Constraint Satisfaction: Modeling and Reformulation with Application to Geospatial Reasoning

Berthe Y. Choueiry

Constraint Systems Laboratory
Department of Computer Science & Engineering
University of Nebraska-Lincoln

Joint work with Ken Bayer, Martin Michalowski and Craig A. Knoblock

Supported by NSF CAREER Award #0133568 and
AFOSR grants FA9550-04-1-0105 and FA9550-07-1-0416

Outline

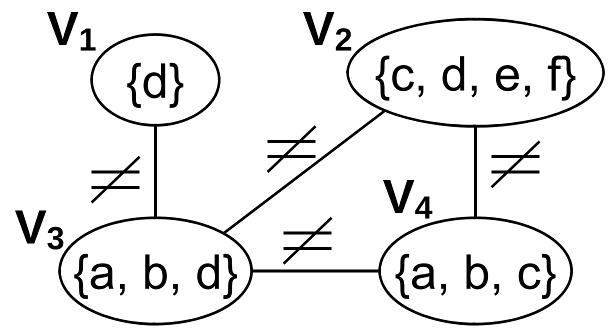
I. Background
   - Constraint Satisfaction Problem (CSP): definition, propagation algorithms, search
   - Reformulation

II. Building Identification Problem [Michalowski & Knoblock, 05]
   - Constraint model
   - Custom solver

III. Reformulation techniques
   - Query reformulation, domain reformulation, constraint relaxation, symmetry detection
   - Application to CSP, BID & evaluation on real-world BID data

- Conclusions & future work

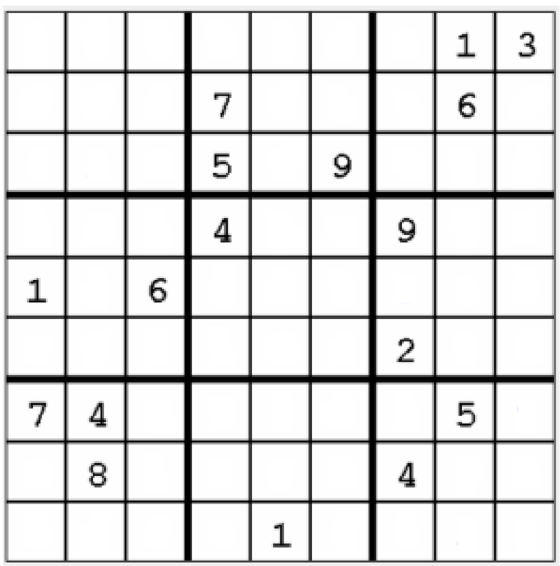
Constraint Satisfaction Problem (CSP)

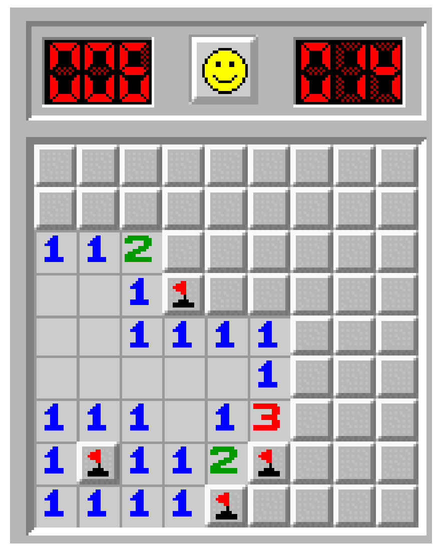


- **Given** $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$
  - $\mathcal{V}$: set of variables
  - $\mathcal{D}$: set of their domains
  - $\mathcal{C}$: set of constraints (relations) restricting the acceptable combination of values for variables
  - Solution is a consistent assignment of values to variables
- **Query**: find 1 solution, all solutions, etc.
- Deciding satisfiability is **NP**-complete in general

Examples

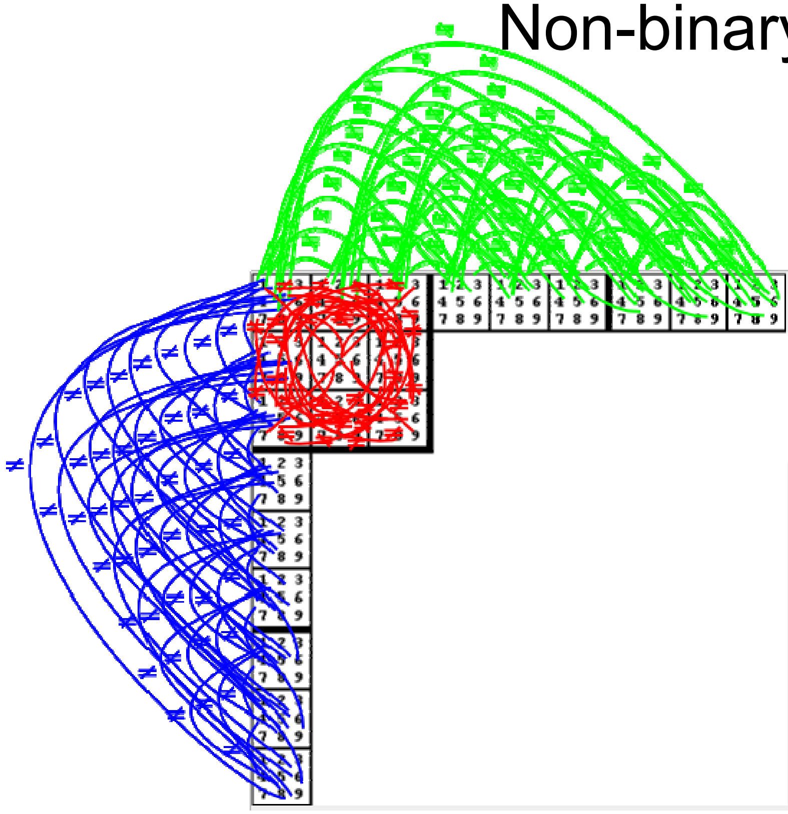
- Industrial applications: scheduling, resource allocation, product configuration, etc.
- AI: Logic inference, temporal reasoning, NLP, etc.
- Puzzles: Sudoku & Minesweeper

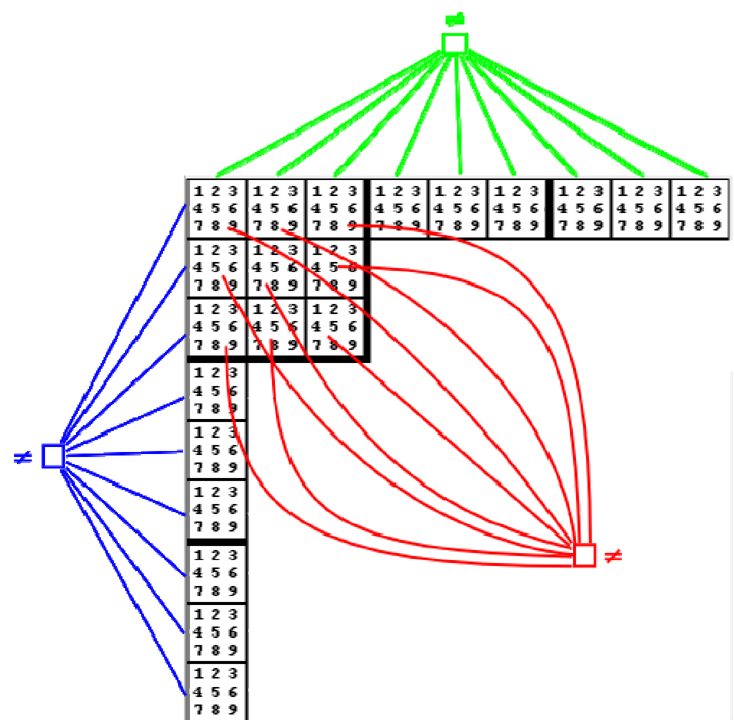




Sudoku as a CSP

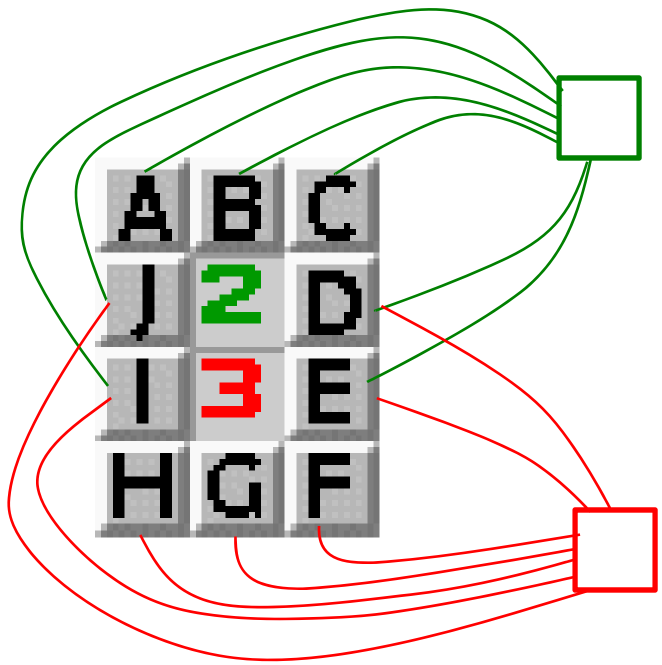
- Each cell is a variable with the domain {1,2,…,9}
- Two models: Binary, 810 AllDiff binary constraints
  Non-binary, 27 AllDiff constraints of arity 9





Minesweeper as a CSP

- Variables are the cells
- Domains are {0,1} (i.e., safe or mined)
- One constraint for each cell with a number (arity 1...8)



Exactly two mines:

0000011
0000101
0000110, etc.

