
Minh Dao-Tran

Advisors: Thomas Eiter
          Michael Fink

Abteilung Wissensbasierte Systeme
Institut für Informationssysteme
Technische Universität Wien

Doctoral Defense — March 24, 2014

supported by the Austrian Science Fund (FWF) project P20841
Overview

- Introduction to Multi-context Systems
- Basic Algorithm DMCS to Evaluate MCS
- Topological-based Optimized Algorithm DMCSOPT
- Streaming Models with DMCS-STREAMING
- Experimental Evaluation: Setup and Analysis
- Outlook
Multi-Context Systems

- What is a multi-context system? \( M = (C_1, \ldots, C_n) \)
  - a collection of contexts \( C_1, \ldots, C_n \)

- What is a context?
  \( C_i = (L_i, kb_i, br_i) \)
  - a logic \( L_i \)
  - the context’s knowledge base \( kb_i \)
  - a set \( br_i \) of bridge rules
Multi-Context Systems

- What is a multi-context system? \( M = (C_1, \ldots, C_n) \)
  - a collection of contexts \( C_1, \ldots, C_n \)

- What is a context?
  \( C_i = (L_i, kb_i, br_i) \)
  - a logic \( L_i \)
  - the context’s knowledge base \( kb_i \)
  - a set \( br_i \) of bridge rules

- What is a logic?
  \( L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L) \)
  - set \( \mathbf{KB}_L \) of well-formed knowledge bases
  - set \( \mathbf{BS}_L \) of possible belief sets
  - acceptability function \( \mathbf{ACC}_L : \mathbf{KB}_L \rightarrow 2^{\mathbf{BS}_L} \)
    Which belief sets are accepted by a knowledge base?
MCS Example
MCS Example - Encoding

\[
\begin{align*}
\text{at}_{col}(X) & \leftarrow \text{see}_{col}(X), \\
\text{not} \left\{ \text{at}_{col}(X) \right\} & \cup R \cup \left\{ \text{not} \left\{ \text{see}_{col}(X) \right\} \right\}, \\
\text{at}_{row}(X) & \leftarrow \left\{ \text{row}(X) \right\}, \\
\text{not} \left\{ \text{at}_{row}(X) \right\} & \cup \text{covered}_{row}(X) \left\{ \text{not} \left\{ \text{row}(X) \right\} \right\}, \\
\text{at}_{col}(X) & \leftarrow \left\{ \text{col}(X) \right\}, \\
\text{not} \left\{ \text{at}_{col}(X) \right\} & \cup \text{covered}_{col}(X) \left\{ \text{not} \left\{ \text{col}(X) \right\} \right\}, \\
\text{at}_{row}(Y), X \neq Y & \left\{ \text{row}(X) \right\}, \\
\text{not} \left\{ \text{at}_{row}(X) \right\} & \cup \text{covered}_{row}(X) \left\{ \text{not} \left\{ \text{row}(X) \right\} \right\}, \\
\text{at}_{col}(Y), X \neq Y & \left\{ \text{col}(X) \right\}, \\
\text{not} \left\{ \text{at}_{col}(X) \right\} & \cup \text{covered}_{col}(X) \left\{ \text{not} \left\{ \text{col}(X) \right\} \right\}, \\
\text{row}(1), \text{row}(2), \text{row}(3), \text{col}(1), \text{col}(2), \text{col}(3) & \\
\end{align*}
\]

\[\text{C}_1, \text{C}_2\]
MCS Example - Encoding

where:  \[ R = \begin{cases} 
  \text{joker\_in} & \leftarrow \text{at\_row}(X). \\
  \text{joker\_in} & \leftarrow \text{at\_col}(X). \\
  \text{at\_row}(X) & \leftarrow \text{joker\_in}, \text{row}(X), \text{not} \neg \text{at\_row}(X). \\
  \neg \text{at\_row}(X) & \leftarrow \text{joker\_in}, \text{row}(X), \text{at\_row}(Y), X \neq Y. \\
  \text{at\_col}(X) & \leftarrow \text{joker\_in}, \text{col}(X), \text{not} \neg \text{at\_col}(X). \\
  \neg \text{at\_col}(X) & \leftarrow \text{joker\_in}, \text{col}(X), \text{at\_col}(Y), X \neq Y. \\
  \text{row}(1). \text{row}(2). \text{row}(3). \\
  \text{col}(1). \text{col}(2). \text{col}(3). 
\end{cases} \]
MCS Example - Encoding

\[
\begin{align*}
C_1 & \quad \{ \text{at}_\text{col}(X) \leftarrow \text{see}_\text{col}(X). \} \cup R \cup \{ \neg \text{see}_\text{col}(2), \neg \text{see}_\text{col}(3). \} \\
C_2 & \quad \{ \text{at}_\text{row}(X) \leftarrow \text{see}_\text{row}(X). \} \cup R \cup \{ \text{see}_\text{row}(1). \}
\end{align*}
\]

