Distributed Nonmonotonic Multi-Context Systems: Algorithms and Efficient Evaluation

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?

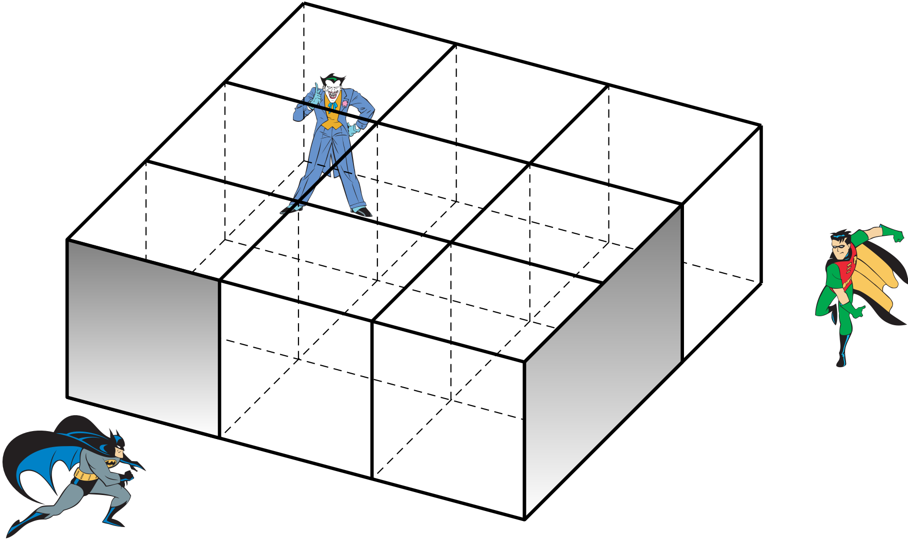
  $C_i = (L_i, kb_i, br_i)$

  - a logic L_i
  - the context's knowledge base kb_i
  - a set br_i of bridge rules
- What is a logic?

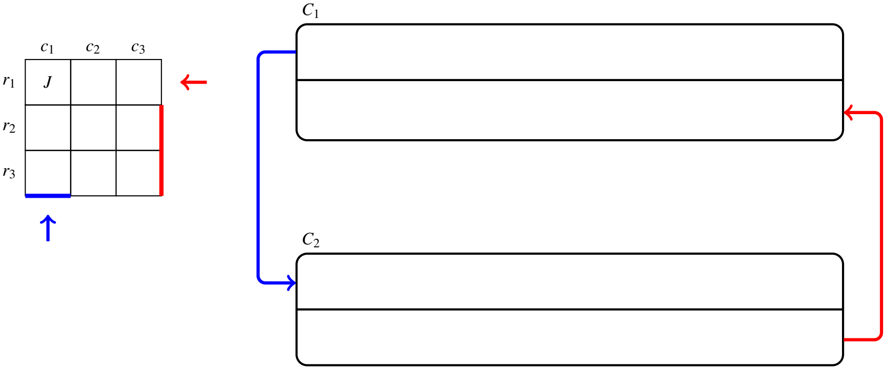
  $L = (\mathbf{KB}_L, \mathbf{BS}_L, \mathbf{ACC}_L)$

  - 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?

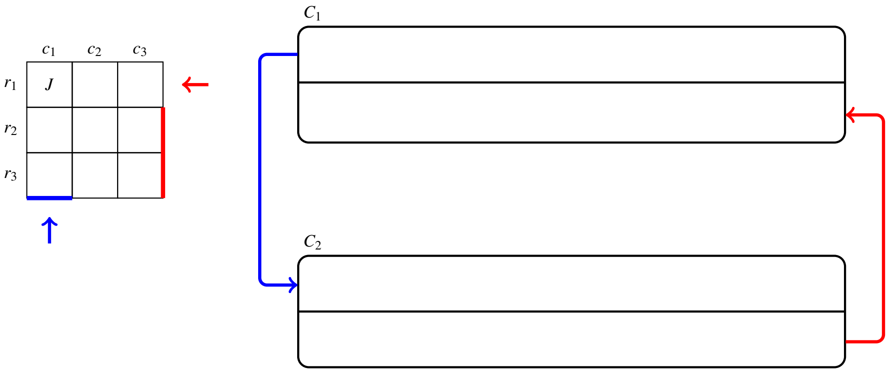
MCS Example



MCS Example - Encoding

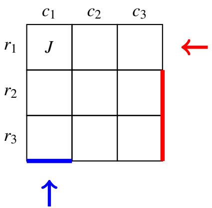


MCS Example - Encoding



where: $$R = \left\{ \begin{array}{rcl} joker\_in & \leftarrow & at\_row(X). \\ joker\_in & \leftarrow & at\_col(X). \\ at\_row(X) & \leftarrow & joker\_in, row(X), not\ \neg at\_row(X). \\ \neg at\_row(X) & \leftarrow & joker\_in, row(X), at\_row(Y), X \neq Y. \\ at\_col(X) & \leftarrow & joker\_in, col(X), not\ \neg at\_col(X). \\ \neg at\_col(X) & \leftarrow & joker\_in, col(X), at\_col(Y), X \neq Y. \\ & & row(1).\ row(2).\ row(3). \\ & & col(1).\ col(2).\ col(3). \end{array} \right\}$$

MCS Example - Encoding



C_1

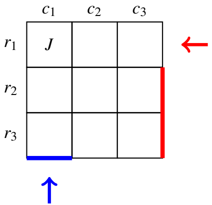
$$\left\{\begin{array}{rcl} at\_col(X) & \leftarrow & see\_col(X). \\ \neg at\_col(X) & \leftarrow & \neg see\_col(X). \end{array}\right\} \cup R \cup \{\neg see\_col(2).\ \neg see\_col(3).\}$$

C_2

$$\left\{\begin{array}{rcl} at\_row(X) & \leftarrow & see\_row(X). \\ \neg at\_row(X) & \leftarrow & \neg see\_row(X). \end{array}\right\} \cup R \cup \{see\_row(1).\}$$

where: $$R = \left\{\begin{array}{rcl} joker\_in & \leftarrow & at\_row(X). \\ joker\_in & \leftarrow & at\_col(X). \\ at\_row(X) & \leftarrow & joker\_in, row(X), not\ \neg at\_row(X). \\ \neg at\_row(X) & \leftarrow & joker\_in, row(X), at\_row(Y), X \neq Y. \\ at\_col(X) & \leftarrow & joker\_in, col(X), not\ \neg at\_col(X). \\ \neg at\_col(X) & \leftarrow & joker\_in, col(X), at\_col(Y), X \neq Y. \\ & & row(1).\ row(2).\ row(3). \\ & & col(1).\ col(2).\ col(3). \end{array}\right\}$$