Exactly three mines:

0000111
0001101
0001110, etc.

Solving CSPs

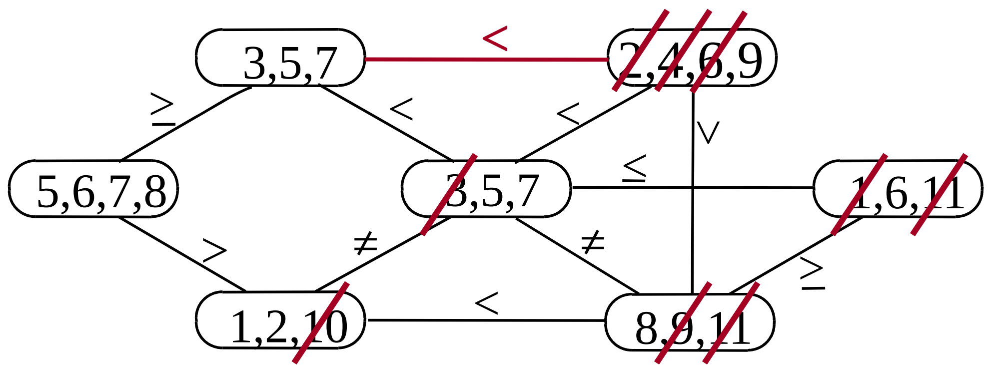
1. Constraint propagation
2. Search

Look-ahead:
propagate while searching

3. Islands of tractability
   – Special constraint types (e.g., linear inequalities)
   – Special graph structures (e.g., bounded width)

Constraint propagation

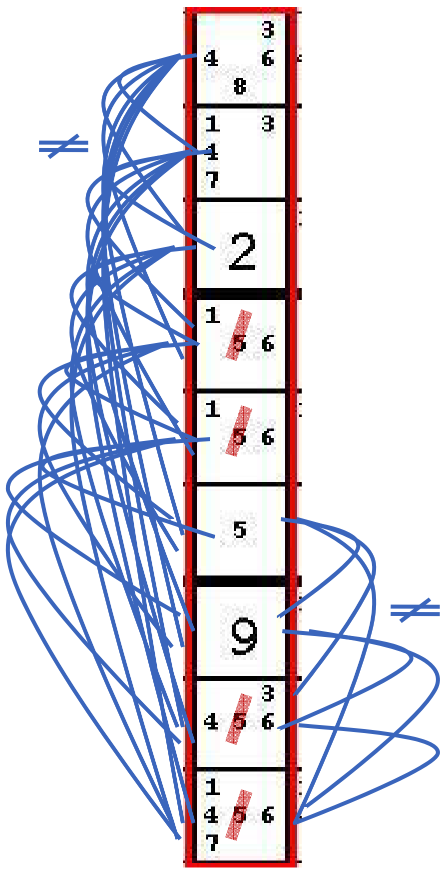
- Removes from the problem values (or combinations of values) that are inconsistent with the constraints



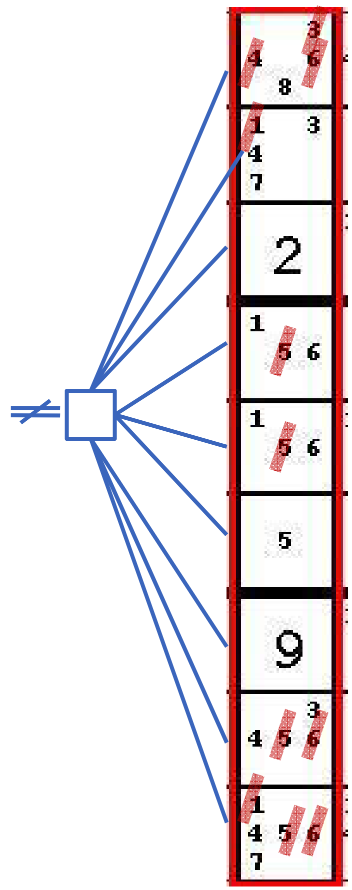
- Does not eliminate any solution

Consistency algorithms: examples

- Arc Consistency (AC)

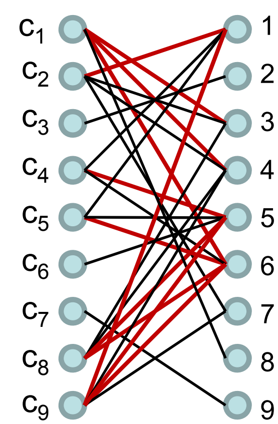


- Generalized AC (GAC)



GAC on AllDiff [Régin, 94]

- Arcs that do not appear in any matching that saturates the variables correspond to variable-value pairs that cannot appear in any solution
- GAC on AllDiff is poly time

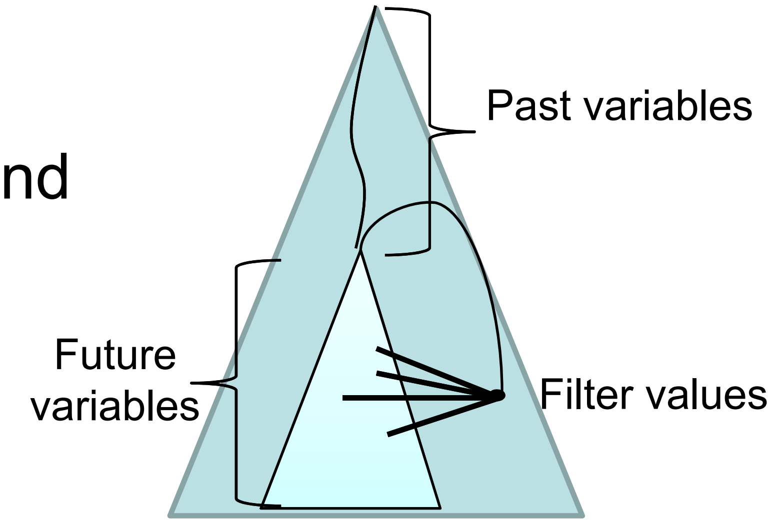


Levels of consistency

- Properties & algorithms for achieving them
  - In general, efficient (polynomial time)
  - Applicable to arbitrary constraints
  - Dedicated to specific constraint types
    - Basis for Constraint Programming (e.g., AllDiff)
- Examples on the Sudoku Solver
  - sudoku.unl.edu/Solver [with Reeson, 07]
  - Conjecture: SGAC solves every 9x9 well-formed Sudoku

Search

1. Backtrack search
   - Constructive
   - Complete (in theory) and sound
   - Note:
     - Variable ordering (backdoor)
     - Look-ahead



2. Iterative repair (i.e., local search)
   - Repairs a complete but inconsistent assignment of values to variables by doing local repairs
   - In general, neither sound nor complete

Abstraction & Reformulation

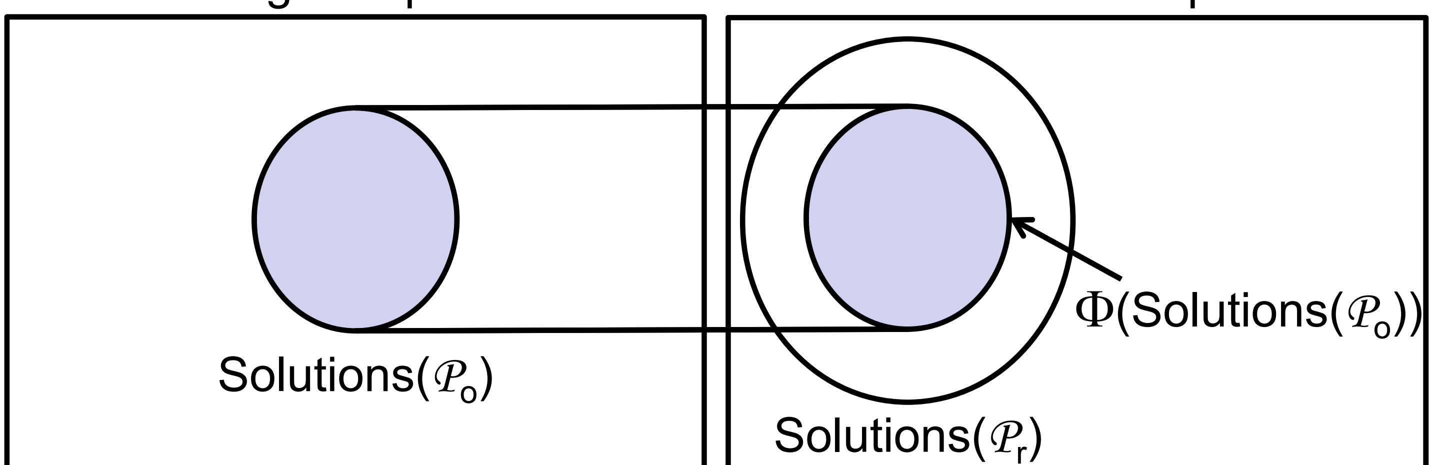
Original problem
- Original **formulation**
- Original **query**

