The Thousand Faces of Constraint Propagation

Emmanuel Hebrard

LAAS-CNRS

Toulouse
Outline

1. Propagation and Constraint Programming
2. An Example of Propagator
   - The Switch constraint
3. Complexity of Propagating vs. Complexity of Solving
   - The AtMostSeqCard constraint
4. NP-hard Constraints
   - The SoftAllEqual constraint
5. Assessing the Tradeoff
   - The NValue constraint
6. Conclusion
Constraint Programming

- Solving hard combinatorial problems by Decomposition
Constraint Programming

- Solving hard combinatorial problems by Decomposition
  - Good news: We can always decompose
    - into known/easy/small subproblems
Constraint Programming

- Solving hard combinatorial problems by Decomposition
  - Good news: We can always decompose
    - into known/easy/small subproblems
  - Bad news: Solving each subproblem is not enough
Constraint Programming

- Solving hard combinatorial problems by Decomposition
  - Good news: We can always decompose
    - into known/easy/small subproblems
  - Bad news: Solving each subproblem is not enough

- Constraint Programming:
Constraint Programming

- Solving hard combinatorial problems by Decomposition
  - Good news: We can always decompose
    - into known/easy/small subproblems
  - Bad news: Solving each subproblem is not enough

- Constraint Progamming:
  - Constraint \( \Leftrightarrow \) “easy” Subproblem
**Constraint Programming**

- Solving hard combinatorial problems by **Decomposition**
  - **Good news**: We can **always** decompose
  - **⋆** into known/easy/small subproblems
  - **Bad news**: Solving each subproblem is not enough

- Constraint Programming:
  - **Constraint ⇔ “easy” Subproblem**
  - **Propagation** to link the reasoning done on each constraint
Solving hard combinatorial problems by Decomposition

- Good news: We can always decompose into known/easy/small subproblems
- Bad news: Solving each subproblem is not enough

Constraint Programming:

- Constraint $\Leftrightarrow$ “easy” Subproblem
- Propagation to link the reasoning done on each constraint
Constraint Satisfaction Problem
Constraint Satisfaction Problem

- Variables: $x_1, \ldots, x_n$
Constraint Satisfaction Problem

- Variables: $x_1, \ldots, x_n$
- Constraints: $C_1, \ldots, C_k$
  - Relations $=$ set of solutions
Constraint Satisfaction Problem

- Variables: $x_1, \ldots, x_n$
- Constraints: $C_1, \ldots, C_k$
  - Relations $\equiv$ set of solutions
- Domains: $D_1, \ldots, D_n$
  - Domain representation $\simeq$ relaxation of the problem’s solutions
    - $D_1 \times D_2 \times \ldots \times D_n$ is a super set of the solutions
Domain Representation (discrete domains)

4 Letters Palindrome City Problem

- Con. 1: It is a city
- Con. 2: It is a palindrome
- Domain: any 2 letters (676 sol.)
Domain Representation (discrete domains)

- 4 Letters Palindrome City Problem
  - Con. 1: It is a city
  - Con. 2: It is a palindrome
  - Domain: any 2 letters (676 sol.)

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
<th>$L_2$</th>
<th>$L_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>K</td>
<td>K</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>I</td>
<td>L</td>
<td>L</td>
<td>I</td>
</tr>
<tr>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>O</td>
<td>T</td>
<td>T</td>
<td>O</td>
</tr>
</tbody>
</table>
Domain Representation (discrete domains)

- 4 Letters Palindrome City Problem
  - Con. 1: It is a city
  - Con. 2: It is a palindrome
  - Domain: any 2 letters (676 sol.)

$L_1$ | $L_2$ | $L_2$ | $L_1$
---|---|---|---
A   | K   | K   | A   
A   | N   | N   | A   
I   | L   | L   | I   
M   | A   | A   | M   
O   | T   | T   | O   

{$A, I, M, O$}  {$A, K, L, N, T$}
### Domain Representation (discrete domains)

- **4 Letters Palindrome City Problem**
  - **Con. 1:** It is a city
  - **Con. 2:** It is a palindrome
  - **Domain:** any 2 letters (676 sol.)

<table>
<thead>
<tr>
<th>$L_1$</th>
<th>$L_2$</th>
<th>$L_2$</th>
<th>$L_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>K</td>
<td>K</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>I</td>
<td>L</td>
<td>L</td>
<td>I</td>
</tr>
<tr>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>O</td>
<td>T</td>
<td>T</td>
<td>O</td>
</tr>
</tbody>
</table>

- **Minimal Relaxation** (discrete domain)

  - $\{A,I,M,O\}$
  - $\{A,K,L,N,T\}$

  - $\{A,A,A,A\}$
  - $\{A,K,K,A\}$
  - $\{A,L,L,A\}$
  - $\{A,N,N,A\}$
  - $\{A,T,T,A\}$
  - $\{I,A,A,I\}$
  - $\{I,K,K,I\}$
  - $\{I,L,L,I\}$
  - $\{I,N,N,I\}$
  - $\{I,T,T,I\}$
  - $\{M,A,A,M\}$
  - $\{M,K,K,M\}$
  - $\{M,L,L,M\}$
  - $\{M,N,N,M\}$
  - $\{M,T,T,M\}$
  - $\{O,A,A,O\}$
  - $\{O,K,K,O\}$
  - $\{O,L,L,O\}$
  - $\{O,N,N,O\}$
  - $\{O,T,T,O\}$
Domain Representation (bounds)

- 4 Letters Palindrome City Problem
  - Con. 1: It is a city
  - Con. 2: It is a palindrome
  - Domain: any 2 letters (676 sol.)