MCS Example - Encoding



C_1

$$\left\{\begin{array}{rcl} at\_col(X) & \leftarrow & see\_col(X). \\ \neg at\_col(X) & \leftarrow & \neg see\_col(X). \end{array}\right\} \cup R \cup \{\neg see\_col(2).\ \neg see\_col(3).\}$$

$$\begin{array}{rcl} at\_row(X) & \leftarrow & (2 : at\_row(X)). \\ \neg at\_row(X) \vee covered\_row(X) & \leftarrow & not\ (2 : see\_row(X)), (1 : row(X)). \end{array}$$

C_2

$$\begin{array}{rcl} at\_col(X) & \leftarrow & (1 : at\_col(X)). \\ \neg at\_col(X) \vee covered\_col(X) & \leftarrow & not\ (1 : see\_col(X)), (2 : col(X)). \end{array}$$

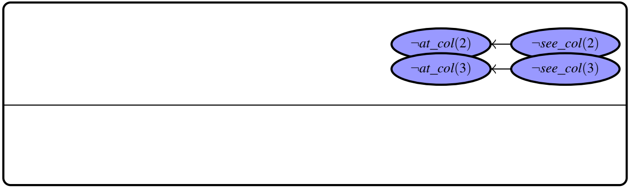
$$\left\{\begin{array}{rcl} at\_row(X) & \leftarrow & see\_row(X). \\ \neg at\_row(X) & \leftarrow & \neg see\_row(X). \end{array}\right\} \cup R \cup \{see\_row(1).\}$$

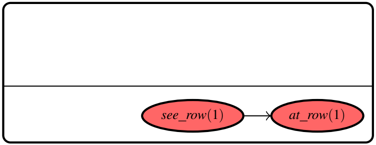
where: $$R = \left\{\begin{array}{rcl} joker\_in & \leftarrow & at\_row(X). \\ joker\_in & \leftarrow & at\_col(X). \\ at\_row(X) & \leftarrow & joker\_in, row(X), not\ \neg at\_row(X). \\ \neg at\_row(X) & \leftarrow & joker\_in, row(X), at\_row(Y), X \neq Y. \\ at\_col(X) & \leftarrow & joker\_in, col(X), not\ \neg at\_col(X). \\ \neg at\_col(X) & \leftarrow & joker\_in, col(X), at\_col(Y), X \neq Y. \\ & & row(1).\ row(2).\ row(3). \\ & & col(1).\ col(2).\ col(3). \end{array}\right\}$$

Reasoning in MCSs: Equilibria

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

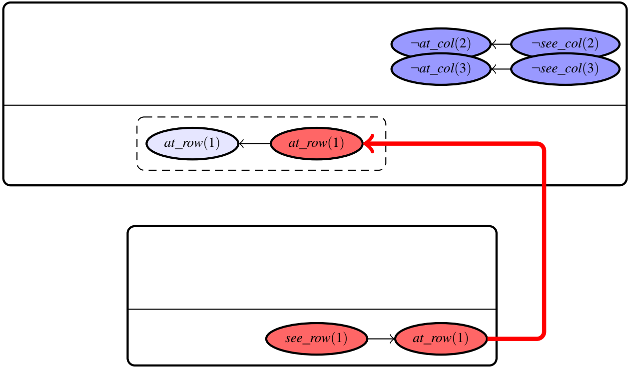




Reasoning in MCSs: Equilibria

- Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in \mathbf{BS}_{L_i}$
  ... 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.

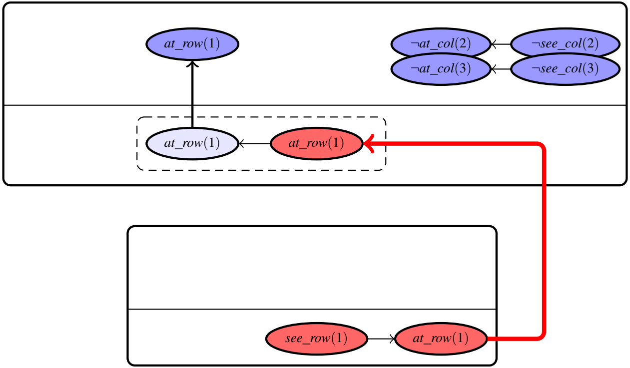
  Equilibrium condition: $S_i \in \mathbf{ACC}(kb_i \cup H_i)$ for all C_i



Reasoning in MCSs: Equilibria

- Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in \mathbf{BS}_{L_i}$
  ... 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.

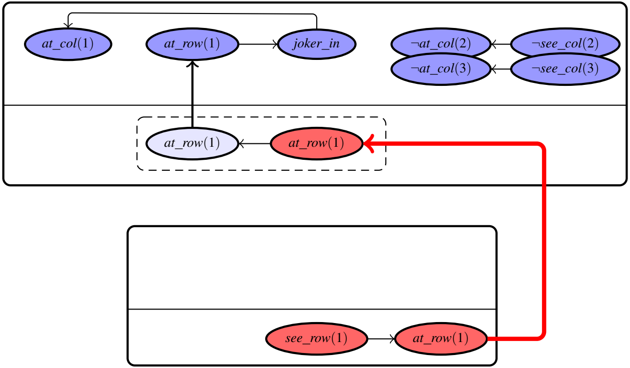
  Equilibrium condition: $S_i \in \mathbf{ACC}(kb_i \cup H_i)$ for all C_i



Reasoning in MCSs: Equilibria

- Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in \mathbf{BS}_{L_i}$
  ... 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.

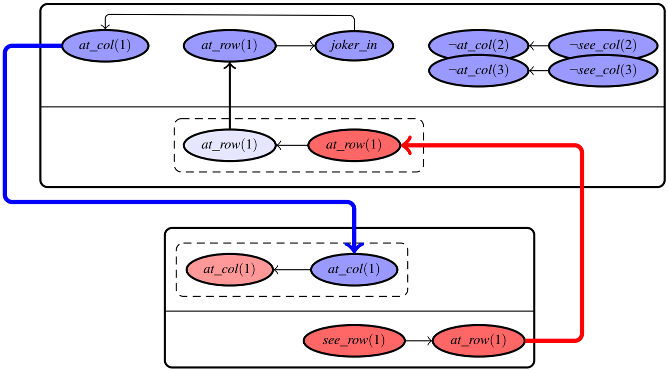
  Equilibrium condition: $S_i \in \mathbf{ACC}(kb_i \cup H_i)$ for all C_i



Reasoning in MCSs: Equilibria

- Equilibrium semantics: a belief state $S = (S_1, \ldots, S_n)$ with $S_i \in \mathbf{BS}_{L_i}$
  ... 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.