→ *Reformulation technique* →

Reformulated problem
- Reformulated formulation
- Reformulated query

The reformulation may be an approximation

Original space | Reformulated space



Outline

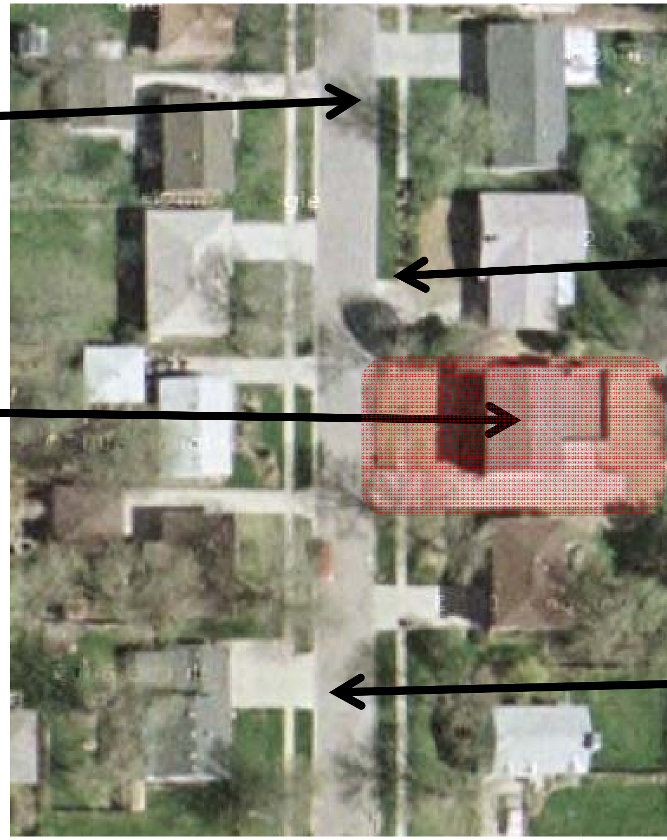
- Background
- **BID: CSP model & custom solver**
- Reformulation techniques
- Conclusions & future work

Issue: finding Ken's house

Google Maps

Yahoo Maps

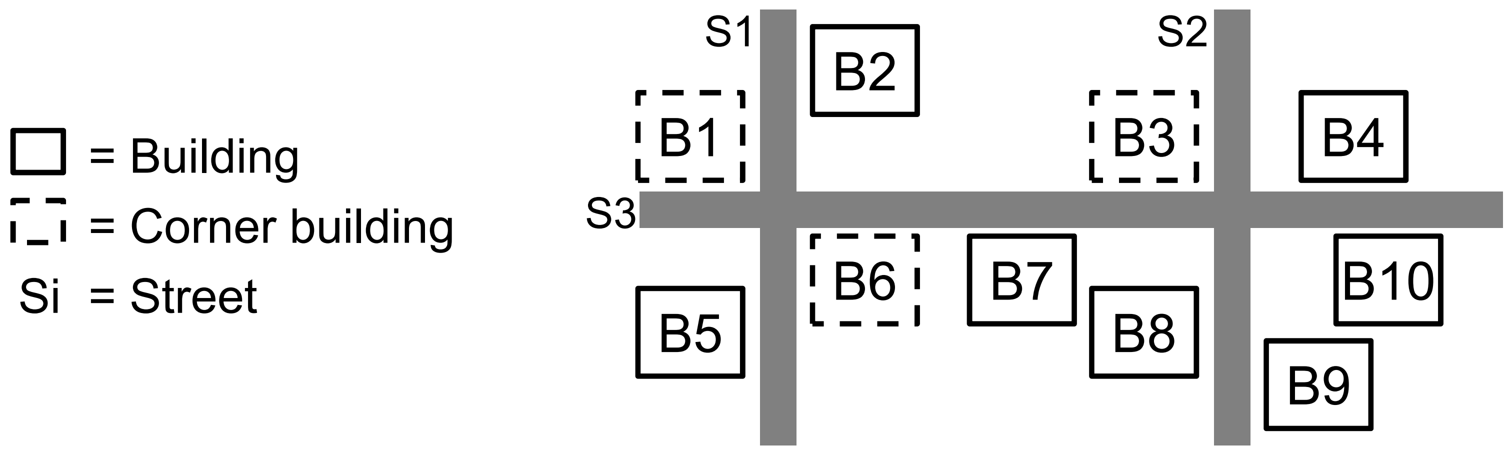
Actual location



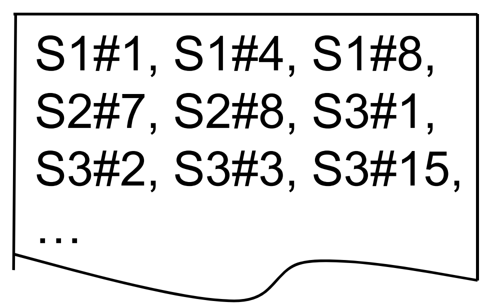
Microsoft Live Local
(as of November 2006)

Building Identification (BID) problem

- Layout: streets and buildings

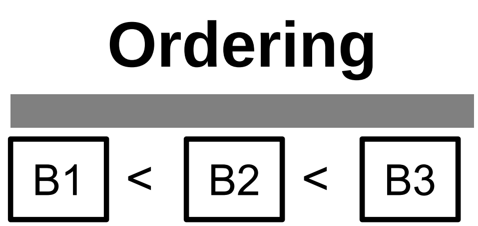


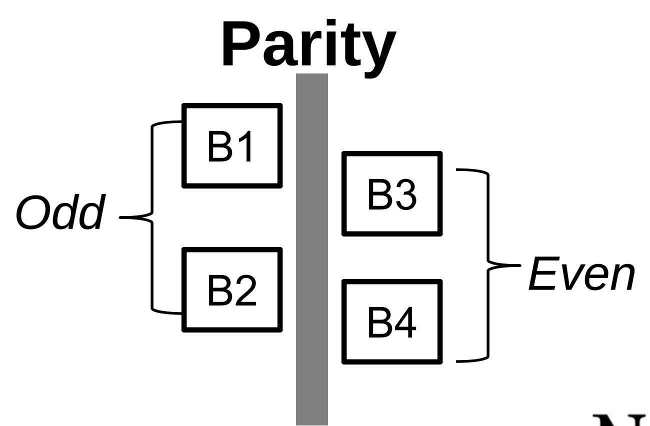
- Phone book
  - Complete/incomplete
  - Assumption: all addresses in phone book correspond to a building in the layout



Basic (address numbering) rules

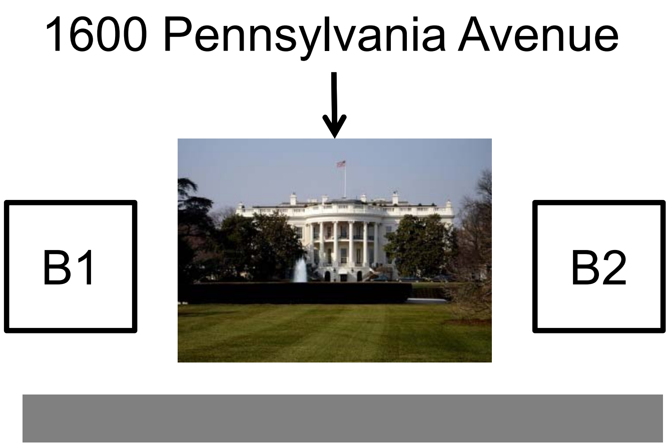
- No two buildings can have the same address
- Ordering
  - Numbers increase/decrease along a street
- Parity
  - Numbers on a given side of a street are odd/even