<table>
<thead>
<tr>
<th>L₁</th>
<th>L₂</th>
<th>L₂</th>
<th>L₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>K</td>
<td>K</td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>N</td>
<td>N</td>
<td>A</td>
</tr>
<tr>
<td>I</td>
<td>L</td>
<td>L</td>
<td>I</td>
</tr>
<tr>
<td>M</td>
<td>A</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>O</td>
<td>T</td>
<td>T</td>
<td>O</td>
</tr>
</tbody>
</table>

[A, . . . , O]  [A, . . . , T]

- Minimal Relaxation (discrete bounds)
Constraint Propagation

• Given a constraint $C$ and a domain representation $D$
Constraint Propagation

- Given a constraint $C$ and a domain representation $D$
  - Compute the intersection $C \cap D$

Consistency: a domain is consistent iff it is a minimal relaxation $D$

- Discrete domain: Arc Consistency (ac)
- Bounds: Bounds Consistency (bc)
- Also:
  - Multi-valued Decision Diagram MDD Consistency [Hooker, Hadzic, van Hoeve 2007]
  - Set variables [Puget 1992, Gervet 1997]
  - Length-Lex representation [Gervet and Van Hentenryck 2006]
  - Graph variables, Function variables, ...
Constraint Propagation

Given a constraint $C$ and a domain representation $D$

- Compute the intersection $C \cap D$
- Find the minimal relaxation $D'$ of $C \cap D$
  - Find all solutions of $C \cap D$ (support)
  - Project on the domain representation (filtering)
Constraint Propagation

- Given a constraint $C$ and a domain representation $D$
  - Compute the intersection $C \cap D$
  - Find the minimal relaxation $D'$ of $C \cap D$
    - Find all solutions of $C \cap D$ (support)
    - Project on the domain representation (filtering)

- **Consistency**: a domain is consistent iff it is a minimal relaxation
Constraint Propagation

- Given a constraint $C$ and a domain representation $D$
  - Compute the intersection $C \cap D$
  - Find the minimal relaxation $D'$ of $C \cap D$
    - Find all solutions of $C \cap D$ (support)
    - Project on the domain representation (filtering)

- Consistency: a domain is consistent iff it is a minimal relaxation
  - Discrete domain: **Arc Consistency** (ac)
  - Bounds: **Bounds Consistency** (bc)
Constraint Propagation

- Given a constraint $C$ and a domain representation $D$
  - Compute the intersection $C \cap D$
  - Find the minimal relaxation $D'$ of $C \cap D$
    - Find all solutions of $C \cap D$ (support)
    - Project on the domain representation (filtering)

- **Consistency**: a domain is consistent iff it is a minimal relaxation
  - **Discrete domain**: *Arc Consistency (ac)*
  - **Bounds**: *Bounds Consistency (bc)*
  - Also:
    - Multi-valued Decision Diagram *MDD Consistency* [Hooker, Hadzic, van Hoeve 2007]
    - Graph variables, Function variables, ...
Example: Kakuro

\[
\begin{array}{cccccc}
23 & 30 & 27 & 12 & 16 \\
16 & 39 & 24 & 17 \\
17 & 35 & 7 & 8 & 12 \\
11 & 10 & 16 & 7 & 7 \\
21 & 5 & 5 & 4 & 4 \\
6 & 3 & 3 & 3 & 3 \\
\end{array}
\]
Example: Kakuro

- \( \sum_{i=1}^{6} x_i = 39 \)
- \( \text{ALLDIFFERENT}\left(\{x_1, \ldots, x_6\}\right) \)

\[
\begin{align*}
  x_1 : & \quad \{8, 9\} \\
  x_2 : & \quad \{1, 2, 6, 7, 8, 9\} \\
  x_3 : & \quad \{8, 9\} \\
  x_4 : & \quad \{1, 5, 6, 8, 9\} \\
  x_5 : & \quad \{1, 2, 6, 7, 8, 9\} \\
  x_6 : & \quad \{4, 5, 8, 9\}
\end{align*}
\]
Example: Kakuro

\[ \sum_{i=1}^{6} x_i = 39 \]

\textbf{ALLDIFFERENT}\(\{x_1, \ldots, x_6\}\)

\begin{align*}
x_1 & : \quad \{8, 9\} \\
x_2 & : \quad \{1, 2, 6, 7, 8, 9\} \\
x_3 & : \quad \{8, 9\} \\
x_4 & : \quad \{1, 5, 6, 8, 9\} \\
x_5 & : \quad \{1, 2, 6, 7, 8, 9\} \\
x_6 & : \quad \{4, 5, 8, 9\}
\end{align*}

\textbf{ALLDIFFERENT}\(\{x_1, \ldots, x_6\}\) \(\cap\) \(D\)
Example: Kakuro

\[ \sum_{i=1}^{6} x_i = 39 \]

\[ \text{ALLDIFFERENT}(\{x_1, \ldots, x_6\}) \]

\[ x_1 : \{8, 9\} \]
\[ x_2 : \{1, 2, 6, 7, 8\} \]
\[ x_3 : \{8, 9\} \]
\[ x_4 : \{1, 5, 6\} \]
\[ x_5 : \{1, 2, 6, 7\} \]
\[ x_6 : \{4, 5\} \]
Example: Kakuro

\[ \sum_{i=1}^{6} x_i = 39 \]

\textbf{ALLDIFFERENT}\(\{x_1, \ldots, x_6\}\)

\begin{align*}
\forall i, j & : x_i \neq x_j, \quad i \neq j \\
& \text{for } i = 1, \ldots, 6 \\
\end{align*}

\begin{align*}
x_1 & : \quad \{8, 9\} \\
x_2 & : \quad \{1, 2, 6, 7, 8\} \\
x_3 & : \quad \{8, 9\} \\
x_4 & : \quad \{1, 5, 6\} \\
x_5 & : \quad \{1, 2, 6, 7\} \\
x_6 & : \quad \{4, 5\} \\
\end{align*}

\[ \sum_{i=1}^{6} x_i = 39 \cap D \]

\begin{align*}
8 & 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 \\
6 & 6 6 6 6 6 6 7 7 7 7 6 6 6 6 6 7 \\
8 & 8 8 8 9 9 9 8 8 8 8 9 8 8 8 9 8 \\
5 & 6 6 5 5 6 5 5 5 6 5 5 5 6 5 5 5 \\
7 & 6 7 6 7 6 6 7 6 6 6 7 6 6 6 6 6 \\
5 & 5 4 5 4 4 5 4 4 4 5 4 4 4 4 4 4 \\
\end{align*}
Example: Kakuro

- $\sum_{i=1}^{6} x_i = 39$
- \(\text{ALLDIFFERENT}\{x_1, \ldots, x_6\}\)

\[
\begin{align*}
x_1 : & \quad \{8, 9\} \\
x_2 : & \quad \{6, 7, 8\} \\
x_3 : & \quad \{8, 9\} \\
x_4 : & \quad \{5, 6\} \\
x_5 : & \quad \{6, 7\} \\
x_6 : & \quad \{4, 5\}
\end{align*}
\]