where: \[ R = \{ \text{joker}_\text{in} \leftarrow \text{at}_\text{row}(X). \]
\[ \text{joker}_\text{in} \leftarrow \text{at}_\text{col}(X). \]
\[ \text{at}_\text{row}(X) \leftarrow \text{joker}_\text{in}, \text{row}(X), \text{not} \neg \text{at}_\text{row}(X). \]
\[ \neg \text{at}_\text{row}(X) \leftarrow \text{joker}_\text{in}, \text{row}(X), \text{at}_\text{row}(Y), X \neq Y. \]
\[ \text{at}_\text{col}(X) \leftarrow \text{joker}_\text{in}, \text{col}(X), \text{not} \neg \text{at}_\text{col}(X). \]
\[ \neg \text{at}_\text{col}(X) \leftarrow \text{joker}_\text{in}, \text{col}(X), \text{at}_\text{col}(Y), X \neq Y. \]
\[ \text{row}(1). \text{row}(2). \text{row}(3). \]
\[ \text{col}(1). \text{col}(2). \text{col}(3). \]
MCS Example - Encoding

where: \( R = \)

\[
\begin{align*}
\text{joker\_in} & \leftarrow \text{at\_row}(X). \\
\text{joker\_in} & \leftarrow \text{at\_col}(X). \\
\text{at\_row}(X) & \leftarrow \text{joker\_in}, \text{row}(X), \neg \text{at\_row}(X). \\
\neg \text{at\_row}(X) & \leftarrow \text{joker\_in}, \text{row}(X), \text{at\_row}(Y), X \neq Y. \\
\text{at\_col}(X) & \leftarrow \text{joker\_in}, \text{col}(X), \neg \text{at\_col}(X). \\
\neg \text{at\_col}(X) & \leftarrow \text{joker\_in}, \text{col}(X), \text{at\_col}(Y), X \neq Y. \\
\text{row}(1). \text{row}(2). \text{row}(3). \\
\text{col}(1). \text{col}(2). \text{col}(3). 
\end{align*}
\]

\[
\begin{align*}
\{ \text{at\_col}(X) & \leftarrow \text{see\_col}(X). \\
\neg \text{at\_col}(X) & \leftarrow \neg \text{see\_col}(X). \} \cup R \cup \{ \neg \text{see\_col}(2). \neg \text{see\_col}(3). \}
\end{align*}
\]

\[
\begin{align*}
\text{at\_row}(X) & \leftarrow (2 : \text{at\_row}(X)). \\
\neg \text{at\_row}(X) \lor \text{covered\_row}(X) & \leftarrow \neg \text{at\_row}(X). \\
\neg \text{at\_row}(X) \lor \text{covered\_row}(X) & \leftarrow \neg \text{at\_row}(X), (2 : \text{see\_row}(X)), (1 : \text{row}(X)). \\
\end{align*}
\]

\[
\begin{align*}
\{ \text{at\_row}(X) & \leftarrow \text{see\_row}(X). \\
\neg \text{at\_row}(X) & \leftarrow \neg \text{see\_row}(X). \} \cup R \cup \{ \text{see\_row}(1). \}
\end{align*}
\]

\[
\begin{align*}
\text{joker\_in} & \leftarrow \text{at\_row}(X). \\
\text{joker\_in} & \leftarrow \text{at\_col}(X). \\
\text{at\_row}(X) & \leftarrow \text{joker\_in}, \text{row}(X), \neg \text{at\_row}(X). \\
\neg \text{at\_row}(X) & \leftarrow \text{joker\_in}, \text{row}(X), \text{at\_row}(Y), X \neq Y. \\
\text{at\_col}(X) & \leftarrow \text{joker\_in}, \text{col}(X), \neg \text{at\_col}(X). \\
\neg \text{at\_col}(X) & \leftarrow \text{joker\_in}, \text{col}(X), \text{at\_col}(Y), X \neq Y. \\
\text{row}(1). \text{row}(2). \text{row}(3). \\
\text{col}(1). \text{col}(2). \text{col}(3). 
\end{align*}
\]
Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in BS_{L_i}$ makes certain bridge rules applicable ... so that we can add their heads into the $kb_i$ of the contexts $S$ is an equilibrium iff each context plus these heads accepts $S_i$.

Equilibrium condition: $S_i \in ACC(kb_i \cup H_i)$ for all $C_i$
Equilibrium semantics: a belief state \( S = (S_1, \ldots, S_n) \) with \( S_i \in \text{BS}_{L_i} \)

... makes certain bridge rules applicable

... so that we can add their heads into the \( kb_i \) of the contexts

\( S \) is an equilibrium iff each context plus these heads accepts \( S_i \).

Equilibrium condition: \( S_i \in \text{ACC}(kb_i \cup H_i) \) for all \( C_i \)
Equilibrium semantics: a belief state \( S = (S_1, \ldots, S_n) \) with \( S_i \in BS_L \)
... makes certain bridge rules applicable
... so that we can add their heads into the \( kb_i \) of the contexts
\( S \) is an equilibrium iff each context plus these heads accepts \( S_i \).

Equilibrium condition: \( S_i \in ACC(kb_i \cup H_i) \) for all \( C_i \)
Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in \text{BS}_{L_i}$

... makes certain bridge rules applicable

... so that we can add their heads into the $kb_i$ of the contexts

$S$ is an equilibrium iff each context plus these heads accepts $S_i$.

Equilibrium condition: $S_i \in \text{ACC}(kb_i \cup H_i)$ for all $C_i$
Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in BS_{Li}$ makes certain bridge rules applicable so that we can add their heads into the $kb_i$ of the contexts. $S$ is an equilibrium iff each context plus these heads accepts $S_i$.