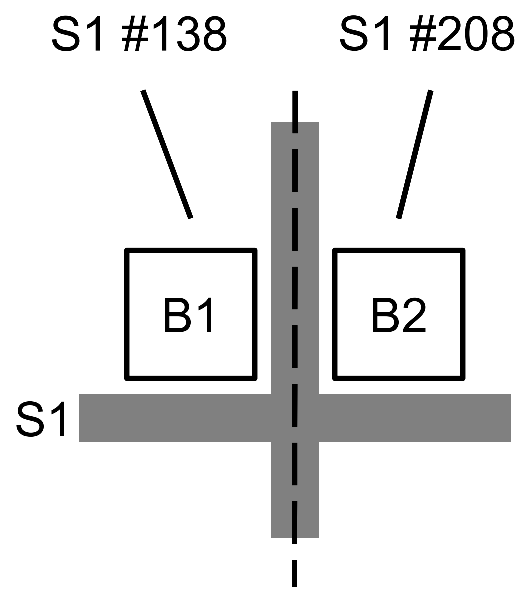


Additional information

Landmarks

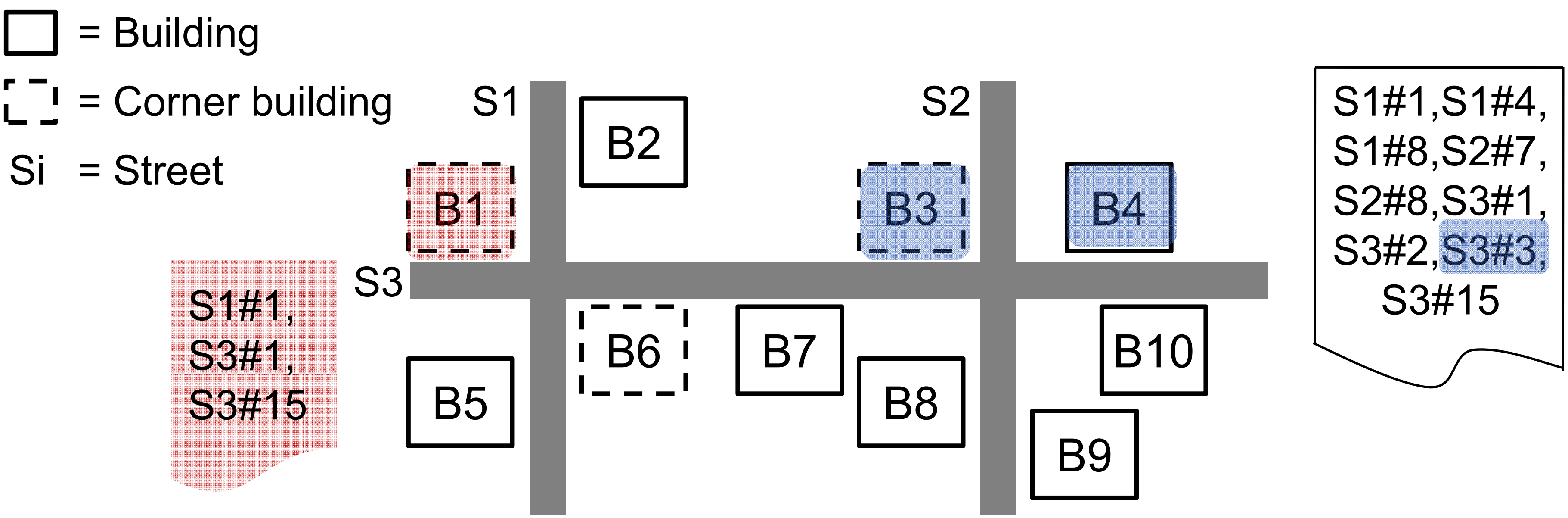


Gridlines



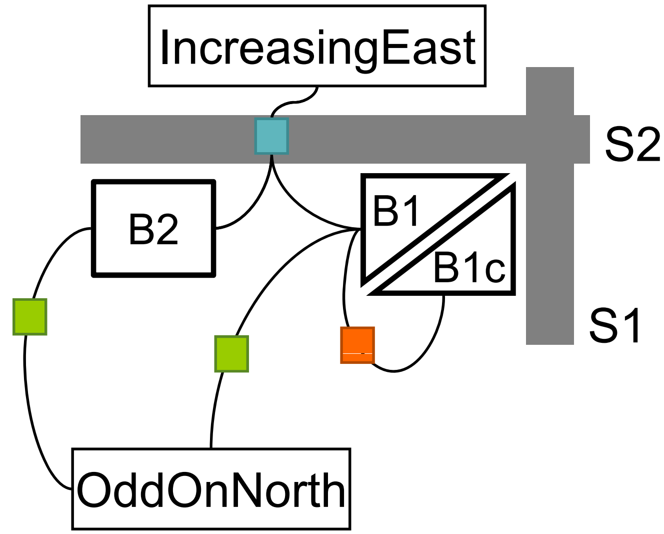
Query

1. Given an address, what buildings could it be?
2. Given a building, what addresses could it have?

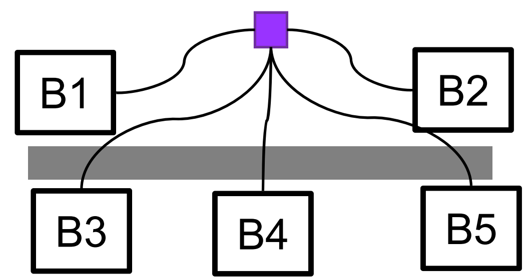


CSP model

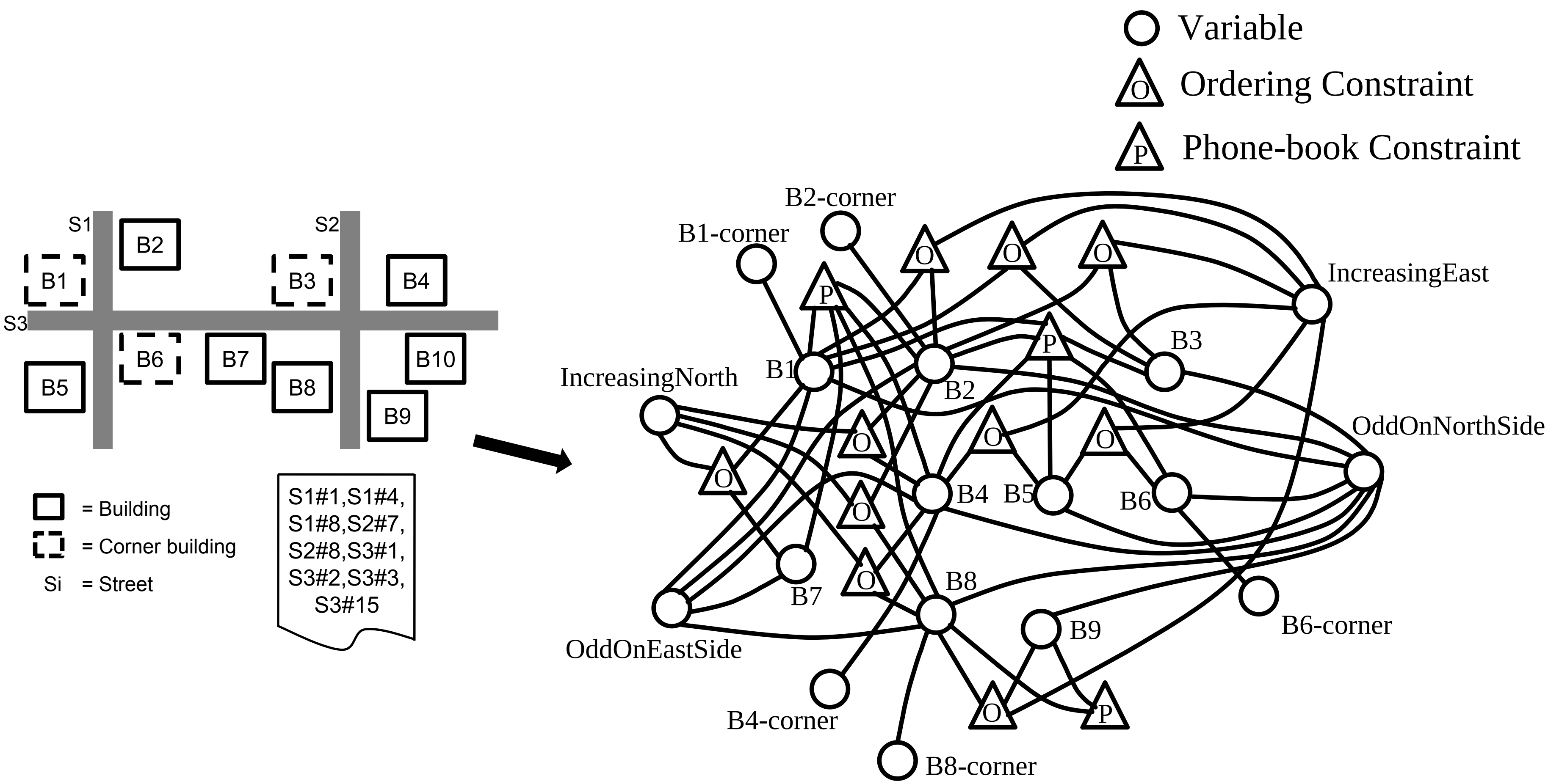
- Parity constraints
- Ordering constraints
- Corner constraints