$\sum_{i=1}^{6} x_i = 39 \cap D$

\[
\begin{align*}
8 & \quad 8 \quad 8 \quad 8 \quad 8 \quad 8 \quad 8 \quad 8 \quad 8 \quad 9 \quad 9 \quad 9 \quad 9 \quad 9 \quad 9 \quad \{8, 9\} \\
6 & \quad 6 \quad 6 \quad 6 \quad 6 \quad 6 \quad 7 \quad 7 \quad 7 \quad 7 \quad 6 \quad 6 \quad 6 \quad 6 \quad 6 \quad \{6, 7\} \\
8 & \quad 8 \quad 8 \quad 9 \quad 9 \quad 9 \quad 8 \quad 8 \quad 8 \quad 9 \quad 8 \quad 8 \quad 8 \quad 9 \quad 8 \quad \{8, 9\} \\
5 & \quad 6 \quad 6 \quad 5 \quad 5 \quad 6 \quad 5 \quad 5 \quad 5 \quad 5 \quad 6 \quad 5 \quad 5 \quad 5 \quad \{5, 6\} \\
7 & \quad 6 \quad 7 \quad 6 \quad 7 \quad 6 \quad 6 \quad 7 \quad 6 \quad 6 \quad 6 \quad 7 \quad 6 \quad 6 \quad 6 \quad \{6, 7\} \\
5 & \quad 5 \quad 4 \quad 5 \quad 4 \quad 5 \quad 4 \quad 4 \quad 4 \quad 5 \quad 4 \quad 4 \quad 4 \quad 4 \quad \{4, 5\}
\end{align*}
\]
Example: Kakuro

\[ \sum_{i=1}^{6} x_i = 39 \]

\textsc{AllDifferent}\(\{x_1, \ldots, x_6\}\)

\begin{align*}
x_1: & \quad \{8, 9\} \\
x_2: & \quad \{6, 7, 8\} \\
x_3: & \quad \{8, 9\} \\
x_4: & \quad \{5, 6\} \\
x_5: & \quad \{6, 7\} \\
x_6: & \quad \{4, 5\} \\
\end{align*}

\textsc{AllDifferent}\(\{x_1, \ldots, x_6\}\) \(\cap\) \(D\):

\begin{align*}
8 & \quad 8 & \quad 9 & \quad 9 & \quad \{8, 9\} \\
6 & \quad 7 & \quad 6 & \quad 7 & \quad \{6, 7\} \\
9 & \quad 9 & \quad 8 & \quad 8 & \quad \{8, 9\} \\
5 & \quad 5 & \quad 5 & \quad 5 & \quad \{5\} \\
7 & \quad 6 & \quad 7 & \quad 6 & \quad \{6, 7\} \\
4 & \quad 4 & \quad 4 & \quad 4 & \quad \{4\} \\
\end{align*}
Example: Kakuro

- \( \sum_{i=1}^{6} x_i = 39 \)
- \( \text{ALLDIFFERENT} \{x_1, \ldots, x_6\} \)

\[
\begin{align*}
x_1 : & \quad \{ \quad 8 \quad 9 \} \\
x_2 : & \quad \{ \quad 6 \quad 7 \quad 8 \} \\
x_3 : & \quad \{ \quad 8 \quad 9 \} \\
x_4 : & \quad \{ \quad 5 \} \\
x_5 : & \quad \{ \quad 6 \quad 7 \} \\
x_6 : & \quad \{ \quad 4 \}
\end{align*}
\]

\( \text{ALLDIFFERENT} \{x_1, \ldots, x_6\} \cap D \)

\[
\begin{array}{cccccc}
8 & 8 & 9 & 9 \\
6 & 7 & 6 & 7 \\
9 & 9 & 8 & 8 \\
5 & 5 & 5 & 5 \\
7 & 6 & 7 & 6 \\
4 & 4 & 4 & 4 \\
\end{array}
\]

\[
\begin{array}{c}
\{8, 9\} \\
\{6, 7\} \\
\{8, 9\} \\
\{5\} \\
\{6, 7\} \\
\{4\}
\end{array}
\]
Of course we do not want to enumerate solutions!
Constraint Propagation

- Of course we do not want to enumerate solutions!
  - Propagating is not necessarily harder than solving
    - AC on \texttt{ALLDIFFERENT} in $O(n^{1.5}d)$, same as Matching
Of course we do not want to enumerate solutions!

- Propagating is not necessarily harder than solving
  - AC on ALLDIFFERENT in $O(n^{1.5} d)$, same as Matching
- In general:
  - Solving in $O(K)$, then propagating in $O(ndK)$
  - Propagating in $O(K)$, then solving in $O(nK)$
Of course we do not want to enumerate solutions!

- Propagating is not necessarily harder than solving
  - AC on AllDifferent in $O(n^{1.5}d)$, same as Matching

- In general:
  - Solving in $O(K)$, then propagating in $O(ndK)$
  - Propagating in $O(K)$, then solving in $O(nK)$

- In practice:
  - Same complexity class, but different algorithms
  - Propagating is harder
An Example of Propagator: the \textsc{Switch} Constraint
A set of $n$ garments to embroid
  - Each garment require a subset of $m$ available colors
A machine with $k$ reels of thread
A set of $n$ garments to embroid

- Each garment requires a subset of $m$ available colors

A machine with $k$ reels of thread

Replacing a reel costs time
A set of $n$ garments to embroid

- Each garment require a subset of $m$ available colors

A machine with $k$ reels of thread

Replacing a reel costs time

What is the best sequence of garments?
Embroidery Scheduling

A
B
C
D
E
F
G

1-st  2-nd  3-rd  4-th  5-th  6-th  7-th

Reel 1

Reel 2

Reel 3
Embroidery Scheduling

Reel 1

Reel 2

Reel 3
Embroidery Scheduling

1-st  2-nd  3-rd  4-th  5-th  6-th  7-th
A     B     C     D     E     F     G

A, B, C, D, E, F, G: 9 switches
B, F, A, E, D, C, G: 5 switches
Embroidery Scheduling