Equilibrium condition: $S_i \in ACC(kb_i \cup H_i)$ for all $C_i$
Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in BS_{Li}$ ... makes certain bridge rules applicable
... so that we can add their heads into the $kb_i$ of the contexts
$S$ is an equilibrium iff each context plus these heads accepts $S_i$.
Equilibrium condition: $S_i \in ACC(kb_i \cup H_i)$ for all $C_i$
Why are MCSs interesting?

Distributedness / Heterogeneity / Nonmonotonicity
Why are MCSs interesting?

Distributedness / Heterogeneity / Nonmonotonicity

⇒ Power to model real life applications:
  ▶ collaboration between business partners,
  ▶ medical applications,
  ▶ reasoning on the web,
  ▶ ...
Why are MCSs interesting?

Distributedness / Heterogeneity / Nonmonotonicity

⇒ Power to model real life applications:
  ▶ collaboration between business partners,
  ▶ medical applications,
  ▶ reasoning on the web,
  ▶ ... 

Thus, algorithms to evaluate MCSs (compute equilibria) are of special interest!
Evaluation of MCSs before this thesis

- Related works on distributed systems: either not truly distributed or homogeneous
  - Distributed Constraints Satisfaction Problems [Yokoo and Hirayama, 2000]
  - DisSAT: finding a single model [Hirayama and Yokoo, 2005]
  - Parallel algorithm for evaluating monotonic MCS [Roelofsen et al., 2004]
  - Distributed Ontology Reasoning (DRAGO) [Serafini et al., 2005]
  - Distributed reasoning in peer-to-peer setting [Adjiman et al., 2006]
  - Distributed query evaluation in “MCS” based on defeasible logic [Bikakis et al., 2010]
Evaluation of MCSs before this thesis

- Related works on distributed systems: either not truly distributed or homogeneous
  - Distributed Constraints Satisfaction Problems [Yokoo and Hirayama, 2000]
  - DisSAT: finding a single model [Hirayama and Yokoo, 2005]
  - Parallel algorithm for evaluating monotonic MCS [Roelofsen et al., 2004]
  - Distributed Ontology Reasoning (DRAGO) [Serafini et al., 2005]
  - Distributed reasoning in peer-to-peer setting [Adjiman et al., 2006]
  - Distributed query evaluation in “MCS” based on defeasible logic [Bikakis et al., 2010]

- For distributed nonmonotonic MCS:
  - Only one proposal for evaluating MCSs in a centralized way using *hex*-programs
  - No implementation available
Evaluation of MCSs before this thesis

- Related works on distributed systems: either not truly distributed or homogeneous
  - Distributed Constraints Satisfaction Problems [Yokoo and Hirayama, 2000]
  - DisSAT: finding a single model [Hirayama and Yokoo, 2005]
  - Parallel algorithm for evaluating monotonic MCS [Roelofsen et al., 2004]
  - Distributed Ontology Reasoning (DRAGO) [Serafini et al., 2005]
  - Distributed reasoning in peer-to-peer setting [Adjiman et al., 2006]
  - Distributed query evaluation in “MCS” based on defeasible logic [Bikakis et al., 2010]

- For distributed nonmonotonic MCS:
  - Only one proposal for evaluating MCSs in a centralized way using $hex$-programs
  - No implementation available

- Obstacles:
  - Abstraction of contexts
  - Information hiding and security aspects
  - Lack of system topology
  - Cyclic dependency between contexts
Towards Evaluation of MCSs

Our aims:

- Algorithms for evaluating equilibria of MCSs in a truly distributed way
- Optimization techniques
- Prototype implementation
- Benchmarking
Towards Evaluation of MCSs

Our aims:

- Algorithms for evaluating equilibria of MCSs in a truly distributed way
- Optimization techniques
- Prototype implementation
- Benchmarking

We fulfill these goals by exploiting and adapting methods from distributed systems area, with special care for MCSs:

- Dependencies between contexts
- Representation of partial knowledge
- Combination/join of local results
Towards Evaluation of MCSs

Our aims:

- Algorithms for evaluating equilibria of MCSs in a truly distributed way
- Optimization techniques
- Prototype implementation
- Benchmarking

We fulfill these goals by exploiting and adapting methods from distributed systems area, with special care for MCSs:

- Dependencies between contexts
- Representation of partial knowledge
- Combination/join of local results

Support notions:

- Import Neighborhood and Closure
- Partial Belief States and Equilibria
- Joining Partial Belief States
Import Neighborhood and Closure

Import neighborhood of $C_k$

$In(k) = \{c_i \mid (c_i : p_i) \in B(r), r \in b r_k\}$
Import neighborhood of $C_k$

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

Import closure $IC(k)$ of $C_k$ is the smallest set $S$ such that
(i) $k \in S$ and
(ii) for all $i \in S$, $\text{In}(i) \subseteq S$. 

![Diagram](image)
Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} \text{BS}_i$. 

Intuitively, partial equilibria wrt. a context $C_k$ cover the reachable contexts of $C_k$. 
Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^n BS_i$.

A partial belief state of $M$ is a sequence $S = (S_1, \ldots, S_n)$, where $S_i \in BS_i \cup \{\epsilon\}$, for $1 \leq i \leq n$. 