- Phone-book constraints
- Optional: grid constraints



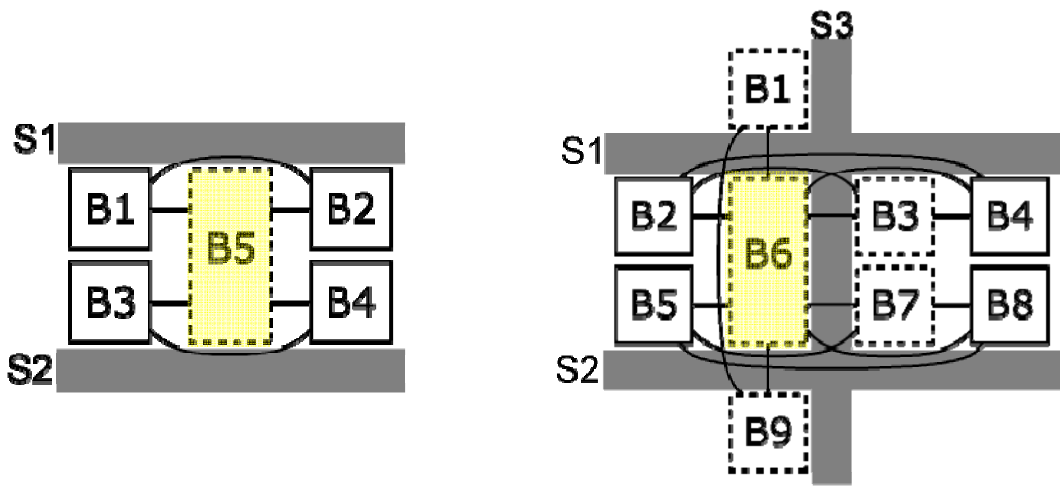
Example constraint network



Special configurations

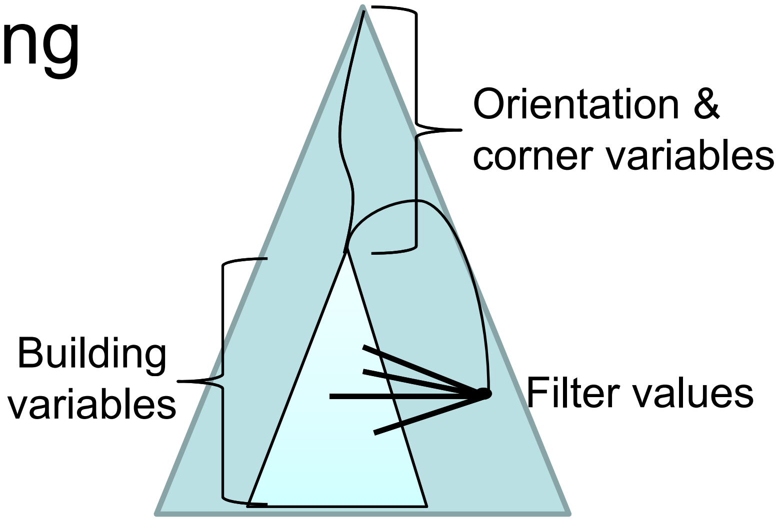
1. Orientations vary per street (e.g., Belgrade)
2. Non-corner building on two streets
3. Corner building on more than two streets

→ All gracefully handled by the model



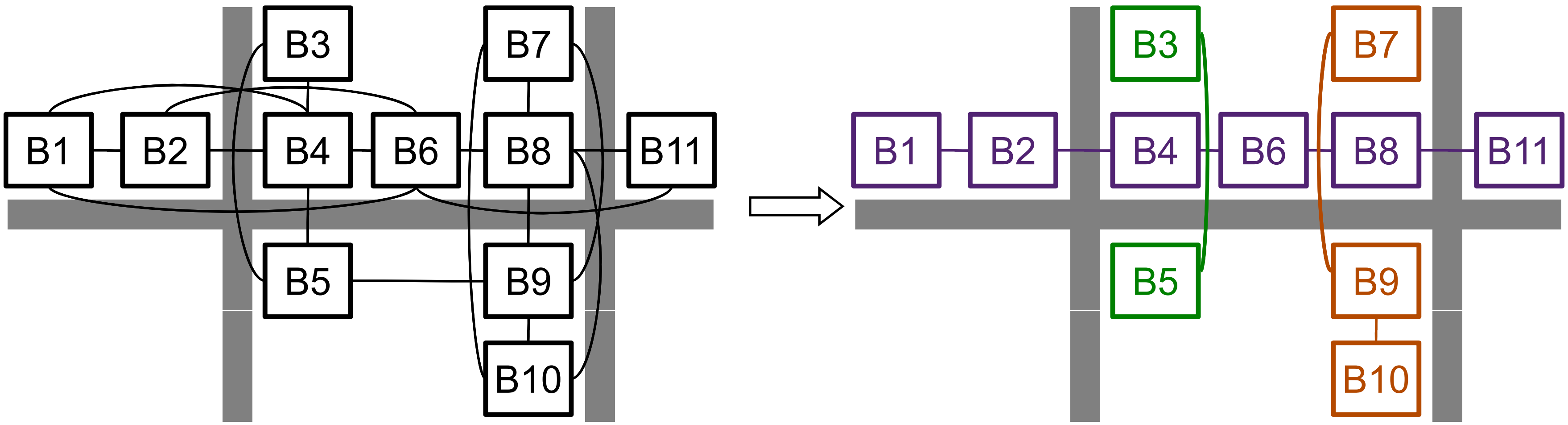
Custom solver

- Backtrack search
- Forward checking (nFC3)
- Conflict-directed backtracking
- Domains implemented as intervals (box consistency)
- Variable ordering
  1. Orientation variables
  2. Corner variables
  3. Building variables
- Backdoor variables
  - Orientation + corner variables



Backdoor variables

- We instantiate **only** orientation & corner variables



- We guarantee solvability **without** instantiating building variables

Features of new model & solver

Improvement over previous work [Michalowski +, 05]

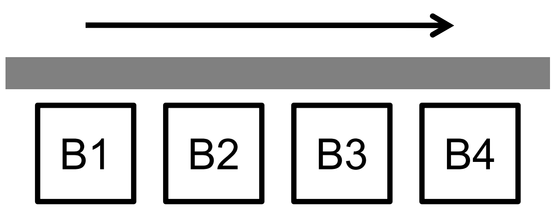
- Model
  - Reduces number of variables and constraints arity
  - Reflects topology: Constraints can be declared locally & in restricted 'contexts,' important feature for Michalowski's work
- Solver
  - Exploits structure of problem (backdoor variables)
  - Implements domains as (possibly infinite) intervals
  - *Incorporates all reformulations (to be introduced)*

Outline

- Background
- BID model & custom solver
- Reformulation techniques
  - **Query reformulation**
  - AllDiff-Atmost & domain reformulation
  - Constraint relaxation
  - Reformulation via symmetry detection
- Conclusions & future work