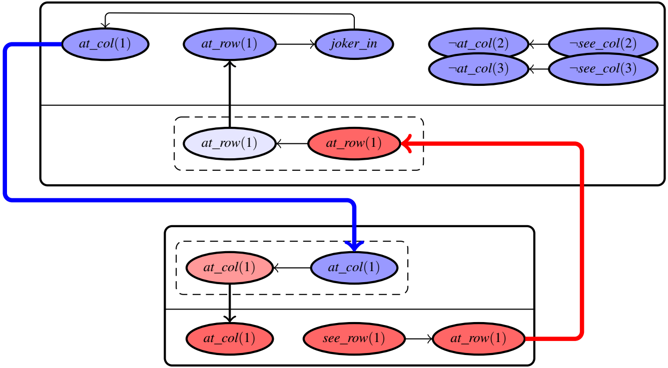
  Equilibrium condition: $S_i \in \mathbf{ACC}(kb_i \cup H_i)$ for all C_i



Reasoning in MCSs: Equilibria

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



Why are MCSs interesting?

Distributedness / Heterogeneity / Nonmonotonicity

Why are MCSs interesting?

Distributedness / Heterogeneity / Nonmonotonicity

$\Rightarrow$ Power to model real life applications:

- collaboration between business partners,
- medical applications,
- reasoning on the web,
- ...

Why are MCSs interesting?

Distributedness / Heterogeneity / Nonmonotonicity

$\Rightarrow$ 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

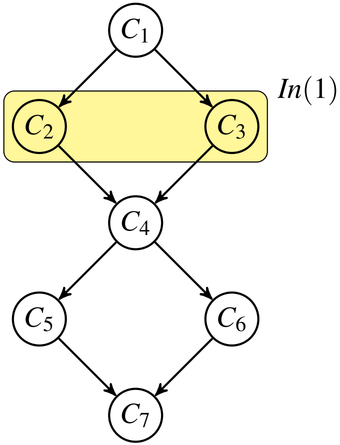
Support notions:

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

Import Neighborhood and Closure

Import neighborhood of C_k

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

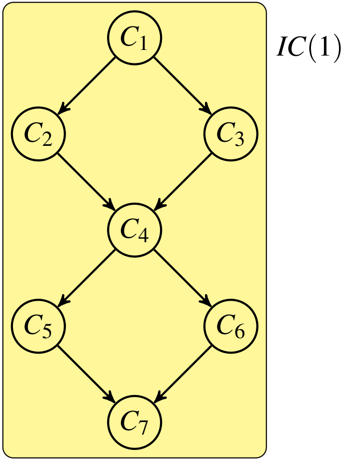


Import Neighborhood and Closure

Import neighborhood of C_k

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

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



Partial Belief States and Equilibria

Let $M = (C_1, \ldots, C_n)$ be an MCS, and let $\epsilon \notin \bigcup_{i=1}^{n} \mathbf{BS}_i$

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 $S = (S_1, \ldots, S_n)$, where $S_i \in \mathbf{BS}_i \cup \{\epsilon\}$, for $1 \leq i \leq n$

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 $S = (S_1, \ldots, S_n)$, where $S_i \in \mathbf{BS}_i \cup \{\epsilon\}$, 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 $i \in IC(k)$ then $S_i \in \mathbf{ACC}_i(kb_i \cup \{head(r) \mid r \in app(br_i, S)\})$
- 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.

$S =$	S_1	$\cdots$	ϵ	$\cdots$	ϵ	$\cdots$	S_j	$\cdots$	S_n
$T =$	ϵ	$\cdots$	ϵ	$\cdots$	T_i	$\cdots$	T_j	$\cdots$	ϵ
$S \bowtie T =$	S_1	$\cdots$	ϵ	$\cdots$	T_i	$\cdots$	$S_j = T_j$	$\cdots$	S_n

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

$$\mathcal{S} \bowtie \mathcal{T} = \{S \bowtie T \mid S \in \mathcal{S}, T \in \mathcal{T}\}$$

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 lsolve(S) for each context C_k is available which computes $\mathbf{ACC}_k(kb_k \cup app_k(S))$

Algorithm DMCS

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

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

Input parameters for DMCS:

- V: set of "interesting" variables (to project the partial equilibria)
- $hist$: 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 lsolve(S) for each context C_k is available which computes $\mathbf{ACC}_k(kb_k \cup app_k(S))$

Input parameters for DMCS:

- V: set of “interesting” variables (to project the partial equilibria)
- $hist$: visited path

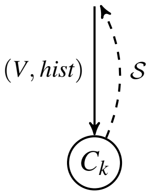
Strategy: DFS-traversal of M starting with C_k, visiting all C_i for $i \in IC(k)$

Distributedness: instances of DMCS

- running at each context node,
- communicating with each other for exchanging sets of belief states

Acyclic case

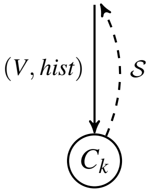
Leaf context C_k $(br_k = \emptyset)$



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

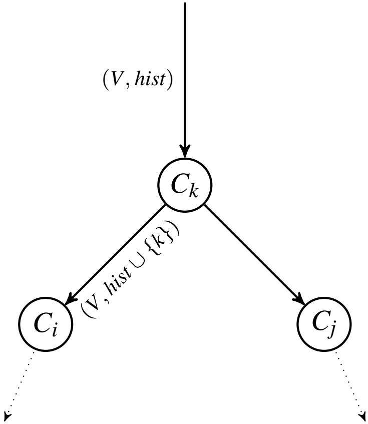
Acyclic case

Leaf context C_k ($br_k = \emptyset$)



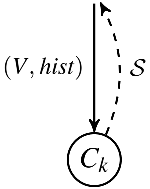
$\mathsf{lsolve}((\epsilon, \ldots, \epsilon)) = \mathcal{S}$

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



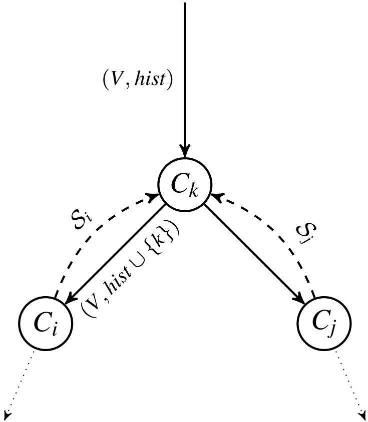
Acyclic case

Leaf context C_k ($br_k = \emptyset$)



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

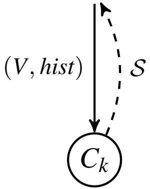
Intermediate context C_k ($(i:p), (j:q)$ appear in br_k)