A - B - C - D - E - F - G

Switches:
- A, B, C, D, E, F, G: 9 switches
- B, F, A, E, D, C, G: 5 switches
Embroidery Scheduling

A  B  C  D  E  F  G

1-st  2-nd  3-rd  4-th  5-th  6-th  7-th
A  B  C  D  E  F  G

Reel 1

Reel 2

Reel 3

A,B,C,D,E,F,G: 9 switches
B,F,A,E,D,C,G: 5 switches
**Embroidery Scheduling**

- **Reel 1**
  - 1-st: switch
  - 2-nd: switch
  - 3-rd: switch
  - 4-th: switch
  - 5-th: switch
  - 6-th: switch
  - 7-th: switch

- **Reel 2**
  - 1-st: switch
  - 2-nd: switch
  - 3-rd: switch
  - 4-th: switch
  - 5-th: switch
  - 6-th: switch
  - 7-th: switch

- **Reel 3**
  - 1-st: switch
  - 2-nd: switch
  - 3-rd: switch
  - 4-th: switch
  - 5-th: switch
  - 6-th: switch
  - 7-th: switch

- **A, B, C, D, E, F, G: 9 switches**
Embroidery Scheduling

- A, B, C, D, E, F, G: 9 switches
Embroidery Scheduling

A, B, C, D, E, F, G: 9 switches
Embroidery Scheduling

A, B, C, D, E, F, G: 9 switches
Embroidery Scheduling

A, B, C, D, E, F, G: 9 switches
B, F, A, E, D, C, G: 5 switches
The Switch Constraint

One Boolean variable per color \( j \) and position \( i \):

\[ x_{ji} = 1 \text{ iff the } j\text{-th color is on the buffer at time } i \]

A variable \( M \) equal to the number of changes:

\[ x_{ji} < x_{ji} + 1 \]

Not only useful for embroidery:

- Instruction scheduling (compilation)
- Test sequencing
- File transfer (observation satellites)

Switch turned up in 3/4 of the industrial projects I was involved in!
The **Switch Constraint**

- One Boolean variable per **color** $j$ and **position** $i$
  - $x^j_i = 1$ iff the $j$-th color is on the buffer at time $i$
The **Switch** Constraint

- One Boolean variable per **color** \( j \) and **position** \( i \)
  - \( x^j_i = 1 \) iff the \( j \)-th color is on the buffer at time \( i \)
- A variable \( M \) equal to the number of changes \( x^j_i < x^j_{i+1} \)
The **Switch Constraint**

- One Boolean variable per color $j$ and position $i$
  - $x^j_i = 1$ iff the $j$-th color is on the buffer at time $i$

- A variable $M$ equal to the number of changes $x^j_i < x^j_{i+1}$

- Not only useful for embroidery
  - Instruction scheduling (compilation)
  - Test sequencing
  - File transfer (observation satellites)
The **Switch** Constraint

- One Boolean variable per color \( j \) and position \( i \)
  
  \[ x^j_i = 1 \text{ iff the } j\text{-th color is on the buffer at time } i \]

- A variable \( M \) equal to the number of changes \( x^j_i < x^j_{i+1} \)

- Not only useful for embroidery
  
  - Instruction scheduling (compilation)
  - Test sequencing
  - File transfer (observation satellites)

- **Switch** turned up in 3/4 of the industrial projects I was involved in!
Propagating the Switch Constraint

1-st  2-nd  3-rd  4-th  5-th  6-th  7-th

Reel 1

Reel 2

Reel 3
Propagating the Switch Constraint

1-st \{B, F\}  2-nd \{A\}  3-rd \{B, F\}  4-th \{C, D, E\}  5-th \{C, D, E\}  6-th \{G\}  7-th \{C, D, E\}
Propagating the \textbf{Switch} Constraint

\begin{itemize}
  \item Reel 1: \{B, F\}
  \item Reel 2: \{A\}
  \item Reel 3: \{B, F\}
\end{itemize}

1-st: \{B, F\}
2-nd: \{A\}
3-rd: \{B, F\}
4-th: \{C, D, E\}
5-th: \{C, D, E\}
6-th: \{G\}
7-th: \{C, D, E\}

How many switches at least? (compute a lower bound for $M$)

What are the possible colors (given the upper bound on $M$)
Propagating the **Switch** Constraint

![Diagram showing switches and colors for Reels 1, 2, and 3]

<table>
<thead>
<tr>
<th>Reel 1</th>
<th>Reel 2</th>
<th>Reel 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-st</td>
<td>2-nd</td>
<td>3-rd</td>
</tr>
<tr>
<td>{B, F}</td>
<td>{A}</td>
<td>{B,F}</td>
</tr>
<tr>
<td>4-th</td>
<td>5-th</td>
<td>6-th</td>
</tr>
<tr>
<td>{C, D, E}</td>
<td>{C, D, E}</td>
<td>{G}</td>
</tr>
<tr>
<td>7-th</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{C, D, E}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How many switches at least? (compute a lower bound for $M$)

What are the possible colors (given the upper bound on $M$)?
Propagating the Switch Constraint

How many switches at least? (compute a lower bound for $M$)
What are the possible colors (given the upper bound on $M$)
Propagating the \textbf{Switch Constraint}
The **Switch Constraint**

**capacity of the buffer**

An edge for each color and each position
The **Switch Constraint**

The capacity of the buffer for each color and each position is shown in the diagram. Each edge represents a constraint between the source (S) and the target (T) nodes.
The Switch Constraint
The **Switch Constraint**

An edge for each color and each position
The Switch Constraint

Capacity of the buffer

An edge for each color and each position
The Switch Constraint
The **Switch Constraint**

![Diagram of the Switch Constraint]

1-st | 2-nd
--- | ---

1. **Switch Constraint**
2. **1-st**
3. **2-nd**
4. **S**
5. **C₁**

---

17 / 54
The **Switch Constraint**

![Diagram of the Switch Constraint](image-url)
The **Switch Constraint**
The **Switch** Constraint
The Switch Constraint
The Switch Constraint
The Switch Constraint
The **Switch** Constraint