Query in the Building Identification Problem

- Problem: BID instances have many solutions

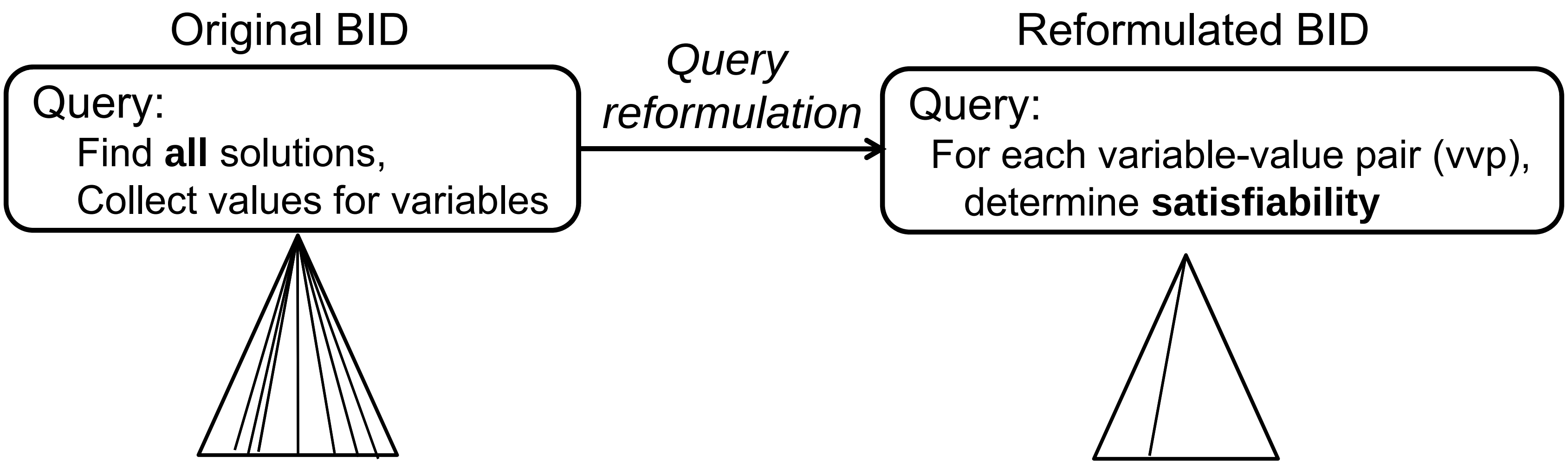


Phone book: {4,8}

B1	B2	B3	B4
2	4	6	8
2	4	8	10
2	4	8	12
4	8	10	12
4	6	8	10
4	6	8	12

We **only** need to know which values (address) appear in ***at least one*** solution for a variable (building)

Query reformulation



Original query	**Reformulated query** / For every variable-value pair (vvp)
Single enumeration problem	Many satisfiability problems / Consider CSP + vvp
All solutions	One solution per variable-value pair / Find one solution using BT search
Exhaustive search	One path
Impractical when there are many solutions	Costly when there are few solutions

Evaluations: real-world data from El Segundo

[Shewale]

Case study	Phone book	Number of…		
	Completeness	Buildings	Corner buildings	Blocks
NSeg125-c NSeg125-i	100.0% 45.6%	125	17	4
NSeg206-c NSeg206-I	100.0% 50.5%	206	28	7
SSeg131-c SSeg131-i	100.0% 60.3%	131	36	8
SSeg178-c SSeg178-i	100.0% 65.6%	178	46	12

Previous work did not scale up beyond 34 7 1

Evaluation: query reformulation

Incomplete phone book → many solutions → better performance

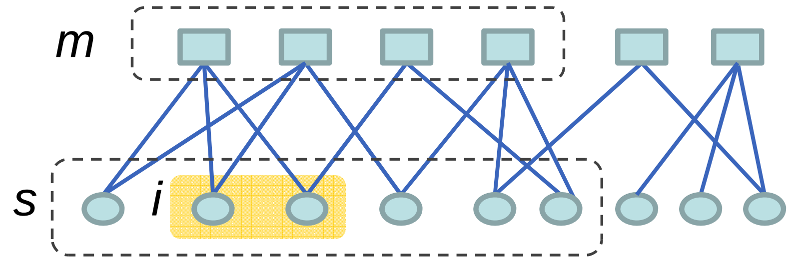
Case study	Original query	New query [s]
NSeg125-i	>1 week	744.7
NSeg206-i	>1 week	14,818.9
SSeg131-i	>1 week	66,901.1
SSeg178-i	>1 week	119,002.4

Complete phone book → few solutions → worse performance

Case study	Original query [s]	New query [s]
NSeg125-c	1.5	139.2
NSeg206-c	20.2	4,971.2
SSeg131-c	1123.4	38,618.4
SSeg178-c	3291.2	117,279.1

Generalizing query reformulation

- Relational (i,m)-consistency, algorithm R(i,m)C
  - For every m constraints
    - Compute **all solutions** of length s
    - To generate tuples of length i
  - Space: $O(d^s)$



- Query reformulation for Relational (i,m)-consistency
  - For each combination of values for i variables
    - Try to extend to **one** solution of length s
  - Space: $O(\binom{s}{i} d^i)$, $i < s$
- Reformulated BID query is R(1,$|\mathcal{C}|$)C

Application to Minesweeper

- Current implementation [with Bayer & Snyder, 06] of Minesweeper achieves
  - R(1,1)C ≡ GAC
  - R(1,2)C
  - R(1,3)C
  - By generates all solutions of length *s*

- On-going [with Woodward]

  Use query reformulation to compute R(1,*x*)C for *x*>3

Outline

- Background
- BID model & custom solver
- Reformulation techniques
  - Query reformulation
  - **AllDiff-Atmost & domain reformulation**
  - Constraint relaxation
  - Reformulation via symmetry detection
- Conclusions & future work

AllDiff-Atmost in the BID

Even side B1 B2 B3 B4 B5

Phone book: {12,48}

Original domain = {2, 4, …, 998, 1000}

12	**48**	30	32	34
12	14	16	38	**48**
10	**12**	14	20	**48**
2	4	6	**12**	**48**
…	…	**12**	**48**	**…**

- Can use at most
  - **3** addresses in [2,12) AllDiff-Atmost({B1,B2,..,B5},3,[2,12))
  - **3** addresses in (12,48) AllDiff-Atmost({B1,B2,..,B5},3,(12,48))
  - **3** addresses in (48,1000] AllDiff-Atmost({B1,B2,..,B5},3,(48,1000))

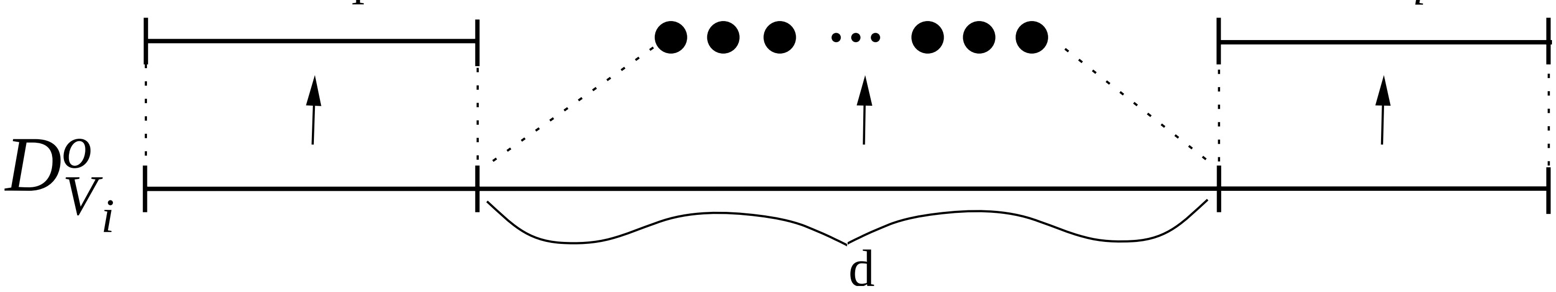
Reformulated domain $\{s_1, s_2, s_3, 12, s_4, s_5, s_6, 48, s_7, s_8, s_9\}$

Original domain { 2, 4, …, 10, 12, 14, …, 46, 48, 30, …, 998, 1000 }

AllDiff-Atmost reformulation

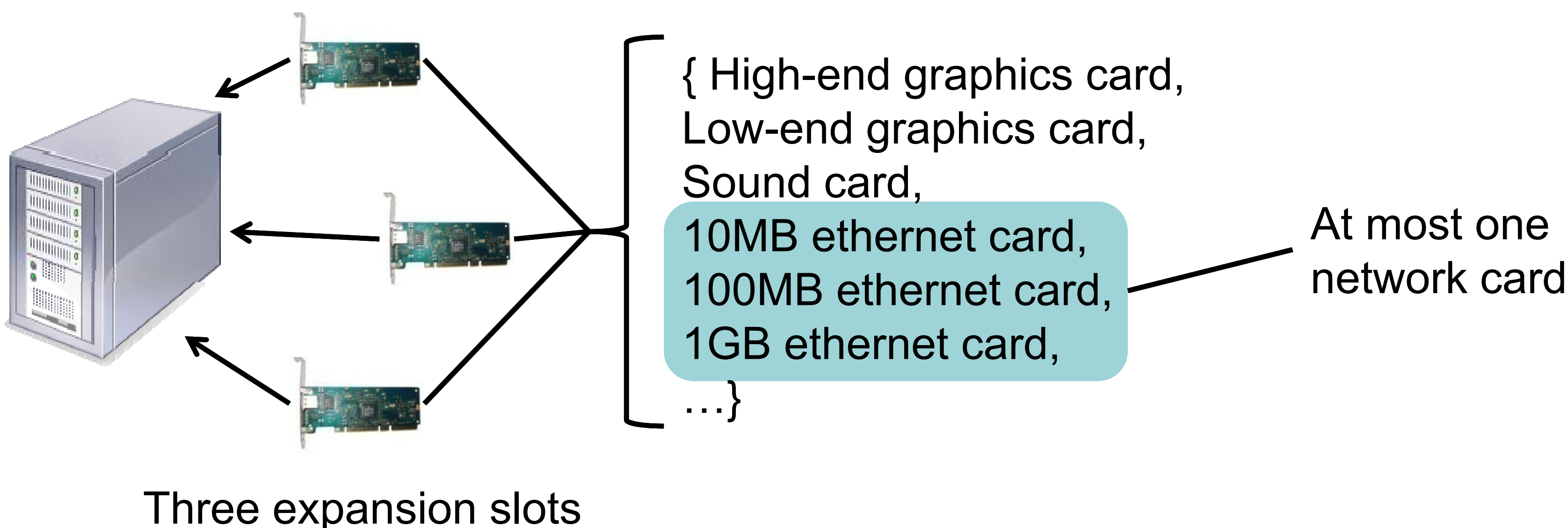
- Given AllDiff-Atmost($\mathcal{A}$,*k*,*d*)
  - The variables in $\mathcal{A}$ can be assigned at most *k* values from the set *d*
- Replace
  - interval *d* of values (potentially infinite)
  - with *k* **symbolic values**