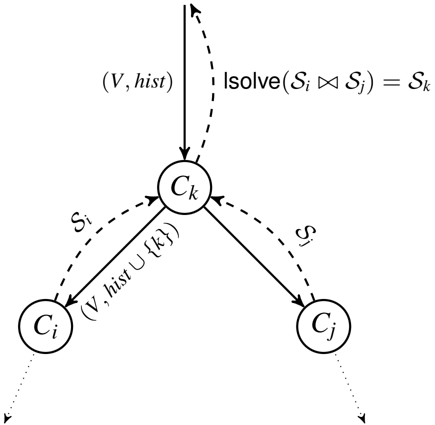
Acyclic case

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

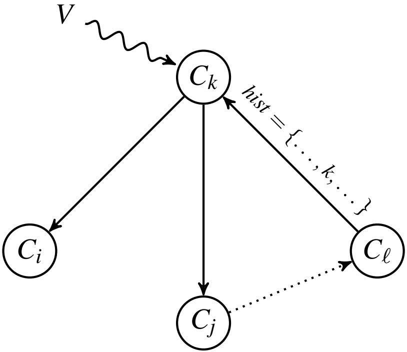
Leaf context C_k $(br_k = \emptyset)$



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

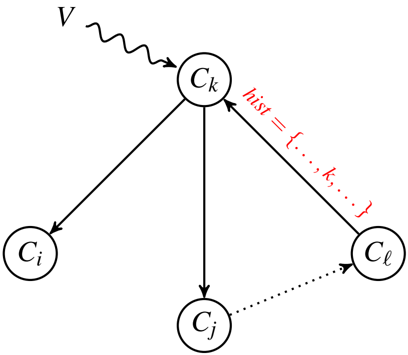


Cycle Breaking



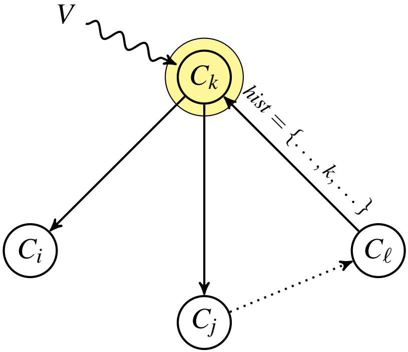
Cycle Breaking

C_k detects a cycle in *hist*



Cycle Breaking

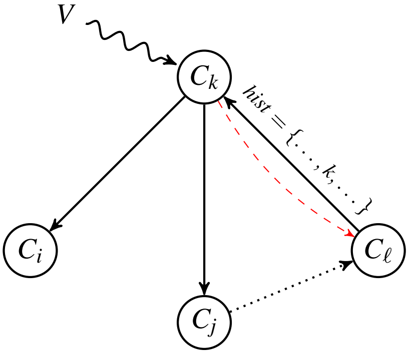
C_k detects a cycle in $hist$



- guesses local belief sets

Cycle Breaking

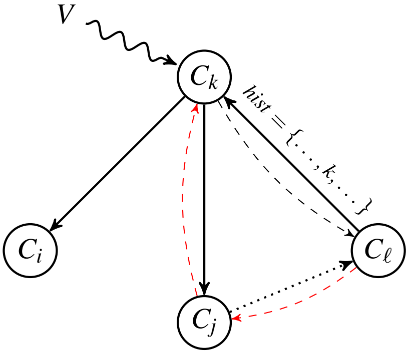
C_k detects a cycle in $hist$



- guesses local belief sets
- returns them to invoking context

Cycle Breaking

C_k detects a cycle in $hist$

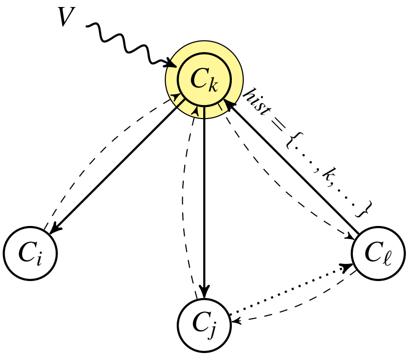


- guesses local belief sets
- returns them to invoking context

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

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?

Example

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

Example

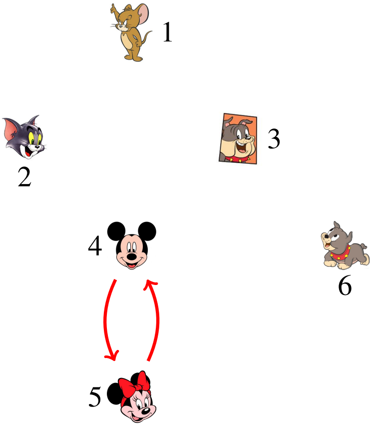
- Problem: How to go home?
- Possible solutions:
  - Car: slower than train
  - Train: should bring some food
- Spike and Mickey have additional information from Tyke and Minnie

Example (ctd.)

- Minnie wants Mickey to come back as soon as possible.

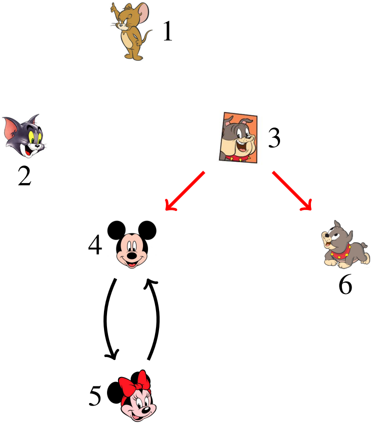
$$kb_4 = \{car_4 \lor train_4 \leftarrow \}$$
$$br_4 = \{train_4 \leftarrow (5 : want\_sooner_5)\}$$

$$kb_5 = \{want\_sooner_5 \leftarrow soon_5\}$$
$$br_5 = \{soon_5 \leftarrow (4 : train_4)\}$$



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.



$$kb_3 = \left\{ \begin{array}{r} car_3 \vee train_3 \leftarrow \\ train_3 \leftarrow urgent_3 \\ sandwiches_3 \vee chocolate\_peanuts_3 \leftarrow train_3 \\ coke_3 \vee juice_3 \leftarrow train_3 \end{array} \right\}$$

$$br_3 = \left\{ \begin{array}{r} urgent_3 \leftarrow (6 : sick_6) \\ train_3 \leftarrow (4 : train_4) \end{array} \right\};$$

$$kb_6 = \{ sick_6 \vee fit_6 \leftarrow \}$$

$$br_6 = \emptyset.$$

Example (ctd.)

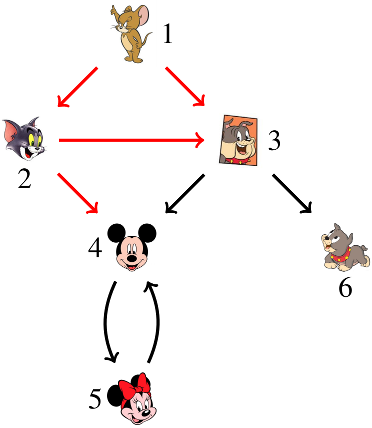
- Jerry is the leader of the group.
- Jerry is allergic to peanuts.
- Tom wants to get home somehow and doesn't want coke.

$$kb_1 = \left\{ \begin{array}{c} car_1 \leftarrow not\ train_1 \\ \bot \leftarrow peanuts_1 \end{array} \right\}$$

$$br_1 = \left\{ \begin{array}{c} train_1 \leftarrow (2 : train_2), (3 : train_3) \\ peanuts_1 \leftarrow (3 : chocolate\_peanuts_3) \end{array} \right\}$$