Intuitively, partial equilibria wrt. a context $C_k$ cover the reachable contexts of $C_k$. 
Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} \text{BS}_i$

A partial belief state of $M$ is a sequence $S = (S_1, \ldots, S_n)$, where $S_i \in \text{BS}_i \cup \{\epsilon\}$, for $1 \leq i \leq n$

$S = (S_1, \ldots, S_n)$ is a partial equilibrium of $M$ w.r.t. a context $C_k$ iff for $1 \leq i \leq n$,

- if $i \in \text{IC}(k)$ then $S_i \in \text{ACC}_i(kb_i \cup \{\text{head}(r) \mid r \in \text{app}(br_i, S)\})$
- otherwise, $S_i = \epsilon$

Intuitively, partial equilibria wrt. a context $C_k$ cover the reachable contexts of $C_k$. 
Joining Partial Belief States

Join $S \bowtie T$ of belief states $S$ and $T$: like join of tuples in a database.

\[ S = \begin{array}{cccccc}
S_1 & \cdots & \epsilon & \cdots & \epsilon & \cdots & S_j & \cdots & S_n \\
\end{array} \]

\[ T = \begin{array}{cccccc}
\epsilon & \cdots & \epsilon & \cdots & T_i & \cdots & T_j & \cdots & \epsilon \\
\end{array} \]

\[ S \bowtie T = \begin{array}{cccccc}
S_1 & \cdots & \epsilon & \cdots & T_i & \cdots & S_j = T_j & \cdots & S_n \\
\end{array} \]

$S \bowtie T$ is undefined, if $\epsilon \neq S_j \neq T_j \neq \epsilon$ for some $j$.

\[ S \bowtie \mathcal{T} = \{ S \bowtie T \mid S \in \mathcal{S}, T \in \mathcal{T} \} \]
Algorithm DMCS

**Input:** an MCS $M$ and a starting context $C_k$

**Output:** all partial equilibria of $M$ wrt. $C_k$
Algorithm DMCS

Input: an MCS $M$ and a starting context $C_k$
Output: all partial equilibria of $M$ wrt. $C_k$

Requirement: solver $\text{Isolve}(S)$ for each context $C_k$ is available which computes $\text{ACC}_k(kb_k \cup \text{app}_k(S))$
Algorithm DMCS

**Input:** an MCS $M$ and a starting context $C_k$

**Output:** all partial equilibria of $M$ wrt. $C_k$

Requirement: solver $\text{lsolve}(S)$ for each context $C_k$ is available which computes $\text{ACC}_k(kb_k \cup \text{app}_k(S))$

Input parameters for DMCS:

- $V$: set of “interesting” variables (to project the partial equilibria)
- $\text{hist}$: visited path
Algorithm DMCS

Input: an MCS $M$ and a starting context $C_k$
Output: all partial equilibria of $M$ wrt. $C_k$

Requirement: solver $\text{Isolve}(S)$ for each context $C_k$ is available which computes $\text{ACC}_k(kb_k \cup \text{app}_k(S))$

Input parameters for DMCS:
- $V$: set of “interesting” variables (to project the partial equilibria)
- $\text{hist}$: visited path

Strategy: DFS-traversal of $M$ starting with $C_k$, visiting all $C_i$ for $i \in IC(k)$

Distributedness: instances of DMCS
- running at each context node,
- communicating with each other for exchanging sets of belief states
Acyclic case

Leaf context $C_k$ ($br_k = \emptyset$)

$$\text{lsolve}((\epsilon, \ldots, \epsilon)) = S$$
Acyclic case

Leaf context $C_k (br_k = \emptyset)$

Intermediate context $C_k ((i : p), (j : q)$ appear in $br_k$)

$\text{Isolve}((\epsilon, \ldots, \epsilon)) = S$

$S$
Acyclic case

Leaf context $C_k (br_k = \emptyset)$

Intermediate context $C_k ((i : p), (j : q) \text{ appear in } br_k)$

$\text{Isolve}((\epsilon, \ldots, \epsilon)) = S$

Diagram:
- $C_k$ with $S$ pointing to it.
- $C_i$ and $C_j$ with dotted lines connecting them to $C_k$.

$SI$:
- $(V, \text{hist})$ pointing to $C_k$.
Acyclic case

Leaf context $C_k (br_k = \emptyset)$

$$\text{lsolve}((\epsilon, \ldots, \epsilon)) = S$$

Intermediate context $C_k ((i : p), (j : q)$ appear in $br_k)$

$$\text{lsolve}(S_i \bowtie S_j) = S_k$$
Cycle Breaking

$\text{hist} = \{ \ldots, k, \ldots \}$

$C_k$ detects a cycle in $\text{hist}$ ▶

guesses local belief sets ▶

returns them to invoking context ▶

on the way back, partial belief states w.r.t. bad guesses will be pruned by $\triangledown\triangleleft$

▶

eventually, $C_k$ will remove wrong guesses by calling $\text{lsolve}$ on each received partial belief state
$C_k$ detects a cycle in $hist$
Cycle Breaking

\[ C_k \text{ detects a cycle in } hist \]

\[ \text{hist} = \{ \ldots, k, \ldots \} \]

- guesses local belief sets
Cycle Breaking

$C_k$ detects a cycle in $hist$

- guesses local belief sets
- returns them to invoking context
Cycle Breaking

$C_k$ detects a cycle in $hist$

- guesses local belief sets
- returns them to invoking context

- on the way back, partial belief states w.r.t. bad guesses will be pruned by $\nabla$
Cycle Breaking