$$D^{ref}_{V_i} = D^{ref,l}_{V_i} \cup \{s_1, s_2, \ldots, s_k\} \cup D^{ref,r}_{V_i}$$



AllDiff-Atmost constraint

- AllDiff-Atmost($\mathcal{A}$,*k*,*d*)
  - The variables in $\mathcal{A}$ can be assigned at most *k* values from the set *d*



Evaluation: domain reformulation

- Reduced domain size → improved search performance

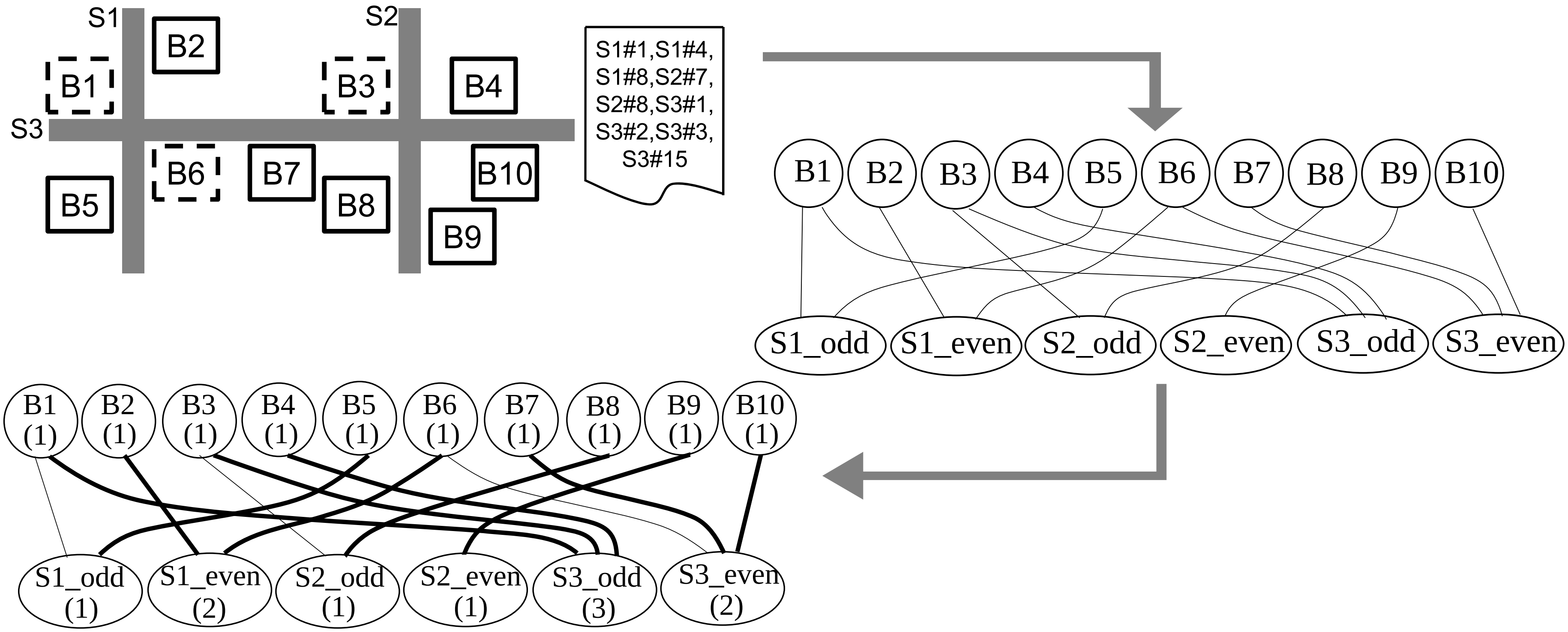
Case study	Phone-book completeness	Average domain size		Runtime [s]	
		Original	Reformulated	Original	Reformulated
NSeg125-i	45.6%	1103.1	236.1	2943.7	744.7
NSeg206-i	50.5%	1102.0	438.8	14,818.9	5533.8
SSeg131-i	60.3%	792.9	192.9	67,910.1	66,901.1
SSeg178-i	65.6%	785.5	186.3	119,002.4	117,826.7

Outline

- Background
- BID model & custom solver
- Reformulation techniques
  - Query reformulation
  - AllDiff-Atmost & domain reformulation
  - **Constraint relaxation**
  - Reformulation via symmetry detection
- Conclusions & future work

BID as a matching problem

- Assume we have no grid constraints



- Original BID is in **P**

BID w/o grid constraints

- BID instances without grid constraints can be solved in ***polynomial time***

Case study	Runtime [s]	
	BT search	Matching
NSeg125-c	139.2	4.8
NSeg206-c	4971.2	16.3
SSeg131-c	38618.3	7.3
SSeg178-c	117279.1	22.5
NSeg125-i	744.7	2.5
NSeg206-i	5533.8	8.5
SSeg131-i	38618.3	7.3
SSeg178-i	117826.7	4.9

BID w/ grid constraints

Matching reformulation exploited in two ways:

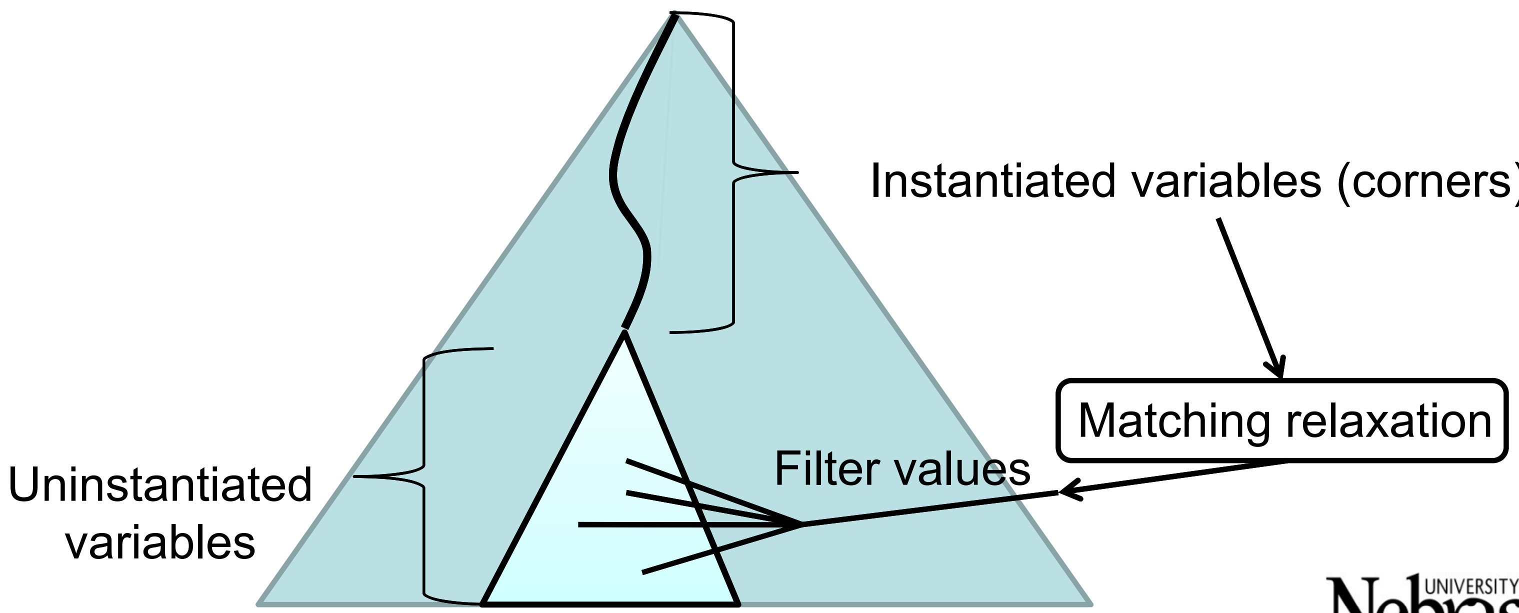
1. Domain filtering à la GAC of [Régin, 94]
   Edges that do not appear in any maximal matching indicate the values that can be filtered out from the domains