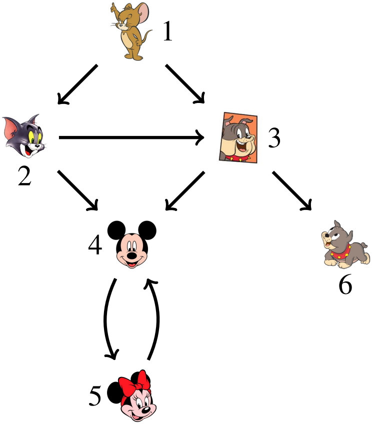
$kb_2 = \{\bot \leftarrow not\ car_2, not\ train_2\}$ and

$$br_2 = \left\{ \begin{array}{c} car_2 \leftarrow (3 : car_3), (4 : car_4) \\ train_2 \leftarrow (3 : train_3), (4 : train_4), \\ not\ (3 : coke_3) \end{array} \right\}$$



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.



$$kb_1 = \left\{ \begin{array}{c} car_1 \leftarrow not\ train_1 \\ \bot \leftarrow peanuts_1 \end{array} \right\}$$

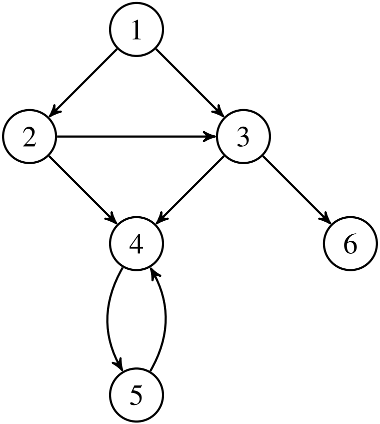
$$br_1 = \left\{ \begin{array}{c} train_1 \leftarrow (2 : train_2), (3 : train_3) \\ peanuts_1 \leftarrow (3 : chocolate\_peanuts_3) \end{array} \right\}$$

$kb_2 = \{\bot \leftarrow not\ car_2, not\ train_2\}$ and

$$br_2 = \left\{ \begin{array}{c} car_2 \leftarrow (3 : car_3), (4 : car_4) \\ train_2 \leftarrow (3 : train_3), (4 : train_4), \\ not\ (3 : coke_3) \end{array} \right\}$$

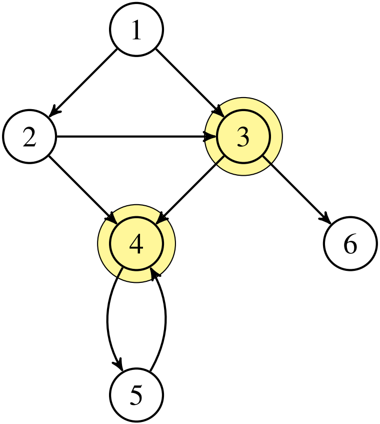
One equilibrium is $S = (\{train_1\}, \{train_2\}, \{train_3, urgent_3, juice_3, sandwiches_3\}, \{train_4\}, \{soon_5, want\_sooner_5\}, \{sick_6\})$

Example (ctd.)



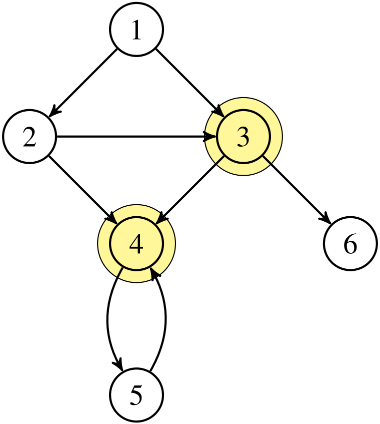
- Jerry decides after gathering information.

Example (ctd.)



- Jerry decides after gathering information.
- Mickey and Spike do not want to bother the others.

MCS Decomposition: Cut vertex



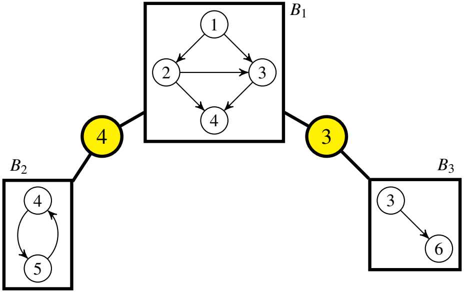
A vertex c of a weakly connected graph G is a *cut vertex*, if $G\backslash c$ is disconnected

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)



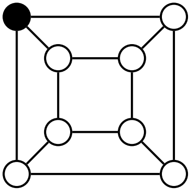
- B_1 induced by $\{1, 2, 3, 4\}$
- B_2 induced by $\{4, 5\}$
- B_3 induced by $\{3, 6\}$

Optimization: Create Acyclic Topologies

cycle breaking by creating a spanning tree of a cyclic MCS

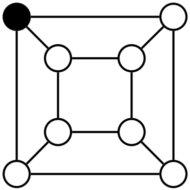
Optimization: Create Acyclic Topologies

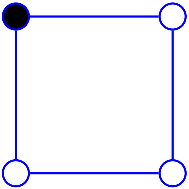
cycle breaking by creating a spanning tree of a cyclic MCS



Optimization: Create Acyclic Topologies

cycle breaking by creating a spanning tree of a cyclic MCS

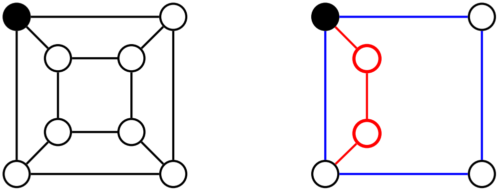




ear decomposition $P = \langle\, P_0, \qquad\qquad\qquad \rangle$

Optimization: Create Acyclic Topologies

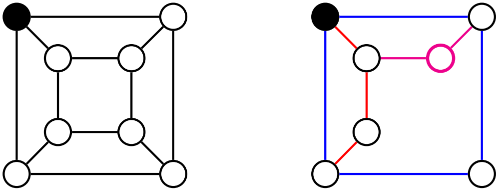
cycle breaking by creating a spanning tree of a cyclic MCS



ear decomposition $P = \langle\, P_0, P_1, \qquad\qquad \rangle$

Optimization: Create Acyclic Topologies

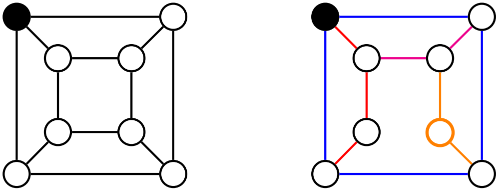
cycle breaking by creating a spanning tree of a cyclic MCS



ear decomposition $P = \langle\, P_0, P_1, P_2, \qquad\qquad \rangle$

Optimization: Create Acyclic Topologies

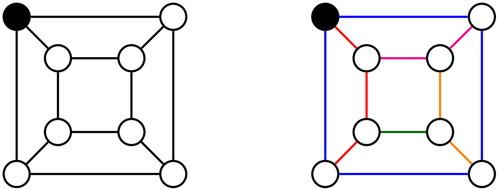
cycle breaking by creating a spanning tree of a cyclic MCS



