Chapter 8: Symmetry Breaking (Balanced Incomplete Block Designs)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning

Licence

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.
Outline

1. Problem
2. Program
3. Symmetry Breaking

What we want to introduce

- BIBD - Balanced Incomplete Block Designs
- Using lex constraints to remove symmetries
- Only one of many ways to deal with symmetry in problems
- Finding all solutions to a problem
- Using timeout to limit search
Problem Definition

BIBD (Balanced Incomplete Block Design)

A BIBD is defined as an arrangement of \( v \) distinct objects into \( b \) blocks such that each block contains exactly \( k \) distinct objects, each object occurs in exactly \( r \) different blocks, and every two distinct objects occur together in exactly \( \lambda \) blocks. A BIBD is therefore specified by its parameters \((v, b, r, k, \lambda)\).

Motivation: Test Planning

Consider a new release of some software with \( v \) new features. You want to regression test the software against combinations of the new features. Testing each subset of features is too expensive, so you want to run \( b \) tests, each using \( k \) features. Each feature should be used \( r \) times in the tests. Each pair of features should be tested together exactly \( \lambda \) times. How do you arrange the tests?
Another way of defining a BIBD is in terms of its incidence matrix, which is a binary matrix with $v$ rows, $b$ columns, $r$ ones per row, $k$ ones per column, and scalar product $\lambda$ between any pair of distinct rows.

A binary $v \times b$ matrix. Entry $V_{ij}$ states if item $i$ is in block $j$.

- Sum constraints over rows, each sum equal $r$
- Sum constraints over columns, each sum equal $k$
- Scalar product between any pair of rows, the product value is $\lambda$. 
Top Level Program

```prolog
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
    bibd(6,10,5,3,2,Matrix), writeln(Matrix).
```

```prolog
bibd(V,B,R,K,L,Matrix):-
    model(V,B,R,K,L,Matrix),\ Set up model
    extract_array(row,Matrix,List),\ Get list
    search(L,0,input_order,indomain,complete,[]).\ Search
```

Constraint Model

```prolog
model(V,B,R,K,L,Matrix,Method):-
    dim(Matrix,[V,B]),\ Define Binary Matrix
    Matrix[1..V,1..B] :: 0..1,
    (for(I,1,V), param(Matrix,B,R) do
        sumlist(Matrix[I,1..B],R)
    ),\ Row Sum = R
    (for(J,1,B), param(Matrix,V,K) do
        sumlist(Matrix[1..V,J],K)
    ),\ Column Sum = K
    (for(I,1,V-1), param(Matrix,V,B,L) do
        (for(I1,I+1,V), param(Matrix,I,B,L) do
            scalar_product(Matrix[I,1..B], Matrix[I1,1..B],L)
        )
    )\ Scalar product between all rows
```
scalar_product(XVector, YVector, V):-
collection_to_list(XVector, XList),
collection_to_list(YVector, YList), % Get lists
( foreach(X, XList), % Iterate over lists
  foreach(Y, YList), % ...in parallel
  fromto(0, A, A1, Term) do % Build term
    A1 = A+X*Y % Construct term
  ),
eval(Term) #= V. % State Constraint

Search Routine

- Static variable order
- First fail does not work for binary variables
- Enumerate variables by row
- Use utility predicate extract_array/3
- Assign with indomain, try value 0, then value 1
- Use simple search call
Basic Model - First Solution

Finding all solutions - Hack!

:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
    bibd(6,10,5,3,2,Matrix), writeln(Matrix),
    fail. ⇔ Force Backtracking

bibd(V,B,R,K,L,Matrix):-
    model(V,B,R,K,L,Matrix),
    extract_array(row,Matrix,List),
    search(L,0,input_order,indomain,complete,[]).
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
findall(Matrix,bibd(6,10,5,3,2,Matrix),Sols),
writeln(Sols).

bibd(V,B,R,K,L,Matrix):-
model(V,B,R,K,L,Matrix),
extract_array(row,Matrix,List),
search(L,0,input_order,indomain,
complete,[]).