$C_k$ detects a cycle in $hist$

- guesses local belief sets
- returns them to invoking context
- on the way back, partial belief states w.r.t. bad guesses will be pruned by $\bowtie$
- eventually, $C_k$ will remove wrong guesses by calling $lsolve$ on each received partial belief state
Motivation for DMCSOPT

Scalability issues with the basic evaluation algorithm DMCS

- unaware of global context dependencies, only know (local) import neighborhood

- a context $C_i$ returns a possibly huge set of partial belief states, which are the join of neighbor belief states of $C_i$ plus local belief sets
Motivation for DMCSOPT

Scalability issues with the basic evaluation algorithm DMCS

- unaware of global context dependencies, only know (local) import neighborhood

- a context $C_i$ returns a possibly huge set of partial belief states, which are the join of neighbor belief states of $C_i$ plus local belief sets

We address these issues by

- capturing inter-context dependencies (topology)
- providing a decomposition based on biconnected components
- characterizing minimal interface variables in each component
- develop the DMCSOPT algorithm which operates on query plans
Example

Problem: How to go home?

Possible solutions:
- Car: slower than train
- Train: should bring some food

Spike and Mickey have additional information from Tyke and Minnie
Problem: How to go home?

Possible solutions:
  - Car
  - Train
Problem: How to go home?

Possible solutions:
- Car: slower than train
- Train: should bring some food

Spike and Mickey have additional information from Tyke and Minnie
Minnie wants Mickey to come back as soon as possible.

\[kb_4 = \{ car_4 \lor train_4 \leftarrow \}\]
\[br_4 = \{ train_4 \leftarrow (5 : want\_sooner_5) \}\]

\[kb_5 = \{ want\_sooner_5 \leftarrow soon_5 \}\]
\[br_5 = \{ soon_5 \leftarrow (4 : train_4) \}\]
Example (ctd.)

- Spike is responsible for buying provisions, if they go by train.
- If his son Tyke is sick, then Spike must attend to him as fast as possible.

\[
kb_3 = \begin{cases} 
    car_3 \lor train_3 & \leftarrow \\
    train_3 & \leftarrow urgent_3 \\
    sandwiches_3 \lor chocolate peptide_3 & \leftarrow train_3 \\
    coke_3 \lor juice_3 & \leftarrow train_3 
\end{cases}
\]

\[
br_3 = \begin{cases} 
    urgent_3 & \leftarrow (6 : sick_6) \\
    train_3 & \leftarrow (4 : train_4) 
\end{cases};
\]

\[
kb_6 = \{ sick_6 \lor fit_6 \leftarrow \}
\]

\[
br_6 = \emptyset.
\]
Example (ctd.)

- Jerry is the leader of the group.
- Jerry is allergic to peanuts.
- Tom wants to get home somehow and doesn’t want coke.

\[
\begin{align*}
kb_1 & = \{ \text{car}_1 \leftarrow \text{not train}_1 \} \\
& \quad \cup \{ \bot \leftarrow \text{peanuts}_1 \} \\
br_1 & = \{ \text{train}_1 \leftarrow (2 : \text{train}_2), (3 : \text{train}_3) \} \\
& \quad \cup \{ \text{peanuts}_1 \leftarrow (3 : \text{chocolate_peanuts}_3) \}
\end{align*}
\]

\[
\begin{align*}
kb_2 & = \{ \bot \leftarrow \text{not car}_2, \text{not train}_2 \} \quad \text{and} \\
& \quad \{ \text{car}_2 \leftarrow (3 : \text{car}_3), (4 : \text{car}_4) \} \\
br_2 & = \{ \text{train}_2 \leftarrow (3 : \text{train}_3), (4 : \text{train}_4), \text{not } (3 : \text{coke}_3) \}
\end{align*}
\]
Jerry is the leader of the group.

Jerry is allergic to peanuts.

Tom wants to get home somehow and doesn’t want coke.

\[ kb_1 = \begin{cases} car_1 \leftarrow not \ train_1 \\ \bot \leftarrow peanuts_1 \end{cases} \]

\[ br_1 = \begin{cases} train_1 \leftarrow (2 : train_2), (3 : train_3) \\ peanuts_1 \leftarrow (3 : chocolate\_peanuts_3) \end{cases} \]

\[ kb_2 = \{ \bot \leftarrow not \ car_2, not \ train_2 \} \]

\[ br_2 = \begin{cases} car_2 \leftarrow (3 : car_3), (4 : car_4) \\ train_2 \leftarrow (3 : train_3), (4 : train_4), \\ not (3 : coke_3) \end{cases} \]

One equilibrium is \( S = (\{train_1\}, \{train_2\}, \{train_3, urgent_3, juice_3, sandwiches_3\}, \{train_4\}, \{soon_5, want\_sooner_5\}, \{sick_6\}) \)
Example (ctd.)

- Jerry decides after gathering information.
Jerry decides after gathering information.