ear decomposition $P = \langle\, P_0, P_1, P_2, P_3, \quad \rangle$

Optimization: Create Acyclic Topologies

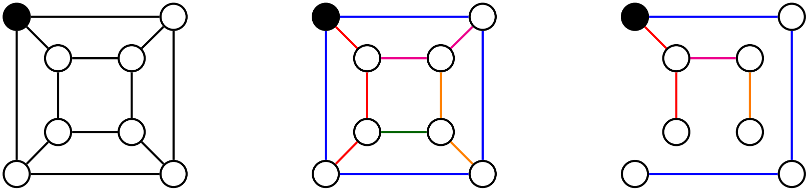
cycle breaking by creating a spanning tree of a cyclic MCS



ear decomposition $P = \langle \, P_0, P_1, P_2, P_3, P_4 \, \rangle$

Optimization: Create Acyclic Topologies

cycle breaking by creating a spanning tree of a cyclic MCS

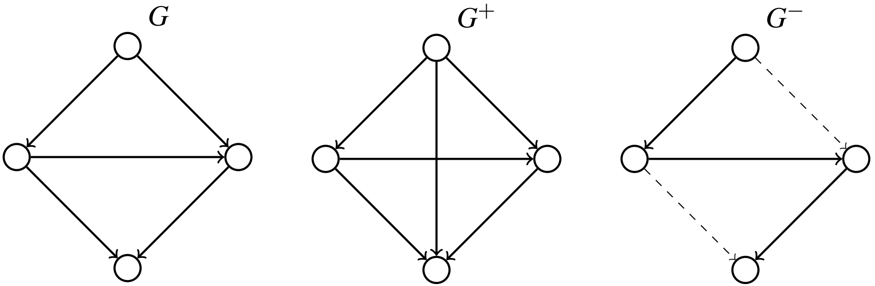


ear decomposition $P = \langle\, P_0, P_1, P_2, P_3, P_4 \,\rangle$

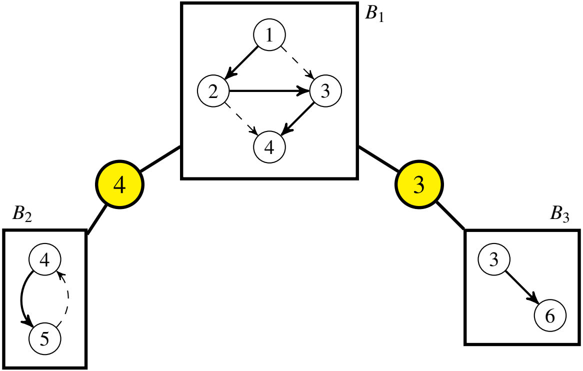
cycle breaker edges $cb(G, P)$: remove last edge from each path P_i in G

Optimization: Avoid Unnecessary Calls

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

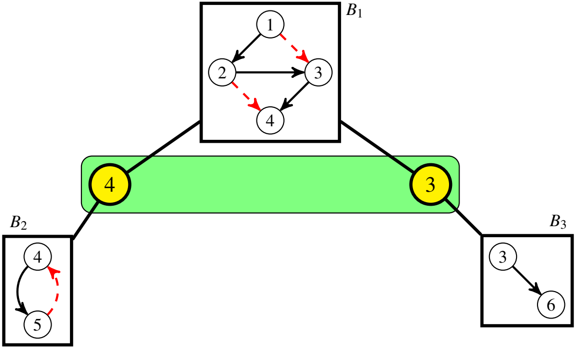


Example (ctd.)



- 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

Optimization: Minimal Interface

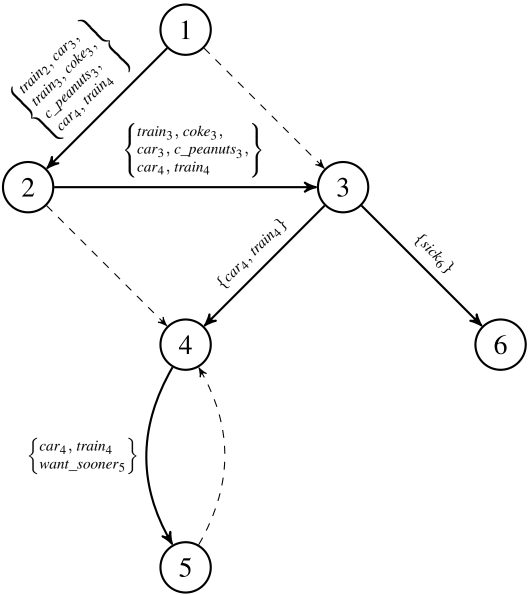


In a pruned block B', take all variables from

- 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



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

C_1:

$$\begin{array}{rcl} a_1^1 \vee \neg a_1^1 & \leftarrow & t_1 \\ & \dots & \\ a_1^\ell \vee \neg a_1^\ell & \leftarrow & t_1 \\ \bot & \leftarrow & \neg t_1 \end{array}$$

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

C_2:

$$\begin{array}{c} a_2^1 \vee \neg a_2^1 \\ \dots \\ a_2^\ell \vee \neg a_2^\ell \end{array}$$

$$\emptyset$$

C_3:

$$\begin{array}{c} a_3^1 \vee \neg a_3^1 \\ \dots \\ a_3^\ell \vee \neg a_3^\ell \end{array}$$

$$\emptyset$$

$k = 1$:

Algorithm DMCS-STREAMING

C_1

$$\begin{array}{rcl} a_1^1 \vee \neg a_1^1 & \leftarrow & t_1 \\ & \dots & \\ a_1^\ell \vee \neg a_1^\ell & \leftarrow & t_1 \\ \bot & \leftarrow & \neg t_1 \end{array}$$

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

C_2

$$\begin{array}{c} a_2^1 \vee \neg a_2^1 \\ \dots \\ a_2^\ell \vee \neg a_2^\ell \end{array}$$

$\emptyset$

C_3

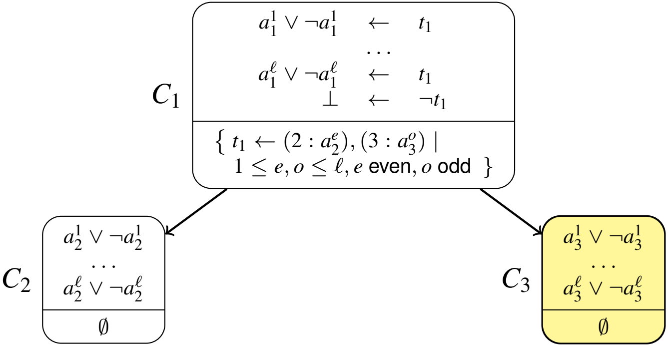
$$\begin{array}{c} a_3^1 \vee \neg a_3^1 \\ \dots \\ a_3^\ell \vee \neg a_3^\ell \end{array}$$

$\emptyset$

$k = 1$:

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

Algorithm DMCS-STREAMING



$k = 1$:

$S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon)$
$S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})$

Algorithm DMCS-STREAMING

C_1:

$$\begin{array}{rcl} a_1^1 \vee \neg a_1^1 & \leftarrow & t_1 \\ & \dots & \\ a_1^\ell \vee \neg a_1^\ell & \leftarrow & t_1 \\ \bot & \leftarrow & \neg t_1 \end{array}$$

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

C_2:

$$\begin{array}{c} a_2^1 \vee \neg a_2^1 \\ \dots \\ a_2^\ell \vee \neg a_2^\ell \end{array}$$

$$\emptyset$$

C_3:

$$\begin{array}{c} a_3^1 \vee \neg a_3^1 \\ \dots \\ a_3^\ell \vee \neg a_3^\ell \end{array}$$

$$\emptyset$$

$k = 1$:

$\cancel{S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon)}$
$\cancel{S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})}$

Algorithm DMCS-STREAMING

C_1:

$$
\begin{array}{rcl}
a_1^1 \lor \neg a_1^1 & \leftarrow & t_1 \\
 & \ldots & \\
a_1^\ell \lor \neg a_1^\ell & \leftarrow & t_1 \\
\bot & \leftarrow & \neg t_1
\end{array}
$$

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

C_2:

$$
\begin{array}{c}
a_2^1 \lor \neg a_2^1 \\
\ldots \\
a_2^\ell \lor \neg a_2^\ell
\end{array}
$$

$\emptyset$

C_3:

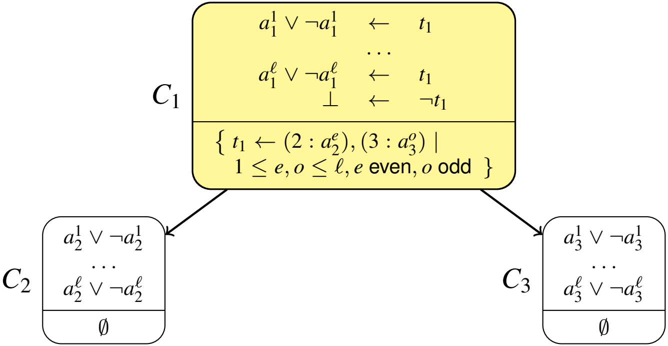
$$
\begin{array}{c}
a_3^1 \lor \neg a_3^1 \\
\ldots \\
a_3^\ell \lor \neg a_3^\ell
\end{array}
$$

$\emptyset$

$k = 1$:

$\cancel{S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon)}$ $S_{2,2} = (\epsilon, \{a_2^2\}, \epsilon)$
$S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})$

Algorithm DMCS-STREAMING



$k = 1$:

$S_{1,1} = (\{a_1^1, t_1\}, \{a_2^2\}, \{a_3^1\})$
$\cancel{S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon)}$ $S_{2,2} = (\epsilon, \{a_2^2\}, \epsilon)$
$S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})$

Algorithm DMCS-STREAMING

C_1:

$$\begin{array}{rcl} a_1^1 \vee \neg a_1^1 & \leftarrow & t_1 \\ & \dots & \\ a_1^\ell \vee \neg a_1^\ell & \leftarrow & t_1 \\ \bot & \leftarrow & \neg t_1 \end{array}$$

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

C_2:

$$\begin{array}{c} a_2^1 \vee \neg a_2^1 \\ \dots \\ a_2^\ell \vee \neg a_2^\ell \end{array}$$

$\emptyset$

C_3:

$$\begin{array}{c} a_3^1 \vee \neg a_3^1 \\ \dots \\ a_3^\ell \vee \neg a_3^\ell \end{array}$$

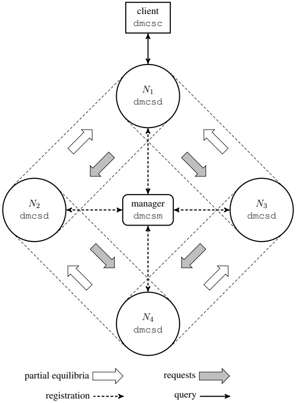
$\emptyset$

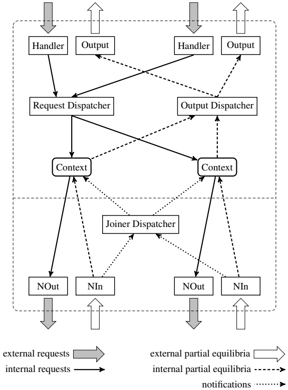
$k = 1$:

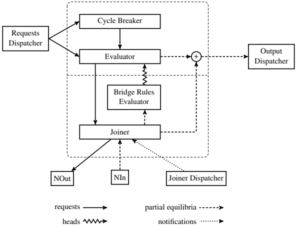
$S_{1,1} = (\{a_1^1, t_1\}, \{a_2^2\}, \{a_3^1\})$
$\cancel{S_{2,1} = (\epsilon, \{a_2^1\}, \epsilon)}$ $S_{2,2} = (\epsilon, \{a_2^2\}, \epsilon)$
$S_{3,1} = (\epsilon, \epsilon, \{a_3^1\})$

Trade-off: recomputation!!!

DMCS System Architecture

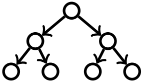
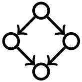
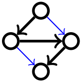
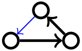






Experiments: Benchmark Setup

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.

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.

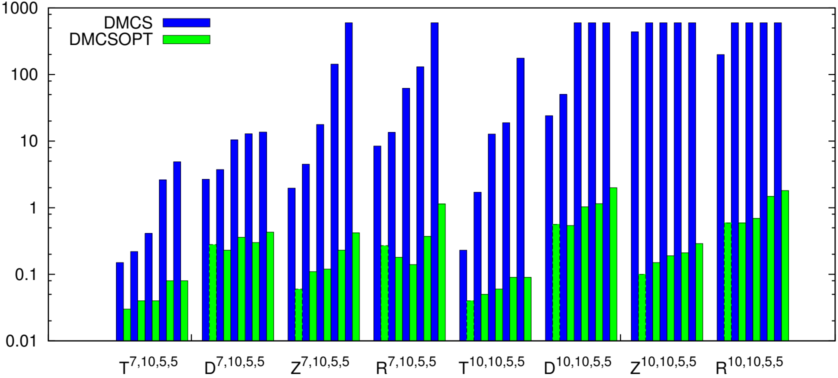
Way to proceed:

- 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)

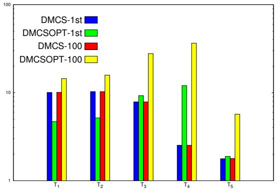
Experiments: Analysis

- Comparing DMCS and DMCSOPT
- Comparing streaming and non-streaming modes
- Effect of the package size
- Role of the topologies

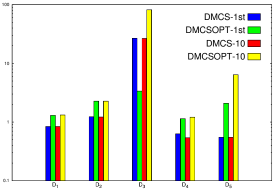
DMCS vs. DMCSOPT (non-streaming)