**findall** predicate

- findall(Template,Goal,Collection)
- Finds all solutions to Goal and collects them into a list Collection
- Template is used to extract arguments from Goal to store as solution
- Backtracks through all choices in Goal
- Solutions are returned in order in which they are found
Problem

- Program now only stops when it has found all solutions
- This takes too long!
- How can we limit the amount of time to wait?
- Use of the `timeout` library

Finding all solutions - Proper

```prolog
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).
:-lib(timeout). % Load library

top:-
    findall(Matrix, timeout(bibd(6,10,5,3,2,Matrix), 10, \to seconds fail), Sols),
    writeln(Sols).
```
timeout library

- `timeout(Goal, Limit, TimeoutGoal)`
- Runs `Goal` for `Limit` seconds
- If `Limit` is reached, `Goal` is stopped and `TimeoutGoal` is run instead
- If `Limit` is not reached, it has no impact
- Must load `:-lib(timeout).`
Surprise! There are many solutions

Search Tree 300 Nodes
Symmetry Breaking

Search Tree 400 Nodes

Helmut Simonis  Symmetry Breaking  23

Search Tree 500 Nodes

Helmut Simonis  Symmetry Breaking  24
Problem

- There are too many solutions to collect in a reasonable time
- Most of these solutions are very similar
- If you take one solution and
  - exchange two rows
  - and/or exchange two columns
- ... you have another solution
- Can we avoid exploring them all?

Symmetry Breaking Techniques

- Remove all symmetries
  - Reduce the search tree as much as possible
  - May be hard to describe all symmetries
  - May be expensive to remove symmetric parts of tree
- Remove some symmetries
  - Search is not reduced as much
  - May be easier to find some symmetries to remove
  - Cost can be low
Symmetry Breaking Techniques

- Symmetry removal by forcing partial, initial assignment
  - Easy to understand
  - Rather weak, does not affect search
- Symmetry removal by stating constraints
  - Removing all symmetries may require exponential number of constraints
  - Can conflict with search strategies
- Symmetry removal by controlling search
  - At each node, decide if it needs to be explored
  - Can be expensive to check

Solution used here: Double Lex

- Partial symmetry removal by adding lexicographical ordering constraints
- Our problem has full row and column symmetries
- Any permutation of rows and/or columns leads to another solution
- Idea: Order rows lexicographically
- Rows must be different from each other, strict order on rows
- Columns might be identical, non strict order on columns
  - This can be improved in some cases
- Constraints only between adjacent rows(columns)
Added Constraints

\[
\text{dim} (\text{Matrix}, [V,B]), \\
(\text{for} (I, 1, V-1), \\
\text{param} (\text{Matrix}, B) \text{ do} \\
\quad I_1 \text{ is } I+1, \\
\quad \text{lex\_less} (\text{Matrix}[I_1,1..B],\text{Matrix}[I,1..B]) \\
), \Rightarrow \text{ Row lex constraints} \\
(\text{for} (J, 1, B-1), \\
\text{param} (\text{Matrix}, V) \text{ do} \\
\quad J_1 \text{ is } J+1, \\
\quad \text{lex\_leq} (\text{Matrix}[1..V,J_1],\text{Matrix}[1..V,J]) \\
), \Rightarrow \text{ Column lex constraints}
\]
Example propagation \texttt{lex\_less}

Before
\begin{align*}
[ & 2, \quad X2 \in \{1, 3, 4\}, \\
[ & Y1 \in \{0, 1, 2\}, \quad 1, \\
[ & 2, \quad 1, \\
[ & 2, \quad 1, \\
\end{align*}

After
\begin{align*}
[ & 2, \quad X3 \in \{1, 2\}, \quad X4 \in \{1, 2\}, \quad X5 \in \{3, 4\}], \\
[ & Y3 \in \{0, 1, 2\}, \quad Y4 \in \{0, 1\}, \quad Y5 \in \{0, 1\}] \\
\end{align*}
Observation

- Enormous reduction in search space
- We are solving a different problem!
- Not just good for finding all solutions, also for first solution!
- Value choice not optimal for finding first solution
- There is a lot of very shallow backtracking, can we avoid that?

Effort for First Solution

Basic Model  With double Lex
### Alternative Value Order

```prolog
:-module(bibd).
:-export(top/0).
:-lib(ic).
:-lib(ic_global).

top:-
    bibd(6,10,5,3,2,Matrix), writeln(Matrix).

bibd(V,B,R,K,L,Matrix):-
    model(V,B,R,K,L,Matrix),
    extract_array(row,Matrix,List),
    search(L,0,input_order,
      indomain_max,\[ Start with 1
      complete,\].
```

### Assigning Value 1 First

```
```

Helmut Simonis  Symmetry Breaking  37

Helmut Simonis  Symmetry Breaking  38
**Observation**

- First solution is found more quickly
- Size of tree for all solutions unchanged
- Value order does not really affect search space when exploring all choices!