Mickey and Spike do not want to bother the others.
A vertex $c$ of a weakly connected graph $G$ is a cut vertex, if $G \setminus c$ is disconnected.
MCS Decomposition: Block Tree

Based on cut vertices, we can decompose the MCS into a block tree: provides a “high-level” view of the dependencies (edge partitioning)
MCS Decomposition: Block Tree

Based on cut vertices, we can decompose the MCS into a block tree: provides a “high-level” view of the dependencies (edge partitioning)

- $B_1$ induced by $\{1, 2, 3, 4\}$
- $B_2$ induced by $\{4, 5\}$
- $B_3$ induced by $\{3, 6\}$
cycle breaking by creating a spanning tree of a cyclic MCS
cycle breaking by creating a spanning tree of a cyclic MCS

![Diagram](attachment:image.png)
Optimization: Create Acyclic Topologies

cycle breaking by creating a spanning tree of a cyclic MCS

\[ P = \langle P_0, P_1, P_2, P_3, P_4 \rangle \]

deployment edges \( cb(G, P) \): remove last edge from each path

ear decomposition \( P = \langle P_0, \rangle \)
cycle breaking by creating a spanning tree of a cyclic MCS

\[ P = \langle P_0, P_1, \ldots, P_4 \rangle \]

**ear decomposition**

\[ P = \langle P_0, P_1, \ldots, P_4 \rangle \]
cycle breaking by creating a spanning tree of a cyclic MCS

\[
\text{ear decomposition } P = \langle P_0, P_1, P_2, \rangle
\]
Optimization: Create Acyclic Topologies

cycle breaking by creating a spanning tree of a cyclic MCS

ear decomposition \( P = \langle P_0, P_1, P_2, P_3, \rangle \)
cycle breaking by creating a spanning tree of a cyclic MCS

ear decomposition $P = \langle P_0, P_1, P_2, P_3, P_4 \rangle$
Optimization: Create Acyclic Topologies

cycle breaking by creating a spanning tree of a cyclic MCS

\[ P = \langle P_0, P_1, P_2, P_3, P_4 \rangle \]

cycle breaker edges \( cb(G, P) \): remove last edge from each path \( P_i \) in \( G \)
transitive reduction of a digraph $G$ is the graph $G^-$ with the smallest set of edges whose transitive closure $G^+$ equals the one of $G$
Example (ctd.)

- $B_1$: acyclic $\rightarrow$ apply transitive reduction
- $B_2$: cyclic $\rightarrow$ apply ear decomposition, then apply transitive reduction (already reduced)
- $B_3$: acyclic and already reduced
In a pruned block $B'$, take all variables from

- the minimal interface in $B'$
- child cut vertices $c$
- removed edges $E$

Outcome: query plan for the MCS to restrict calls and partial belief states
Example - Query Plan

1

2

3

4

5

6

\{ \text{train}_2, \text{car}_3, \text{train}_3, \text{coke}_3, \text{car}_4, \text{train}_4 \}

\{ \text{train}_3, \text{coke}_3, \text{car}_3, \text{c_peanuts}_3, \text{car}_4, \text{train}_4 \}

\{ \text{car}_4, \text{train}_4 \}

\{ \text{sick}_6 \}

\{ \text{car}_4, \text{train}_4 \}

\{ \text{want}_\text{sooner}_5 \}
Algorithm DMCSOPT

- Operate on the (optimized) query plan
- Does not need to break cycle
- Proceed on the leaf and intermediate cases almost similar to DMCS
- ...Except: guessing for removed edges because of cycles
Motivation for Streaming Models in MCS

For large context knowledge bases, we still face scalability issues:

- potentially many models: exhaust memory at combination- or at solving-time

- synchronous evaluation (one context may block the parent)

- this is mainly due to computing all (partial) equilibria
Motivation for Streaming Models in MCS

For large context knowledge bases, we still face scalability issues:

- potentially many models: exhaust memory at combination- or at solving-time
- synchronous evaluation (one context may block the parent)
- this is mainly due to computing all (partial) equilibria

Idea: Adapt existing algorithms with streaming mode:

- request at most $k$ partial equilibria (obtain some instead of all answers)
- allow for asynchronous communication
- allow to request further partial equilibria: communication in multiple rounds
Algorithm DMCS-STREAMING

\[ a_1^1 \lor \neg a_1^1 \leftarrow t_1 \]
\[ \ldots \]
\[ a_\ell^\ell \lor \neg a_\ell^\ell \leftarrow t_1 \]
\[ \bot \leftarrow \neg t_1 \]

\{ t_1 \leftarrow (2 : a_2^e), (3 : a_3^o) \mid 1 \leq e, o \leq \ell, e \text{ even, } o \text{ odd} \}

**Trade-off:** recomputation!!!
Algorithm DMCS-STREAMING

\begin{itemize}
  \item \( a_1^1 \lor \neg a_1^1 \leftarrow t_1 \)
  \item \( \ldots \)
  \item \( a_1^\ell \lor \neg a_1^\ell \leftarrow t_1 \)
  \item \( \bot \leftarrow \neg t_1 \)
  \item \( \{ t_1 \leftarrow (2 : a_2^e), (3 : a_3^o) \mid 1 \leq e, o \leq \ell, e \text{ even}, o \text{ odd} \} \)
\end{itemize}

\[ C_1 \]

\begin{itemize}
  \item \( a_2^1 \lor \neg a_2^1 \)
  \item \( \ldots \)
  \item \( a_2^\ell \lor \neg a_2^\ell \)
  \item \( \emptyset \)
\end{itemize}

\[ C_2 \]

\[ C_3 \]

\[ a_3^1 \lor \neg a_3^1 \]
\[ \ldots \]
\[ a_3^\ell \lor \neg a_3^\ell \]
\[ \emptyset \]

\[ k = 1: \]

\[ S_{2,1} = (\epsilon, \{a_1^1\}, \epsilon) \]
Algorithm DMCS-STREAMING

$\begin{align*}
    a_1^1 \lor \neg a_1^1 & \leftarrow t_1 \\
    \ldots \\
    a_\ell^\ell \lor \neg a_\ell^\ell & \leftarrow t_1 \\
    \bot & \leftarrow \neg t_1 \\
    \{ t_1 \leftarrow (2 : a_2^e), (3 : a_3^o) \mid 1 \leq e, o \leq \ell, e \text{ even, } o \text{ odd } \}
\end{align*}$

