcup \mathcal{B}$, $v$: labels for $(\bigcup \mathcal{B})\backslash F$
$\mathcal{B}' := \emptyset,\ F := \emptyset,\ v := \emptyset$          // initialize siblings $\mathcal{B}'$ and return values
**if** $c_p = c_r$ **then** $\mathcal{B}' := \{B \in \mathcal{B} \mid c_r \in V(B)\}$ **else** $\mathcal{B}' := \{B \in \mathcal{B} \mid (B, c_p) \in \mathcal{E}\}$
**foreach** *sibling block* $B \in \mathcal{B}'$ **do**          // sibling blocks $B$ of parent $c_p$
    $E := \mathsf{OptimizeBlock}(B, c_p)$          // prune block
    $\mathcal{C}' := \{c \in \mathcal{C} \mid (B, c) \in \mathcal{E} \wedge c \neq c_p\}$          // children cut vertices of $B$
    $B' := B\backslash E, F := F \cup E$
    **foreach** *edge* $(i, j)$ *of* $B'$ **do**          // setup interface of pruned $B$
        $v(i, j) := V^*(i, j)_{B'} \cup \bigcup_{c \in \mathcal{C}'} V^*(c_p)|_{\Sigma_c} \cup \bigcup_{(\ell,t) \in E} V^*(c_p)|_{\Sigma_t}$
    **foreach** *child cut vertex* $c \in \mathcal{C}'$ **do**          // accumulate children
        $(F', v') := \mathsf{OptimizeTree}(T\backslash B, c, c_p)$
        $F := F \cup F', v := v \cup v'$
**return** $(F, v)$
```

Algorithm 2: $\mathsf{OptimizeBlock}(G\colon \text{graph}, r\colon \text{context id})$

```
$F := \emptyset$
**if** *$G$ is cyclic* **then** $F := \mathsf{CycleBreaker}(G, r)$     // ear decomp. of strong components
Let $G^-$ be the transitive reduction of $G\backslash F$
**return** $E(G) \setminus E(G^-)$          // removed edges from $G$
```

Given G_M, the block tree graph $T(G_M)$ can be constructed in linear time; transitive reductions thereof can be computed in quadratic time. Since no other operation of the algorithm exceeds this bound, the following holds.

Proposition 2. *For any context C_k in an MCS M,* $\mathsf{OptimizeTree}(T(G_M), k, k)$ *runs in time polynomial (quadratic) in the size of $T(G_M)$ resp. G_M.*

Given the topology of an MCS, we need to represent a stripped version of it which contains both the minimal dependencies between contexts and interface variables that need to be transferred between contexts. This representation will be a *query plan* that can be used for execution processing. Syntactically, query plans have the following form.

Definition 8 (Query Plan). *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 particular, for any MCS M and context C_k of M, the labeled graph $\Pi_k = (V(G), E(G)\backslash F, v)$ is a query plan of M w.r.t. C_k, where G is the subgraph of G_M induced by $IC(k)$ and $(F, v) = \mathsf{OptimizeTree}(T(G_M), k, k)$. This query plan is in fact effective; we show how to use it for MCS evaluation.

4 Nonmonotonic MCS Evaluation with Query Plans

Given an MCS M and a starting context C_k, we aim at finding all projected partial equilibria of M w.r.t. C_k in a distributed way. To this end, we design an algorithm called

DMCSOPT that is based on the algorithm DMCS in [7], but exploits properties of the optimization techniques described above.

As a by-product, we obtain a simplification, because explicit cycle breaking is not needed. At each context node, an instance of DMCSOPT runs independently and communicates with other instances for exchanging sets of partial belief states. This provides a method for distributed model building, such that DMCSOPT can be deployed to any MCS where appropriate solvers for the respective context logics are available. The main feature of DMCSOPT is that it computes projected partial equilibria based on a query plan. This can be exploited for specific tasks like, e.g., local query answering or consistency checking. When computing projected partial equilibria, the information communicated between contexts is minimized, keeping communication cost low.

In the sequel, we present a basic version of the algorithm, abstracting from low-level implementation issues. The idea is as follows: we start with context C_k and traverse a given query plan by expanding the outgoing edges of that plan at each context, like in a depth-first search, until a leaf context is reached. A leaf context C_i simply computes its local belief sets, transforms all belief sets into partial belief states, and returns this result to its parent. If the leaf C_i contains $(j : p)$ in bodies of bridge rules such that there is no context C_j to visit in the query plan—this means we broke a cycle by removing the last edge to C_j—, all possible truth assignments to the import interface to C_j are considered.