2. Constraint-model relaxation
   Ignoring the grid constraint yields a necessary approximation of the BID

Filtering the CSP

Remove variable-value pairs that do not appear in any maximum matching

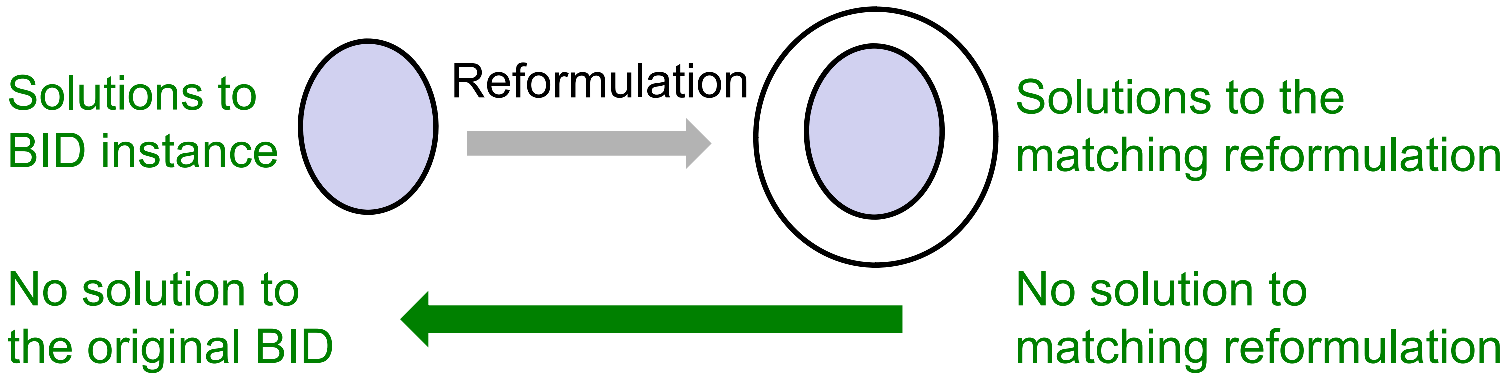
- Before search: Preprocessing 1
- During search: Look-ahead



Approximating the BID

Relaxed CSP is a *necessary approximation* of the BID

Preprocessing 2



Matching reformulation in Solver

Filter CSP.. **Preproc1**

For every variable-value pair

Consider CSP + variable-value pair

If relaxed CSP is solvable **Preproc2**

Find one solution using BT search

At each instantiation, filter CSP **Lookahead**

Evaluation: matching reformulation

- Generally, improves performance

Case Study	BT	Preproc2 +BT	% (from BT)	Lkhd +BT	% (from BT)	Lkhd +Preproc1&2 + BT	% (from Lkhd+BT)
NSeg125-i	1232.5	1159.1	**6.0%**	726.6	**41.0%**	701.1	**3.5%**
NSeg206-c	2277.5	614.2	**73.0%**	1559.2	**31.5%**	443.8	**71.5%**
SSeg178-i	138404.2	103244.7	**25.4%**	121492.4	**12.2%**	85185.9	**29.9%**

- Rarely, the overhead exceeds the gains

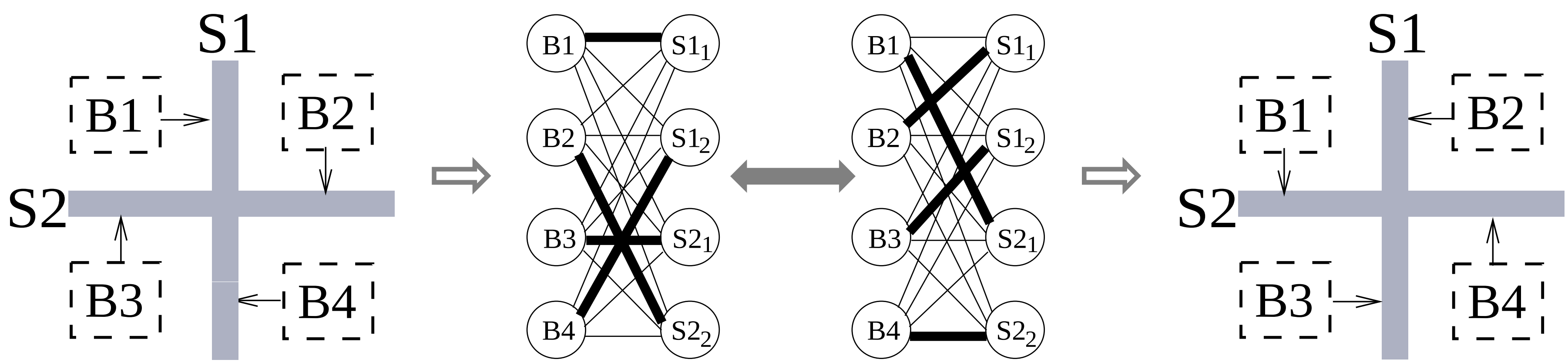
Case Study	BT	Preproc2 +BT	% (from BT)	Lkhd +BT	% (from BT)	Lkhd +Preproc1&2 + BT	% (from Lkhd+BT)
NSeg125-c	100.8	33.2	**67.1%**	**140.2**	**-39.0%**	29.8	**78.7%**
NSeg131-i	114405.9	114141.3	**0.2%**	107896.3	**5.7%**	**108646.6**	**-0.7%**

Outline

- Background
- BID model & custom solver
- Reformulation techniques
  - Query reformulation
  - AllDiff-Atmost & domain reformulation
  - Constraint relaxation
  - **Reformulation via symmetry detection**
- Conclusions & future work

Symmetric solutions in BID

- Exploring symmetric solutions is time consuming

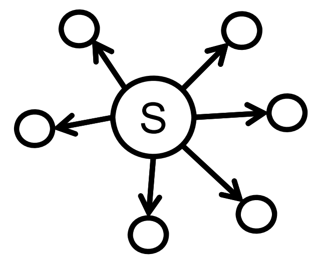


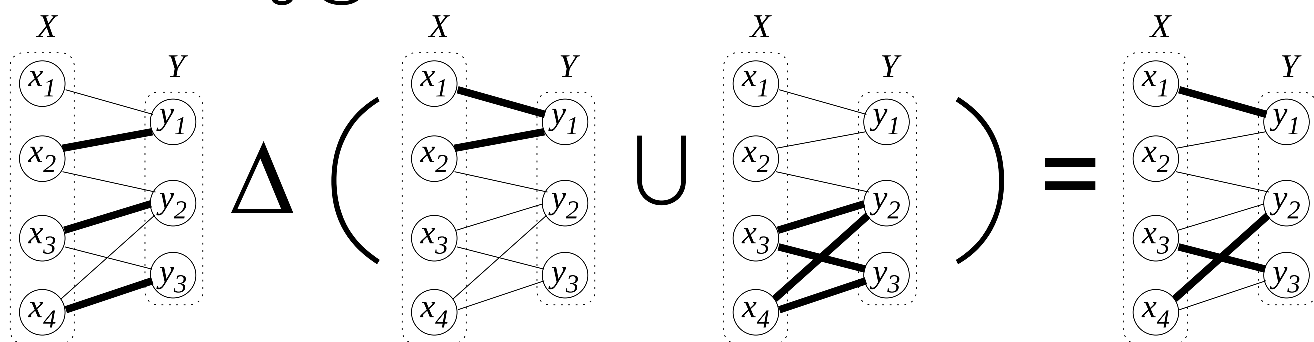
- **Goal**: break symmetries to improve scalability

Hot topic in CP

Symmetric maximum matchings

- ***All*** matchings can be produced from the symmetric difference of
  - a single matching and
  - a set of disjoint alternating cycles & paths starting @ free vertex





- Some symmetric solutions do not break grid constraints
  - Ignore symmetric solutions during search
- Some do, we do not know how to use them…

Conclusions

- We showed that the original BID problem is in **P**
- We proposed four reformulation techniques
- We described their usefulness for general CSPs
- We demonstrated their effectiveness on the BID

Lesson:

Reformulation is an effective approach to improve the scalability of complex combinatorial systems

Future work

- Empirically evaluate our new algorithm for relational *(i,m)-consistency*
- Exploit the symmetries we identified
- Enhance the model by incorporating new constraints [Michalowski]

Questions?