$k = 1$: 

$S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon)$

$S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})$
Algorithm DMCS-STREAMING

\[ a_1^1 \lor \neg a_1^1 \leftarrow t_1 \]
\[ \ldots \]
\[ a_\ell^\ell \lor \neg a_\ell^\ell \leftarrow t_1 \]
\[ \bot \leftarrow \neg t_1 \]

\{ t_1 \leftarrow (2 : a^e_2), (3 : a^o_3) \mid 1 \leq e, o \leq \ell, e \text{ even, } o \text{ odd} \}

\begin{align*}
C_1 & = (a_1^1 \lor \neg a_1^1, \ldots, a_\ell^\ell \lor \neg a_\ell^\ell) \\
C_2 & = (a_2^1 \lor \neg a_2^1, \ldots, a_\ell^\ell \lor \neg a_\ell^\ell) \\
C_3 & = (a_3^1 \lor \neg a_3^1, \ldots, a_\ell^\ell \lor \neg a_\ell^\ell) \\
\emptyset & = (\emptyset, \emptyset, \emptyset) \\
k = 1: & \quad S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon) \\
& \quad S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})
\end{align*}
Algorithm DMCS-STREAMING

\[
\begin{align*}
& a_1^1 \lor \neg a_1^1 \leftarrow t_1 \\
& \ldots \\
& a_\ell^1 \lor \neg a_1^1 \leftarrow t_1 \\
& \bot \leftarrow \neg t_1 \\
& \{ t_1 \leftarrow (2 : a_e^2), (3 : a_o^3) \mid 1 \leq e, o \leq \ell, e \text{ even}, o \text{ odd} \} \\
\end{align*}
\]

\[
\begin{align*}
& a_2^1 \lor \neg a_2^1 \\
& \ldots \\
& a_\ell^2 \lor \neg a_2^\ell \\
& \emptyset
\end{align*}
\]

\[
\begin{align*}
& a_3^1 \lor \neg a_3^1 \\
& \ldots \\
& a_\ell^3 \lor \neg a_3^\ell \\
& \emptyset
\end{align*}
\]

\[
k = 1: \\
S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon) \\
S_{2,2} = (\epsilon, \{a_2^2\}, \epsilon) \\
S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})
\]
Algorithm DMCS-STREAMING

\[ a_1^1 \lor -a_1^1 \leftarrow t_1 \]
\[ \ldots \]
\[ a_1^\ell \lor -a_1^\ell \leftarrow t_1 \]
\[ \bot \leftarrow -t_1 \]

\{ \ t_1 \leftarrow (2 : a_2^e), (3 : a_3^o) \mid 1 \leq e, o \leq \ell, e \text{ even}, o \text{ odd} \ \}

C_1

C_2

C_3

\[ a_1^2 \lor -a_2^1 \]
\[ \ldots \]
\[ a_1^\ell \lor -a_2^\ell \]
\[ \emptyset \]

\[ a_3^1 \lor -a_3^1 \]
\[ \ldots \]
\[ a_3^\ell \lor -a_3^\ell \]
\[ \emptyset \]

\[ S_{1,1} = (\{a_1^1, t_1\}, \{a_2^2\}, \{a_3^1\}) \]
\[ S_{2,1} = (\epsilon, \overline{\{a_2^1\}}, \epsilon) \]
\[ S_{2,2} = (\epsilon, \{a_2^2\}, \epsilon) \]
\[ S_{3,1} = (\epsilon, \epsilon, \{a_3^1\}) \]

k = 1:
Algorithm DMCS-STREAKING

\begin{align*}
S_{1,1} &= (\{a_1^1, t_1\}, \{a_2^2\}, \{a_3^1\}) \\
S_{2,1} &= (\epsilon, \{a_2^1\}, \epsilon) \\
S_{2,2} &= (\epsilon, \{a_2^2\}, \epsilon) \\
S_{3,1} &= (\epsilon, \epsilon, \{a_3^1\})
\end{align*}

Trade-off: recomputation!!!
DMCS System Architecture

Client dmcs

N1 dmcs

N2 dmcs

N3 dmcs

N4 dmcs

Manager dmcs

Partial equilibria requests

Registration

Query

Requests

Request Dispatcher

Output Dispatcher

Context

Joiner Dispatcher

NOut NIn

NOut NIn

External requests external partial equilibria

Internal requests internal partial equilibria

Notifications

Cycle Breaker

Evaluator

Bridge Rules Evaluator

Joiner

Output Dispatcher

Requests Dispatcher

Evaluator

Joiner Dispatcher

Requests heads

Partial equilibria notifications
Experiments: Benchmark Setup

Topologies:

- Binary Tree
- Diamond
- Zig-zag
- Ring

Other quantitative parameters:

- \( n \): system size
- \( s \): local theory size
- \( b \): number of interface atoms
- \( r \): maximal number of bridge rules

Local theories’ structure:

A local theory has \( 2^m \) answer sets, where \( m \in [0, s/2] \).
Experiments: The Run

Parameter choice (based on some initial testing):

- $n$ was chosen based on the topology:
  - $T : n \in \{7, 10, 15, 31, 70, 100\}$
  - $D : n \in \{4, 7, 10, 13, 25, 31\}$
  - $Z : n \in \{4, 7, 10, 13, 25, 31, 70\}$
  - $R : n \in \{4, 7, 10, 13, 70\}$
- $s, b, r$ are fixed to either $10, 5, 5$ or $20, 10, 10$, respectively.

Way to proceed:

- Test 5 instances per parameter setting
- Run DMCS, DMCSOPT on non-streaming and streaming mode (DMCS-STREAMING)
- In streaming mode, run with different package sizes: 1, 10, 100
- Measure:
  - Total number of returned partial equilibria
  - Total running time (in secs)
  - Running time to get the first set of answers (in streaming mode)
Experiments: The Run

Parameter choice (based on some initial testing):

▶ \( n \) was chosen based on the topology:
  ▶ \( T : n \in \{7, 10, 15, 31, 70, 100\} \)
  ▶ \( D : n \in \{4, 7, 10, 13, 25, 31\} \)
  ▶ \( Z : n \in \{4, 7, 10, 13, 25, 31, 70\} \)
  ▶ \( R : n \in \{4, 7, 10, 13, 70\} \)

▶ \( s, b, r \) are fixed to either 10, 5, 5 or 20, 10, 10, respectively.

Way to proceed:

▶ test 5 instances per parameter setting
▶ run DMCS, DMCSOPT on non-streaming and streaming mode (DMCS-STREAMING)
▶ in streaming mode, run with different package sizes: 1, 10, 100
▶ measure:
  ▶ total number of returned partial equilibria
  ▶ total running time (in secs)
  ▶ running time to get the first set of answers (in streaming mode)
Experiments: Analysis

- Comparing DMCS and DMCSOPT
- Comparing streaming and non-streaming modes
- Effect of the package size
- Role of the topologies
DMCS vs. DMCSOPT (non-streaming)
DMCS vs. DMCSOPT (streaming)

(a) $T(25, 10, 5, 5)$

(b) $D(10, 10, 5, 5)$

(c) $Z(10, 10, 5, 5)$

(d) $R(4, 10, 5, 5)$

- stream $N$ partial equilibria: not a fair comparison due to projection
- first return: might have the above effect from intermediate contexts
Streaming vs. Nonstreaming (DMCS)

- Streaming wins in most of the cases
- Ring behaves irregularly!
Streaming vs. Nonstreaming (DMCSOPT)

with small systems and local theories

Streaming loses because of recomputation
Streaming vs. Nonstreaming (DMCSOPT)

With large systems and local theories:

- Streaming starts gaining back...
- ...but does not always win, again due to recomputation
Effect of the Package Size

Average time to find 1 partial equilibrium in streaming mode

- $k = 1$ looks ok, too large package size is not always a good idea
- Ring behaves irregularly
Roles of Topologies

Topological aspects that affect the performance:

(i) number of connections
(ii) structure of block trees and cut vertices
(iii) cyclicity

Observations:

\[ T >_{\text{DMCS}} (i, ii) \ DMCS \ D >_{\text{DMCS}} (i) \ DMCS \ Z >_{\text{DMCS}} (iii) \ DMCS \ R \]

\[ T >_{\text{DMCSOPT}} (i, ii) \ DMCSOPT \ Z >_{\text{DMCSOPT}} (ii) \ DMCSOPT \ D >_{\text{DMCSOPT}} (iii) \ DMCSOPT \ R \]
Summary of Contributions

Exploration of an area that had not been considered before:

design, implement, and analyze truly distributed algorithms to evaluate partial equilibria of Heterogeneous Multi-Context Systems.
Summary of Contributions

Exploration of an area that had not been considered before:

design, implement, and analyze truly distributed algorithms to evaluate partial equilibria of Heterogeneous Multi-Context Systems.

- Algorithms DMCS, DMCSOPT, DMCS-STREAMING,
- The DMCS System,
- Experimental Evaluation.
Summary of Contributions

Exploration of an area that had not been considered before:

*design, implement, and analyze truly distributed algorithms to evaluate partial equilibria of Heterogeneous Multi-Context Systems.*

- Algorithms DMCS, DMCSOPT, DMCS-STREAMING,
- The DMCS System,
- Experimental Evaluation.

Thus establish another step to bring MCSs to real life applications!
Future Work

- Implementation issues for DMCS
- Grounding-on-the-fly for non-ground ASP-based MCS
- Conflict learning in DMCS
- Query answering in MCS
- Distributed Heterogeneous Stream Reasoning
Future Work

- Implementation issues for DMCS
- Grounding-on-the-fly for non-ground ASP-based MCS
- Conflict learning in DMCS
- Query answering in MCS
- Distributed Heterogeneous Stream Reasoning

Thank you very much for your attention!


References IV