1-st | 2-nd | 3-rd | 4-th | 5-th | 6-th | 7-th
--- | --- | --- | --- | --- | --- | ---

\[
\begin{array}{cccccccc}
\text{S} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{T} \\
\text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\text{1} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} & \text{} \\
\end{array}
\]

\[
c_3 \quad \text{OK} \quad \text{OK} \quad c_5
\]
The **Switch** Constraint
Propagation Algorithm (details at CP-AI-OR)
Propagation Algorithm (details at CP-AI-OR)

- Find a maximum flow of minimum cost: $O(nd)$
Propagation Algorithm (details at CP-AI-OR)

- Find a maximum flow of minimum cost: $O(nd)$
- Eliminate negative costs of the residual graph: $O((nd)^{1.5})$
Propagation Algorithm (details at CP-AI-OR)

- Find a maximum flow of minimum cost: $O(nd)$
- Eliminate negative costs of the residual graph: $O((nd)^{1.5})$
- Find all cycles of costs 0 and 1 (but not 2 or greater): $O(n^2d)$
Propagation Algorithm (details at CP-AI-OR)

- Find a maximum flow of minimum cost: $O(nd)$
- Eliminate negative costs of the residual graph: $O((nd)^{1.5})$
- Find all cycles of costs 0 and 1 (but not 2 or greater): $O(n^2d)$

**Total time complexity:** $O(n^2d + nd^{1.5})$
Complexity of Propagating vs. Complexity of Solving

- Propagating is often harder
  - E.g. **Switch**: solving in $O(nd)$, propagating in $O(n^2d + nd^{1.5})$
Complexity of Propagating vs. Complexity of Solving

- Propagating is often harder
  - E.g. **Switch**: solving in $O(nd)$, propagating in $O(n^2d + nd^{1.5})$
- Sometimes, we can propagate “for free”, i.e., with the same time complexity as for solving
Propagating for “Free”: the \texttt{AtMostSeqCard} Constraint
The **AtMostSeqCard Constraint**

- A **sequence** of variables \([x_1, \ldots, x_n]\)
  - \(x_i = 1\) iff the bolt is of the type that Charlot cares about

**Definition:** \(\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n]) \iff\)
The **AtMostSeqCard** Constraint

- A sequence of variables \([x_1, \ldots, x_n]\)
  - \(x_i = 1\) iff the bolt is of the type that Charlot cares about
- There is a given demand for bolts of this type

**Definition:** \(\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n]) \iff \left( \sum_{i=1}^{n} x_i = d \right) \land \left( n - q \bigwedge_{i=0}^{q} \left( \sum_{l=1}^{q} x_{i+l} \leq u \right) \right)\)
The **AtMostSeqCard** Constraint

- **A sequence** of variables $[x_1, \ldots, x_n]$
  - $x_i = 1$ iff the bolt is of the type that Charlot cares about
- **There is a given demand** for bolts of this type
- **They should not occur alltogether (capacity constraints)**
  - On each subsequence of size $q$, at most $u$ are of this type

**Definition:** \[ \text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n]) \iff \]

\[
\left( \sum_{i=1}^{n} x_i = d \right) \land \bigwedge_{i=0}^{n-q} \left( \sum_{l=1}^{q} x_{i+l} \leq u \right)
\]
The **AtMostSeqCard** Constraint

- A sequence of variables \([x_1, \ldots, x_n]\)
  - \(x_i = 1\) iff the bolt is of the type that Charlot cares about
- There is a given **demand** for bolts of this type
- They should not occur altogether (**capacity** constraints)
  - On each subsequence of size \(q\), at most \(u\) are of this type
- Example: \(u = 2, q = 4, d = 10, [x_1, \ldots, x_{22}]\)

**Definition:** \(\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n]) \iff\)

\[
\left( \sum_{i=1}^{n} x_i = d \right) \land \left( \sum_{i=0}^{n-q} x_i + \sum_{l=1}^{q} x_{i+l} \leq u \right)
\]

\[
\begin{array}{cccccccccccccccccccc}
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1
\end{array}
\]
The **AtMostSeqCard** Constraint

- A sequence of variables $[x_1, \ldots, x_n]$
  - $x_i = 1$ iff the bolt is of the type that Charlot cares about
- There is a given demand for bolts of this type
- They should not occur altogether (capacity constraints)
  - On each subsequence of size $q$, at most $u$ are of this type
- Example: $u = 2$, $q = 4$, $d = 10$, $[x_1, \ldots, x_{22}]$

**Definition:**

$\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n]) \iff$

$\left( \sum_{i=1}^{n} x_i = d \right) \land \left( \bigwedge_{i=0}^{n-q} \left( \sum_{l=1}^{q} x_{i+l} \leq u \right) \right)$

```
1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1
```
The `AtMostSeqCard` Constraint

- A sequence of variables \([x_1, \ldots, x_n]\)
  - \(x_i = 1\) iff the bolt is of the type that Charlot cares about
- There is a given demand for bolts of this type
- They should not occur altogether (capacity constraints)
  - On each subsequence of size \(q\), at most \(u\) are of this type
- Example: \(u = 2, q = 4, d = 10, [x_1, \ldots, x_{22}]\)

**Definition:**  
`AtMostSeqCard(u, q, d, [x_1, \ldots, x_n]) ⇐⇒`  

\[
\left( \sum_{i=1}^{n} x_i = d \right) \land \bigwedge_{i=0}^{n-q} \left( \sum_{l=1}^{q} x_{i+l} \leq u \right)
\]

1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1
The **AtMostSeqCard** Constraint

- A sequence of variables \([x_1, \ldots, x_n]\)
  - \(x_i = 1\) iff the bolt is of the type that Charlot cares about
- There is a given **demand** for bolts of this type
- They should not occur altogether (capacity constraints)
  - On each subsequence of size \(q\), at most \(u\) are of this type
- Example: \(u = 2, q = 4, d = 10, [x_1, \ldots, x_{22}]\)

**Definition:** \(\text{AtMostSeqCard}(u, q, d, [x_1, \ldots, x_n]) \iff \bigwedge_{i=0}^{n-q} \left( \sum_{l=1}^{q} x_{i+l} \leq u \right) \land \left( \sum_{i=1}^{n} x_i = d \right)\)

1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1
Solving the \textbf{AtMostSeqCard} Constraint