**Effort for All Solutions**

Assign 0, then 1  
Assign 1, then 0
Conclusions

- Symmetry breaking can have huge impact on model
- Mainly works for pure problems
- Partial symmetry breaking with additional constraints
- Double lex for row/column symmetries
- Only one variant of many symmetry breaking techniques

Row- or Column- wise Assignment?

- We did assign matrix by row, why?
- What happens if we assign variables by column?
Observation

- Good, but not as good as row order
- Value choice (0/1) or (1/0) unimportant even for first solution
- Changing the variable selection does affect size of search space, even for all solutions
Why assign by row?

Exercises

Possible Explanations

- There are fewer rows than columns
- Strict lex constraints on rows, but not on columns
  - More impact of first row
- Needs better understanding
### Why assign by row?

**Exercises**

Does this scale?

<table>
<thead>
<tr>
<th>v</th>
<th>b</th>
<th>r</th>
<th>k</th>
<th>λ</th>
<th>asym</th>
<th>(\text{lex}^2)</th>
<th>STAB</th>
<th>(\text{lex}^2 + \text{SBNO})</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>24</td>
<td>8</td>
<td>3</td>
<td>2</td>
<td>36</td>
<td>5,987</td>
<td>344</td>
<td>311</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>46</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>21</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>26</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>12,800</td>
<td>21</td>
<td>101</td>
</tr>
<tr>
<td>7</td>
<td>35</td>
<td>15</td>
<td>3</td>
<td>5</td>
<td>109</td>
<td>33,304</td>
<td>542</td>
<td>282</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>5</td>
<td>118</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>12</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>30</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>864</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>18</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>21</td>
<td>8,031</td>
<td>302</td>
<td>139</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>18</td>
<td>3</td>
<td>6</td>
<td>418</td>
<td>250,878</td>
<td>2,334</td>
<td>1,247</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>49</td>
<td>21</td>
<td>3</td>
<td>7</td>
<td>1,508</td>
<td>1,460,332</td>
<td>8,821</td>
<td>4,353</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>14</td>
<td>4</td>
<td>6</td>
<td>2,310</td>
<td>2,058,523</td>
<td>17,890</td>
<td>11,424</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>6,520</td>
<td>71</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>9</td>
<td>3</td>
<td>2</td>
<td>960</td>
<td>724,662</td>
<td>24,563</td>
<td>15,169</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>864</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>56</td>
<td>24</td>
<td>3</td>
<td>8</td>
<td>5,413</td>
<td>6,941,124</td>
<td>32,038</td>
<td>14,428</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>22,521</td>
<td>14,843,772</td>
<td>315,531</td>
<td>85,605</td>
</tr>
<tr>
<td>7</td>
<td>63</td>
<td>27</td>
<td>3</td>
<td>9</td>
<td>?</td>
<td>28,079,394</td>
<td>105,955</td>
<td>43,259</td>
</tr>
<tr>
<td>15</td>
<td>35</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>80</td>
<td>32,127,296</td>
<td>6,782</td>
<td>35,183</td>
</tr>
<tr>
<td>21</td>
<td>28</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>26</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>2461</td>
<td>3,664,243</td>
<td>83,337</td>
<td>31,323</td>
</tr>
<tr>
<td>11</td>
<td>22</td>
<td>10</td>
<td>5</td>
<td>4</td>
<td>4393</td>
<td>6,143,408</td>
<td>106,522</td>
<td>32,908</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
<td>11</td>
<td>6</td>
<td>5</td>
<td>?</td>
<td>?</td>
<td>228,146</td>
<td>76,572</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>?</td>
<td>?</td>
<td>17,016</td>
<td>1,355</td>
</tr>
<tr>
<td>16</td>
<td>24</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>?</td>
<td>?</td>
<td>769,482</td>
<td>76,860</td>
</tr>
</tbody>
</table>

---

**Scalability**

- \(\text{lex}^2\) good, but not good enough
- Still leaves too many symmetries to explore
- Better techniques in the literature
  - STAB, group theory based, Puget 2003.
  - SBNO, local search based domination check, Prestwich, 2008.
Do we need binary variables?

- The 0/1 model does very little propagation
- Consider a model with finite domain variables
- Each of $b$ blocks consists of $k$ variables ranging over $v$ values
- The values in a block must be alldifferent (ordered)
- Each value can occur $r$ times
- Scalar product more difficult
- Even better expressed with finite set variables

More Information