The result of any context C_i is a set of partial belief states, which amounts to the join, i.e., the consistent combination, of its local belief sets with the results of its neighbors; the final result is obtained from C_k. To keep re-computation and recombination of belief states with local belief sets at a minimum, partial belief states are cached in every context.

Alg. 3 shows our distributed algorithm, DMCSOPT, with its instance at a context C_k that runs in a background process (or daemon in Unix). On input of the id c of a predecessor context (which the process awaits), it proceeds based on an (acyclic) query plan Π_r w.r.t. context C_r, i.e., the starting context of the system. The algorithm maintains a cache $cache(k)$ at C_k, which is kept persistent by the background process. It uses the following helper functions:

- C_i.DMCSOPT(c): send id c to DMCSOPT at context C_i and wait for its result.
- guess(V): guess all possible truth assignments for the interface variables V.
- lsolve(S) (Alg. 4): given a partial belief state S, augment kb_k with all heads from bridge rules br_k applicable w.r.t. S (=: kb'_k), compute local belief sets by $\mathbf{ACC}(kb'_k)$, and merge them with S; return the resulting set of partial belief states.

The steps of Alg. 3 are explained as follows:

(a)+(b) check the cache, and if it is 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

Algorithm 3: DMCSOPT(c : context id of predecessor) at $C_k = (L_k, kb_k, br_k)$

```
    Data: $\Pi_r$: 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
        $\mathcal{T} := \{(\epsilon, \dots, \epsilon)\}$
(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
            $\mathcal{T} :=$ guess($v(c, k)$) $\bowtie \mathcal{T}$    // guess for removed dependencies in $\Pi_r$
(d)     foreach $T \in \mathcal{T}$ do $\mathcal{S} := \mathcal{S} \cup$ 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 4: 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 $\{(S_1, , \dots, S_{k-1}, T_k, S_{k+1}, \dots, S_n) \mid T_k \in \mathbf{T}\}$
```

(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 $T(G_M)$.

The following proposition shows that DMCSOPT is sound and complete.

Proposition 3. *Let C_k be a context of an MCS M, let Π_k be the query plan as defined above and let $V = \{p \in v(k, j) \mid (k, j) \in E(\Pi_k)\}$. Then, (i) for each $S' \in C_k$.*DMCSOPT*(k), there exists a partial equilibrium S of M w.r.t. C_k such that $S' = S|_V$; and (ii) for each partial equilibrium S of M w.r.t. C_k, there exists an $S' \in C_k$.*DMCSOPT*(k) such that $S' = S|_V$.*

5 Implementation and Experimental Results

We present some results for a SAT-solver based prototype implementation of DMCSOPT under Ubuntu Linux 9.10, written in C++. Full details of the experiments and the implementation are available at `http://www.kr.tuwien.ac.at/research/systems/dmcs/`. The host system was using a Pentium Core2 Duo 2.53GHz processor with 4GB RAM. Based on generated benchmarks, we compare the average response time and the median of the number of results of DMCSOPT to our implementation of DMCS. The processes encoding the contexts communicated over local TCP/IP connections. We used *clasp* 1.3.3 as a SAT solver, which accepts DIMACS CNF input [8]. Specifically, all generated instantiations of MCS have contexts with ASP logics. The translation defined in [7] is used to create SAT instances at all contexts and *clasp* builds all models.

For initial experimentation, we created random MCS instances with various fixed topologies that should resemble the context dependencies of realistic scenarios. We have generated instances with ordinary (D) and zig-zag (Z) diamond stack, house stack (H), ring (R), and binary tree topologies. A diamond stack combines multiple diamonds in a row (stacking m diamonds in a tower of $3m + 1$ contexts). Ordinary diamonds have, in contrast to zig-zag diamonds like block B_1 in Fig. 1c, no connection between the two

	n	A_ϕ	$A_\bowtie$	$A_\leftrightarrow$	A_Σ (σ)	# (σ)	B_ϕ	$B_\bowtie$	$B_\leftrightarrow$	B_Σ (σ)	# (σ)
D	13	0.9	0.0	0.0	1.0 (0.2)	28 (17.6)	0.8	8.4	0.0	9.4 (5.5)	3136 (3155.8)
	25	11.2	0.5	0.0	12.8 (1.3)	17 (18.9)	—				
	31	51.1	3.7	0.0	59.5 (8.9)	58 (49.7)	—				
R	10	0.1	0.0	0.0	0.1 (0.0)	3.5 (3.4)	0.1	0.0	0.0	0.2 (0.1)	300 (694.5)
	13	0.1	0.0	0.0	0.2 (0.1)	6 (1.2)	0.1	1.5	1.9	3.9 (5.3)	5064 (21523.8)
	301	4.1	0.1	2.1	10.2 (2.2)	8 (4.9)	—				
Z	13	0.6	0.1	0.0	0.7 (0.2)	34 (41.8)	5.5	4.2	0.0	11.5 (4.0)	3024 (1286.8)
	151	8.9	22.3	0.4	32.2 (7.3)	33 (28.5)	—				
	301	21.6	99.5	1.7	124.3 (20.6)	22 (41.4)	—				
H	9	0.2	0.0	0.0	0.2 (0.0)	28 (44.4)	1.1	0.9	0.0	2.0 (1.3)	684 (1308.0)
	101	1.8	0.3	0.3	3.8 (1.0)	48 (76.6)	—				
	301	7.8	2.0	2.4	25.1 (8.7)	38 (34.2)	—				

Table 1: Runtime for DMCSOPT (A_x) and DMCS (B_x), timeout 180 secs (—)

middle contexts. 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); house stacks are subsequently built up by using the basement nodes as ridges for the next houses (thus, m houses have $4m + 1$ contexts). Binary trees grow balanced, i.e., every level is complete except for the last level, which grows from the left-most context.

