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** \( P = (V, D, C) \)
  - \( V \): set of variables
  - \( D \): set of their domains
  - \( C \): 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,\ldots,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:

\[ 0000011 \]
\[ 0000101 \]
\[ 0000110, \text{ etc.} \]

Exactly three mines:

\[ 0000111 \]
\[ 0000111 \]
\[ 0001101 \]
\[ 0001110, \text{ etc.} \]
Solving CSPs

1. Constraint propagation
   Look-ahead: propagate while searching

2. Search

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

Solutions($P_o$)

Reformulated space

$\Phi$(Solutions($P_o$))

Solutions($P_i$)
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

□ = Building
- = Corner building
Si = Street

S1#1, S1#4, S1#8, S2#7, S2#8, S3#1, S3#2, S3#3, S3#15, ...

Constraint Systems Laboratory

10/16/2007 Math Colloquium
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

1600 Pennsylvania Avenue

Gridlines

S1 #138  S1 #208

B1  B2

Constraint Systems Laboratory

10/16/2007  Math Colloquium
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

10/16/2007
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

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

**Original BID**

**Query:**
- Find all solutions,
- Collect values for variables

---

**Reformulated BID**

**Query:**
- For each variable-value pair (vvp),
  - determine **satisfiability**

<table>
<thead>
<tr>
<th>Original query</th>
<th>Reformulated query</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single</strong> enumeration problem</td>
<td>Consider CSP - vvp</td>
</tr>
<tr>
<td>All solutions</td>
<td>One solution per variable-value pair by BT search</td>
</tr>
<tr>
<td>Exhaustive search</td>
<td>One path</td>
</tr>
<tr>
<td>Impractical when there are many solutions</td>
<td>Costly when there are few solutions</td>
</tr>
</tbody>
</table>

---

**Constraint Systems Laboratory**

10/16/2007 Math Colloquium
Evaluations: real-world data from El Segundo

<table>
<thead>
<tr>
<th>Case study</th>
<th>Phone book</th>
<th>Completeness</th>
<th>Buildings</th>
<th>Corner buildings</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSeg125-c</td>
<td></td>
<td>100.0%</td>
<td>125</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td>NSeg125-i</td>
<td></td>
<td>45.6%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NSeg206-c</td>
<td></td>
<td>100.0%</td>
<td>206</td>
<td>28</td>
<td>7</td>
</tr>
<tr>
<td>NSeg206-I</td>
<td></td>
<td>50.5%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSeg131-c</td>
<td></td>
<td>100.0%</td>
<td>131</td>
<td>36</td>
<td>8</td>
</tr>
<tr>
<td>SSeg131-i</td>
<td></td>
<td>60.3%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SSeg178-c</td>
<td></td>
<td>100.0%</td>
<td>178</td>
<td>46</td>
<td>12</td>
</tr>
<tr>
<td>SSeg178-i</td>
<td></td>
<td>65.6%</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Previous work did not scale up beyond

34 7 1
Evaluation: query reformulation

Incomplete phone book $\rightarrow$ many solutions $\rightarrow$ better performance

<table>
<thead>
<tr>
<th>Case study</th>
<th>Original query [s]</th>
<th>New query [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSeg125-i</td>
<td>&gt;1 week</td>
<td>744.7</td>
</tr>
<tr>
<td>NSeg206-i</td>
<td>&gt;1 week</td>
<td>14,818.9</td>
</tr>
<tr>
<td>SSeg131-i</td>
<td>&gt;1 week</td>
<td>66,901.1</td>
</tr>
<tr>
<td>SSeg178-i</td>
<td>&gt;1 week</td>
<td>119,002.4</td>
</tr>
</tbody>
</table>

Complete phone book $\rightarrow$ few solutions $\rightarrow$ worse performance

<table>
<thead>
<tr>
<th>Case study</th>
<th>Original query [s]</th>
<th>New query [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSeg125-c</td>
<td>1.5</td>
<td>139.2</td>
</tr>
<tr>
<td>NSeg206-c</td>
<td>20.2</td>
<td>4,971.2</td>
</tr>
<tr>
<td>SSeg131-c</td>
<td>1123.4</td>
<td>38,618.4</td>
</tr>
<tr>
<td>SSeg178-c</td>
<td>3291.2</td>
<td>117,279.1</td>
</tr>
</tbody>
</table>
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: \(O\left(\binom{s}{i}d^i\right), i < s\)

- Reformulated BID query is \(R(1,|C|)C\)
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**

<table>
<thead>
<tr>
<th>Even side</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
<th>B5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Phone book: \{12, 48\}

Original domain = \{2, 4, ..., 998, 1000\}

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

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

\[
D_{V_i}^{ref} = D_{V_i}^{ref,l} \cup \{S_1, S_2, \ldots, S_k\} \cup D_{V_i}^{ref,r}
\]
AllDiff-Atmost constraint