- Finding a support (solving)
Solving the AtMostSeqCard Constraint

- Finding a support (solving)
  - We can be greedy and assign 1 whenever possible ⇒ leftmost
Example: $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4)$

$$D(x_i) \cdot 0.1.0.0.0.1.1.1.1.1.1.1$$
Example: \( \overrightarrow{w} = \text{leftmost } (u = 2, q = 4) \)

\[
\begin{array}{c}
D(x_i) \quad 0 \quad 1 \quad \ldots \quad 0 \quad 0 \quad 1 \quad \ldots \quad 1 \\
\overrightarrow{w}[i]
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
Example: $\overrightarrow{w} = \text{leftmost } (u = 2, \ q = 4)$

\[
\begin{array}{cccccccccc}
D(x_i) & . & 0 & . & 1 & . & . & 0 & . & 0 & 1 & . & 1 & . & . & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
Example: $\overline{w} = \text{leftmost } (u = 2, \ q = 4)$

\[
\begin{array}{cccccccc}
\mathcal{D}(x_i) & 0 & . & 1 & . & . & 0 & . & 0 & 1 & . & 1 & . & . & . & . & . & . & . & 1 \\
\overline{w}[i] & 0 & 1 & 0 & 0 & 1 \text{ [red box]} & 0 & 1 & 0 & 0 & 1 & 1 \\
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
Example: $\overrightarrow{w} = \text{leftmost (} u = 2, \ q = 4 \text{)}$

\[
\begin{array}{cccccccc}
\mathcal{D}(x_i) & . & 0 & . & 1 & . & . & 0 & . & 0 & 1 & . & . & 1 & . & . & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
Example: \( \overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4) \)

\[
\begin{array}{cccccccccccc}
D(x_i) & . & 0 & . & 1 & . & . & . & 0 & . & 0 & 1 & . & 1 & . & . & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & & & & & 0 & & 0 & 1 & 0 & 0 & 1 & & & & 1
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
Example: $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4)$

\[
\begin{array}{cccccccc}
D(x_i) & . & 0 & . & 1 & . & . & 0 & . & 0 & 1 & . & . & 1 & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & & & & & & 1 \\
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
Example: $\overrightarrow{w} = \text{leftmost } (u = 2, \ q = 4)$

\[
\begin{array}{ccccccccccc}
\mathcal{D}(x_i) & . & 0 & . & 1 & . & . & . & 0 & . & 0 & 1 & . & . & . & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value $1$ to $0$ has no impact on capacity constraints
Example: $\overrightarrow{w} =$ leftmost ($u = 2$, $q = 4$)

\[\begin{array}{cccccccccccccccc}
D(x_i) & . & 0 & . & 1 & . & . & . & 0 & . & 0 & 1 & . & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1
\end{array}\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
Example: \( \overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4) \)

\[ \begin{array}{cccccccccccccc}
D(x_i) & . & 0 & . & 1 & . & . & 0 & . & 0 & 1 & . & . & 1 & . & . & . & . & . & 1 \\
\hline
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1
\end{array} \]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
  - Greedily inserting 1 works
Example: $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4)$

\[
\begin{array}{cccccccccccccccccccc}
\mathcal{D}(x_i) & . & 0 & . & 1 & . & . & . & 0 & . & 0 & 1 & . & 1 & . & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\end{array}
\]

- Support: maximizing the cardinality while respecting capacities
  - Changing the value 1 to 0 has no impact on capacity constraints
  - Greedily inserting 1 works
- We can find a support in $O(nq)$
- We can achieve $\text{AC}$ in $O(n^2q)$
Filtering the $\text{AtMostSeqCard}$ Constraint

- Can we propagate in the same time complexity?
Filtering the \texttt{AtMostSeqCard} Constraint

- Can we propagate in the same time complexity?
  - There is nothing to propagate unless $|\vec{w}| = d$
Filtering the **AtMostSeqCard** Constraint

- Can we propagate in the same time complexity?
  - There is nothing to propagate unless $|\overrightarrow{w}| = d$
  - Two runs of `leftmost` are enough
Example: $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10)$

\[
\begin{array}{c|cccccccccccccccc}
D(x_i) & 0 & . & 1 & . & . & 0 & . & 0 & 1 & . & 1 & . & . & . & . & . & . & . & 1 \\
\hline
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1
\end{array}
\]
**Example:** $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10)$

\[
\begin{array}{c|cccccccccccccccc}
D(x_i) & . & 0 & . & 1 & . & . & 0 & . & 0 & 1 & . & 1 & . & . & . & . & . & . & 1 \\
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
L[i] & 0 & 1 & 1 & 1 & 2 & 3 & 3 & 3 & 3 & 4 & 4 & 5 & 5 & 5 & 6 & 7 & 7 & 7 & 8 & 9 & 9 & 9 & d
\end{array}
\]
**Example:** $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10)$

<table>
<thead>
<tr>
<th>$D(x_i)$</th>
<th>. 0 . 1 . . . 0 . 0 1 . . 1 . . . . . . . 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overrightarrow{w}[i]$</td>
<td>1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>$\overleftarrow{w}[i]$</td>
<td>0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$L[i]$</td>
<td>0 1 1 1 2 3 3 3 3 4 4 5 5 5 6 7 7 7 8 9 9 9 d</td>
</tr>
</tbody>
</table>
**Example:** $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10)$

<table>
<thead>
<tr>
<th>$D(x_i)$</th>
<th>. 0 . 1 . . . 0 . 0 1 . . 1 . . . . . . . 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overrightarrow{w}[i]$</td>
<td>1 0 0 1 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>$\overleftarrow{w}[i]$</td>
<td>0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$L[i]$</td>
<td>0 1 1 1 2 3 3 3 3 4 4 5 5 5 6 7 7 7 8 9 9 9 $d$</td>
</tr>
<tr>
<td>$R[i]$</td>
<td>$d$ $d$ $d$ 9 8 8 8 7 7 6 6 5 5 5 4 4 4 4 3 2 2 2 1 0</td>
</tr>
</tbody>
</table>

We can achieve $ac$ in $O(nq)$, that is, $O(n^2)$.

We can do better: leftmost can run in $O(n)$.