A parameter setting (n, s, b, r) specifies (i) the number n of contexts, (ii) the local alphabet size $|\Sigma_i| = s$ (each C_i has a random ASP program on s atoms with 2^k answer sets, $0 \leq k \leq s/2$), (iii) the maximum interface size b (number of atoms exported), and (iv) the maximum number r of bridge rules per context, each having ≤ 2 body literals.

Table 1 shows some experimental results for parameter settings $(n, 10, 5, 5)$, where n varies between 10 and 301. Each subtable $X \in \{D, H, R, Z\}$ shows runs with growing number of contexts and correspond to a benchmark topology from above. Each row displays the average of total running time over 10 generated instances for DMCSOPT in A_Σ compared to DMCS in B_Σ. For each context in the instances, the columns A_x and B_x for $x \in \{\phi, \bowtie, \leftrightarrow\}$ show the average over the (i) total time spent in the SAT solver (ϕ), (ii) total time needed to join the belief states ($\bowtie$), and (iii) total time used to transfer the belief states between the contexts ($\leftrightarrow$). The $\#$ columns show the median of numbers of projected partial equilibria computed at C_1 (initiated by sending the request 1 to C_1 for DMCSOPT with a fixed query plan Π_1, respectively $V^*(1)$ to C_1 for DMCS). Entries in parenthesis (σ) show the standard deviation of A_Σ and $\#$.

The optimizations that can be applied in the topologies are quite diverse. In ordinary diamond stacks and in binary trees, we cannot remove edges, as the topologies are equal to their transitive reductions. But we can refine the import interface at each sub-diamond (every fourth context is a cut vertex), thus the partial belief states eventually computed just contain entries for the first four contexts. House stacks have a triangle element as roof (Fig. 1a) 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. The refinement of the import interface in binary trees is even more drastic, as every non-leaf context is a cut vertex, and we can restrict to import interfaces between two neighboring contexts. In the ring topology, we can remove 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. In zig-zag diamond stacks, we remove in each block two edges to obtain the transitive reduction and update the recursive import interface accordingly.

Evaluating the MCS instances with DMCSOPT compared to DMCS yields a drastic improvement in response time. Stacking multiple diamonds in a tower models hard instances with many joins. This is reflected in the ratio of running time to result size in DMCS. Still, DMCSOPT could handle much larger instances. The ring topology shows a similar increase in scalability. Thanks to the refined interface, the system size can be increased dramatically; the runs for n=301 took only a few seconds. House instances use a complex topology with cycles in each block, and each cycle has two entry points. DMCS is already having a hard time evaluating instances with two houses, whereas DMCSOPT could evaluate ten houses in a reasonable time due to the localized computation of the belief states. Also for zig-zag diamond stacks, the optimizations effect that DMCSOPT can run substantially larger systems, with hundreds of contexts; DMCS has an early breakdown at n=16.

Comparing ordinary diamonds (D) to zig-zag diamonds (Z), one can notice a large gap in the size n of the MCS that can be handled. This is explained by the transitive reduction that can be applied to zig-zag diamonds, essentially resulting in a chain of contexts such that each context can take the partial belief states of its single neighbor and simply add its local beliefs. Diamonds cannot be further optimized and additionally need to join the results from their neighbors. We omit detailed outcomes for binary tree topology tests here. However, we noticeably could evaluate an instance with even n=600 contexts in 175.6 seconds (#=4) with our parameter setting; setting the timeout to 12 minutes, DMCS runs out of memory for instances with n=22 during belief state joining.