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

Three expansion slots

\{ High-end graphics card, Low-end graphics card, Sound card, 10MB ethernet card, 100MB ethernet card, 1GB ethernet card, … \}

At most one network card
Evaluation: domain reformulation

- Reduced domain size → improved search performance

<table>
<thead>
<tr>
<th>Case study</th>
<th>Phone-book completeness</th>
<th>Average domain size</th>
<th>Runtime [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original</td>
<td>Reformulated</td>
</tr>
<tr>
<td>NSeg125-i</td>
<td>45.6%</td>
<td>1103.1</td>
<td>236.1</td>
</tr>
<tr>
<td>NSeg206-i</td>
<td>50.5%</td>
<td>1102.0</td>
<td>438.8</td>
</tr>
<tr>
<td>SSeg131-i</td>
<td>60.3%</td>
<td>792.9</td>
<td>192.9</td>
</tr>
<tr>
<td>SSeg178-i</td>
<td>65.6%</td>
<td>785.5</td>
<td>186.3</td>
</tr>
</tbody>
</table>
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 \( \mathcal{P} \)
BID w/o grid constraints

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

<table>
<thead>
<tr>
<th>Case study</th>
<th>Runtime [s]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BT search</td>
<td>Matching</td>
<td></td>
</tr>
<tr>
<td>NSeg125-c</td>
<td>139.2</td>
<td>4.8</td>
<td></td>
</tr>
<tr>
<td>NSeg206-c</td>
<td>4971.2</td>
<td>16.3</td>
<td></td>
</tr>
<tr>
<td>SSeg131-c</td>
<td>38618.3</td>
<td>7.3</td>
<td></td>
</tr>
<tr>
<td>SSeg178-c</td>
<td>117279.1</td>
<td>22.5</td>
<td></td>
</tr>
<tr>
<td>NSeg125-i</td>
<td>744.7</td>
<td>2.5</td>
<td></td>
</tr>
<tr>
<td>NSeg206-i</td>
<td>5533.8</td>
<td>8.5</td>
<td></td>
</tr>
<tr>
<td>SSeg131-i</td>
<td>38618.3</td>
<td>7.3</td>
<td></td>
</tr>
<tr>
<td>SSeg178-i</td>
<td>117826.7</td>
<td>4.9</td>
<td></td>
</tr>
</tbody>
</table>
Matching reformulation exploited in two ways:

1. **Domain filtering**
   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

- Solutions to BID instance
- Reformulation
- Solutions to the matching reformulation

No solution to the original BID

No solution to matching reformulation
Matching reformulation in Solver

Filter CSP

For every variable-value pair
Consider CSP + variable-value pair
If relaxed CSP is solvable
Find one solution using BT search
At each instantiation, filter CSP

Preproc1
Preproc2
Lookahead
**Evaluation: matching reformulation**

- Generally, improves performance

<table>
<thead>
<tr>
<th>Case Study</th>
<th>BT</th>
<th>Preproc2 +BT</th>
<th>% (from BT)</th>
<th>Lkhd +BT</th>
<th>% (from BT)</th>
<th>Lkhd +Preproc1&amp;2 + BT</th>
<th>% (from Lkhd+BT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSeg125-i</td>
<td>1232.5</td>
<td>1159.1</td>
<td>6.0%</td>
<td>726.6</td>
<td>41.0%</td>
<td>701.1</td>
<td>3.5%</td>
</tr>
<tr>
<td>NSeg206-c</td>
<td>2277.5</td>
<td>614.2</td>
<td>73.0%</td>
<td>1559.2</td>
<td>31.5%</td>
<td>443.8</td>
<td>71.5%</td>
</tr>
<tr>
<td>SSag178-i</td>
<td>138404.2</td>
<td>103244.7</td>
<td>25.4%</td>
<td>121492.4</td>
<td>12.2%</td>
<td>85185.9</td>
<td>29.9%</td>
</tr>
</tbody>
</table>

- Rarely, the overhead exceeds the gains

<table>
<thead>
<tr>
<th>Case Study</th>
<th>BT</th>
<th>Preproc2 +BT</th>
<th>% (from BT)</th>
<th>Lkhd +BT</th>
<th>% (from BT)</th>
<th>Lkhd +Preproc1&amp;2 + BT</th>
<th>% (from Lkhd+BT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSeg125-c</td>
<td>100.8</td>
<td>33.2</td>
<td>67.1%</td>
<td>140.2</td>
<td>-39.0%</td>
<td>29.8</td>
<td>78.7%</td>
</tr>
<tr>
<td>NSeg131-i</td>
<td>114405.9</td>
<td>114141.3</td>
<td>0.2%</td>
<td>107896.3</td>
<td>5.7%</td>
<td>108646.6</td>
<td>-0.7%</td>
</tr>
</tbody>
</table>
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
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 at 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…

\[ \Delta \left( \bigcup \right) = \]
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?