Total complexity of $O(n)$, optimal!
Example: $\vec{w} = \text{leftmost } (u = 2, \ q = 4, \ d = 10)$

<table>
<thead>
<tr>
<th>$D(x_i)$</th>
<th>. 0 . 1 . . . 0 . 0 1 . . 1 . . . . . . . 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vec{w}[i]$</td>
<td>1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>$\vec{w}[i]$</td>
<td>0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$L[i]$</td>
<td>0 1 1 1 2 3 3 3 3 4 4 5 5 5 6 7 7 7 8 9 9 9 $d$</td>
</tr>
<tr>
<td>$R[i]$</td>
<td>$d$ $d$ $d$ 9 8 8 8 7 7 6 6 5 5 5 4 4 4 3 2 2 2 1 0</td>
</tr>
</tbody>
</table>

We can achieve $ac$ in $O(nq)$, that is, $O(n^2)$.

We can do better: leftmost can run in $O(n)$, details.

Total complexity of $O(n)$, optimal!
Example: $\vec{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10)$

<table>
<thead>
<tr>
<th>$D(x_i)$</th>
<th>. 0 . 1 . . . 0 . 0 1 . . 1 . . . . . . . . . 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\vec{w}[i]$</td>
<td>1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>$\vec{w}[i]$</td>
<td>0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$L[i]$</td>
<td>0 1 1 1 2 3 3 3 3 4 4 5 5 5 6 7 7 7 8 9 9 9 d</td>
</tr>
<tr>
<td>$R[i]$</td>
<td>d d d 9 8 8 8 7 7 6 6 5 5 5 4 4 4 3 2 2 2 1 0</td>
</tr>
</tbody>
</table>

We can achieve ac in $O(nq)$, that is, $O(n^2)$.

We can do better: leftmost can run in $O(n)$, optimal!
Example: $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10)$

\[
\begin{array}{c|cccccccccccccc}
D(x_i) & . & 0 & 1 & . & . & . & 0 & 1 & 0 & 1 & . & 1 & . & . & . & . & . & 1 \\
\hline
\overrightarrow{w}[i] & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\
\overleftarrow{w}[i] & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
L[i] & 0 & 1 & 1 & 1 & 2 & 3 & 3 & 3 & 3 & 4 & 4 & 5 & 5 & 5 & 6 & 7 & 7 & 7 & 8 & 9 & 9 & 9 & d \\
R[i] & d & d & d & 9 & 8 & 8 & 8 & 7 & 7 & 6 & 6 & 5 & 5 & 5 & 4 & 4 & 4 & 3 & 2 & 2 & 2 & 1 & 0 \\
\end{array}
\]
Example: $\overrightarrow{w} = \text{leftmost } (u = 2, q = 4, d = 10)$

<table>
<thead>
<tr>
<th>$D(x_i)$</th>
<th>. 0 . 1 . . 0 1 0 1 . . 1 . . 1 . . 1 . . 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overrightarrow{w}[i]$</td>
<td>1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>$\overleftarrow{w}[i]$</td>
<td>0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$L[i]$</td>
<td>0 1 1 1 2 3 3 3 4 4 5 5 5 6 7 7 7 8 9 9 9 d</td>
</tr>
<tr>
<td>$R[i]$</td>
<td>d d d 9 8 8 8 7 7 6 6 5 5 5 4 4 4 3 2 2 2 1 0</td>
</tr>
</tbody>
</table>
Example: $\overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10)$

<table>
<thead>
<tr>
<th>$D(x_i)$</th>
<th>.0 1 . . . . 0 1 0 1 . . 1 . . 1 . . 1 .</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\overrightarrow{w}[i]$</td>
<td>1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>$\overleftarrow{w}[i]$</td>
<td>0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>$L[i]$</td>
<td>0 1 1 1 2 3 3 3 3 4 4 5 5 5 6 7 7 7 8 9 9 9 d</td>
</tr>
<tr>
<td>$R[i]$</td>
<td>d d d 9 8 8 8 7 7 6 6 5 5 5 4 4 4 3 2 2 2 1 0</td>
</tr>
</tbody>
</table>

- We can achieve $AC$ in $O(nq)$, that is, $O(n^2)$
**Example:** \( \overrightarrow{w} = \text{leftmost} \ (u = 2, \ q = 4, \ d = 10) \)

<table>
<thead>
<tr>
<th>( D(x_i) )</th>
<th>. 0 . 1 . . . 0 1 0 1 . . 1 . . 1 . . 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \overrightarrow{w}[i] )</td>
<td>1 0 0 1 1 0 0 0 1 0 1 0 0 1 1 0 0 1 1 0 0 1</td>
</tr>
<tr>
<td>( \overleftarrow{w}[i] )</td>
<td>0 0 1 1 0 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>( L[i] )</td>
<td>0 1 1 1 2 3 3 3 4 4 5 5 5 6 7 7 7 8 9 9 9 d</td>
</tr>
<tr>
<td>( R[i] )</td>
<td>d d d 9 8 8 8 7 7 6 6 5 5 5 4 4 4 3 2 2 2 1 0</td>
</tr>
</tbody>
</table>

- We can achieve AC in \( O(nq) \), that is, \( O(n^2) \)
- We can do better: **leftmost** can run in \( O(n) \) **details**
- Total complexity of \( O(n) \), optimal!
NP-hard Constraints

- If solving is NP-hard, then achieving AC is NP-hard
NP-hard Constraints

- If solving is **NP-hard**, then achieving $\Delta$ is **NP-hard**
- Should we *decompose* into simpler constraints?
NP-hard Constraints: the SoftAllEqual Constraints
The **AllDifferent Constraint**

- Each variable should take a distinct value (**Matching**)
  - Propagation algorithm [Régin 1994, Costa 1994]
The **AllDifferent** Constraint

- Each variable should take a distinct value (**Matching**)
  - Propagation algorithm [Régin 1994, Costa 1994]
- What about an “**AllEqual**” constraint?
The **AllDifferent** Constraint

- Each variable should take a distinct value (Matching)
  - Propagation algorithm [Régis 1994, Costa 1994]
- What about an “**AllEqual**” constraint?
  - Trivial to solve and propagate
The **AllDifferent** Constraint

- Each variable should take a distinct value (Matching)
  - Propagation algorithm [Régis 1994, Costa 1994]
- What about an "**AllEqual**" constraint?
  - Trivial to solve and propagate
  - However, the *soft* version is far from trivial
The \textbf{AllDifferent} Constraint