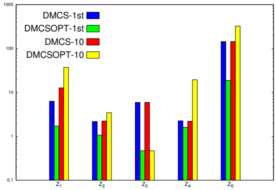
DMCS vs. DMCSOPT (streaming)



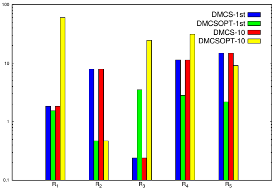
(a) $T(25, 10, 5, 5)$



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



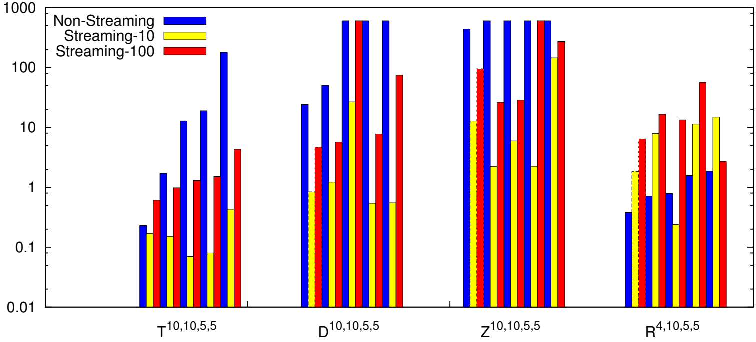
(c) $Z(10, 10, 5, 5)$



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

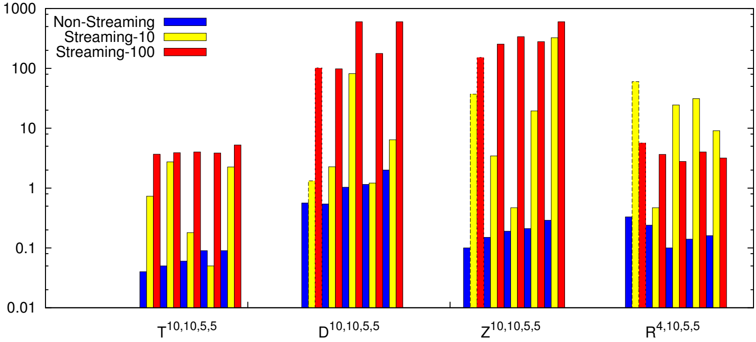
- stream N partial equilibria: not a fair comparison due to projection
- first return: might have the above effect from intermediate contexts

Streaming vs. Nonstreaming (DMCS)



- Streaming wins in most of the cases
- Ring behaves irregularly!

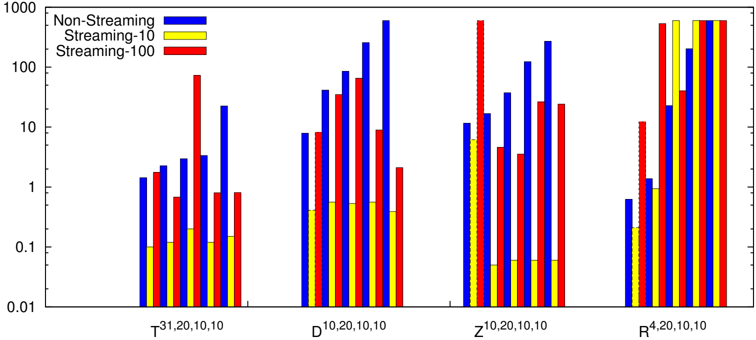
Streaming vs. Nonstreaming (DMCSOPT)



with small systems and local theories

- Streaming loses because of recomputation

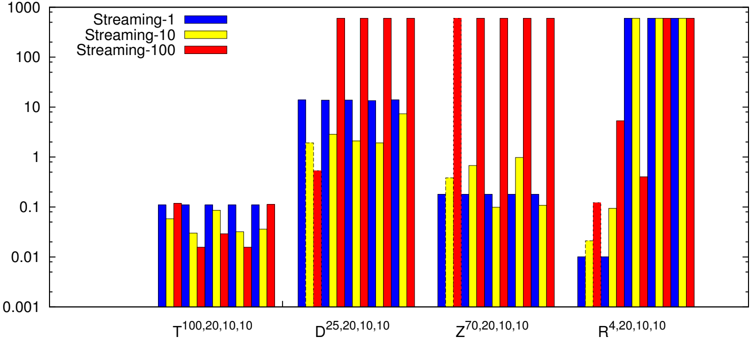
Streaming vs. Nonstreaming (DMCSOPT)



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:

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

$$T >^{(i,ii)}_{\text{DMCSOPT}} Z >^{(ii)}_{\text{DMCSOPT}} D >^{(iii)}_{\text{DMCSOPT}} R$$

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 I

- Philippe Adjiman, Philippe Chatalic, François Goasdoué, Marie-Christine Rousset, and Laurent Simon. Distributed reasoning in a peer-to-peer setting: Application to the semantic web. *J. Artif. Intell. Res.*, 25:269–314, 2006.
- 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, *Logics in Artificial Intelligence - 12th European Conference, JELIA 2010, Helsinki, Finland, September 13-15, 2010. Proceedings*, volume 6341 of *Lecture Notes in Computer Science*, pages 24–37. Springer, September 2010.

References II

- 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, *Logics in Artificial Intelligence - 12th European Conference, JELIA 2010, Helsinki, Finland, September 13-15, 2010. Proceedings*, volume 6341 of *Lecture Notes in Computer Science*, pages 352–355. Springer, September 2010.
- Antonis Bikakis, Grigoris Antoniou, and Panayiotis Hassapis. Strategies for contextual reasoning with conflicts in ambient intelligence. *Knowledge and Information Systems*, April 2010.
- Gerhard Brewka and Thomas Eiter. Equilibria in Heterogeneous Nonmonotonic Multi-Context Systems. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 385–390. AAAI Press, 2007.

References III

- Minh Dao-Tran, Thomas Eiter, Michael Fink, and Thomas Krennwallner. Distributed Nonmonotonic Multi-Context Systems. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. AAAI Press, 2010.
- Katsutoshi Hirayama and Makoto Yokoo. The distributed breakout algorithms. *Artif. Intell.*, 161(1–2):89–115, 2005.
- 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, *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*, pages 58–62. IOS Press, August 2004.

References IV

- Luciano Serafini and Andrei Tamilin. Drago: Distributed reasoning architecture for the semantic web. In Asunción Gómez-Pérez and Jérôme Euzenat, editors, *The Semantic Web: Research and Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings*, Lecture Notes in Computer Science, pages 361–376. Springer, 2005.
- Luciano Serafini, Alexander Borgida, and Andrei Tamilin. Aspects of distributed and modular ontology reasoning. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 570–575. AAAI Press, 2005.
- Makoto Yokoo and Katsutoshi Hirayama. Algorithms for distributed constraint satisfaction: A review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207, 2000.