The A_ϕ/B_ϕ and $A_\bowtie/B_\bowtie$ columns show that most time is spent in the SAT solver in D, R, and H instances, whereas Z uses most time in combining the models. This is explained by the need to guess many interface variables in those instances, as we need to cater for the interface of the removed edges. $A_\leftrightarrow$ reveals that almost no time is used to transfer the belief states, even as n grows, thus showing the effectiveness of the projection over blocks, as only a small amount of models have to be shipped. In $B_\leftrightarrow$ we can see that already small instances produce many models and the price is that the network is under heavy load.

6 Related Work and Conclusion

In [13], the authors described evaluation of monotone MCS with classical theories using SAT solvers for the contexts in parallel. They used a (co-inductive) fixpoint strategy to check MCS satisfiability, where a centralized process iteratively combines results of the SAT solvers. Apart from being not truly distributed, an extension to nonmonotonic MCS is non-obvious; also, no caching was used. Distributed tableaux algorithms for reasoning in distributed ontologies are defined in [14, 15]. They can be used to decide consistency of distributed description logic knowledge bases, provided that the distributed TBox is acyclic. The DRAGO system is an implementation of this approach.

The authors of [1] presented a framework of peer-to-peer inference systems. Local theories of propositional clause sets share atoms, and a special algorithm can be used for consequence finding. As we pursue the dual problem of model building, application for our needs is not straightforward. Similarly, [3] developed a distributed algorithm for query evaluation in a MCS framework based on defeasible logic. Here, contexts are built using defeasible rules, and the algorithm can determine for a given literal l three values: whether l is (not) a logical conclusion of the MCS, or whether it cannot be proved that l is a logical conclusion. Again, applying this approach to model building is not easy.

Biconnected components are used in [2] to decompose constraint satisfaction problems. The decomposition is used to localize the computation of a single solution in the components of undirected constraint graphs. Likened to our approach, we are based on directed dependencies, which allows us to use a query plan for MCS evaluation.

We have presented techniques and algorithms for decomposing, pruning, and cycle breaking of dependencies in nonmonotonic multi-context systems. Based on this, we have devised an algorithm, which uses a query plan to compute all partial equilibria of such a system. A prototypical implementation of this approach shows promising experimental results. They are a substantial improvement and encourage to research further algorithms and methods for evaluation of distributed MCS, such that efficient platforms for distributed nonmonotonic reasoning applications will become available.

References

1. Adjiman, P., Chatalic, P., Goasdoué, F., Rousset, M.C., Simon, L.: Distributed reasoning in a peer-to-peer setting: Application to the semantic web. J. Artif. Intell. Res. 25, 269–314 (2006)
2. Baget, J.F., Tognetti, Y.: Backtracking through biconnected components of a constraint graph. In: IJCAI'01. pp. 291–296. Morgan Kaufmann (2001)
3. Bikakis, A., Antoniou, G., Hassapis, P.: Strategies for contextual reasoning with conflicts in ambient intelligence. Knowl Inf Syst. (April 2010), published online: 9 April 2010
4. Bondy, A., Murty, U.S.R.: Graph Theory. Springer (2008)
5. Brewka, G., Eiter, T.: Equilibria in heterogeneous nonmonotonic multi-context systems. In: AAAI'07. pp. 385–390. AAAI Press (July 2007)
6. Buccafurri, F., Caminiti, G.: Logic programming with social features. Theory Pract. Log. Program. 8(5–6), 643–690 (November 2008)
7. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Distributed nonmonotonic multi-context systems. In: KR'10 pp. 60–70. AAAI Press (May 2010)
8. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI'07. pp. 386–392. AAAI Press (2007)
9. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. 9(3–4), 365–385 (1991)
10. Ghidini, C., Giunchiglia, F.: Local models semantics, or contextual reasoning = locality + compatibility. Artif. Intell. 127(2), 221–259 (2001)
11. Giunchiglia, F., Serafini, L.: Multilanguage hierarchical logics or: How we can do without modal logics. Artif. Intell. 65(1), 29–70 (1994)
12. McCarthy, J.: Notes on formalizing context. In: IJCAI'93. pp. 555–562 (1993)
13. Roelofsen, F., Serafini, L., Cimatti, A.: Many Hands Make Light Work: Localized Satisfiability for Multi-Context Systems. In: ECAI'04. pp. 58–62. IOS Press (2004)
14. Serafini, L., Borgida, A., Tamilin, A.: Aspects of distributed and modular ontology reasoning. In: IJCAI'05. pp. 570–575. AAAI Press (2005)
15. Serafini, L., Tamilin, A.: Drago: Distributed reasoning architecture for the semantic web. In: ESWC'05. pp. 361–376. Springer (May 2005)