- Each variable should take a distinct value (\textbf{Matching})
  - Propagation algorithm [Régis 1994, Costa 1994]
- What about an \textbf{ALLEQUAL} constraint?
  - Trivial to solve and propagate
  - However, the soft version is far from trivial
  - Try to make the variables \textit{as equal as possible}
The **AllDifferent** Constraint

- Each variable should take a distinct value (Matching)
  - Propagation algorithm [Régin 1994, Costa 1994]
- What about an “**AllEqual**” constraint?
  - Trivial to solve and propagate
  - However, the soft version is far from trivial
  - Try to make the variables as equal as possible

- Two versions
  - Maximise the number of equalities
  - Minimise the number of value
The **SoftAllEqual Constraint**

**Definition:** \( \text{SoftAllEqual}(k, [x_1, \ldots, x_n]) \iff \)

At least \( k \) pairs of equal variables in the set \( \{x_1, \ldots, x_n\} \)
The SoftAllEqual Constraint

**Definition:** \(\text{SoftAllEqual}(k, [x_1, \ldots, x_n]) \iff\)

At least \(k\) pairs of equal variables in the set \([x_1, \ldots, x_n]\)

- Meeting Scheduling
The \textbf{SoftAllEqual} Constraint

\textbf{Definition:} \texttt{SoftAllEqual}(k, [x_1, \ldots, x_n]) \iff

At least \( k \) pairs of equal variables in the set \{x_1, \ldots, x_n\}

- Meeting Scheduling
  - Each researcher states his/her availabilities
The **SoftAllEqual** Constraint

**Definition:** \( \text{SoftAllEqual}(k, [x_1, \ldots, x_n]) \Leftrightarrow \)

At least \( k \) pairs of equal variables in the set \( \{x_1, \ldots, x_n\} \)

- Meeting Scheduling
  - Each researcher states his/her availabilities
  - Each researcher goes to at most one meeting
**The SoftAllEqual Constraint**

**Definition:** \( \text{SoftAllEqual}(k, [x_1, \ldots, x_n]) \iff \)

At least \( k \) pairs of equal variables in the set \( \{x_1, \ldots, x_n\} \)

- Meeting Scheduling
  - Each researcher states his/her availabilities
  - Each researcher goes to at most one meeting
  - Schedule meetings so that:
    - The number of interactions is maximum
The **SoftAllEqual** Constraint

**Definition:** \( \text{SoftAllEqual}(k, [x_1, \ldots, x_n]) \iff \) 

At least \( k \) pairs of equal variables in the set \( \{x_1, \ldots, x_n\} \)

- **Meeting Scheduling**
  - Each researcher states his/her availabilities
  - Each researcher goes to at most one meeting
  - Schedule meetings so that:
    - The number of **interactions** is maximum
    - **Interaction:** a pair of researchers attend the same meeting
## The SoftAllEqual Constraint

<table>
<thead>
<tr>
<th></th>
<th>Tue. am</th>
<th>Tue. pm</th>
<th>Wed. am</th>
<th>Wed. pm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Bob</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Dave</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Eve</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Frank</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Guy</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Hugo</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ian</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Joe</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
The **SoftAllEqual** Constraint

<table>
<thead>
<tr>
<th></th>
<th>Tue. am</th>
<th>Tue. pm</th>
<th>Wed. am</th>
<th>Wed. pm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Bob</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Charlie</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Dave</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Eve</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Frank</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Guy</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Hugo</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Ian</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Joe</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Alice — Hugo

Bob

Charlie

Joe

Ian

Dave

Eve

Guy
The SoftAllEqual Constraint

Alice
Bob
Charlie
Dave
Eve
Frank
Guy
Hugo
Ian
Joe

Bob — Hugo — Joe — Eve
Guy — Ian — Frank
Finding an optimal solution is NP-hard (and so is Achieving $\Delta C$).
The SoftAllEqual Constraint

- Finding an optimal solution is NP-hard (and so is Achieving $\Delta C$)
- Given an ordering (red, blue, green, ...), we can define a greedy algo:
  - Assign $X$ to the first value whenever possible, then the second, etc.
The SoftAllEqual Constraint

- Finding an optimal solution is NP-hard (and so is Achieving AC)
- Given an ordering (red, blue, green, ...), we can define a greedy algo:
  - Assign $X$ to the first value whenever possible, then the second, etc.

If values are ordered by occurrences: 2-approximation
The SoftAllEqual Constraint

- Finding an optimal solution is NP-hard (and so is Achieving \( AC \))
- Given an ordering (red, blue, green, ...), we can define a greedy algo:
  - Assign \( X \) to the first value whenever possible, then the second, etc.

If values are ordered by occurrences: 2-approximation

There exists such an ordering that gives an optimal solution

- There is an algorithm exponential only in the number of values (\( V \))
The **SoftAllEqual** Constraint

- Finding an optimal solution is NP-hard (and so is Achieving \( AC \))
- Given an ordering (red, blue, green, ...), we can define a greedy algo:
  - Assign \( X \) to the first value whenever possible, then the second, etc.

**If values are ordered by occurrences: 2-approximation**

**There exists such an ordering that gives an optimal solution**

- There is an algorithm exponential only in the number of values \( (V) \)
- Exponential in something that is potentially smaller than the data \( nd \)
  - Fixed Parameter Tractable
The **SoftAllEqual** Constraint

- Achieving $\mathcal{AC}$ is NP-hard
The **SoftAllEqual** Constraint

- Achieving $AC$ is NP-hard
- What about $BC$?
A bc Algorithm for SoftAllEqual

If we assign the value 55 first, we assign it to all variables.

The problem is split into two smaller parts.
If we assign the value 55 first, we assign it to all variables
A bc Algorithm for **SoftAllEqual**

- If we assign the value 55 first, we assign it to all variables.
- The problem is split into two smaller parts.
If we assign the value 55 first, we assign it to all variables

- The problem is split into two smaller parts
  - Polynomial size search tree: Dynamic Programming
Dynamic Programming

• $T_{a,b} =$ Maximum equalities on variables entirely contained in $[a, b]$
Dynamic Programming

- $T_{a,b} = \text{Maximum equalities on variables entirely contained in } [a, b]$
- Given a value $v \in [a, b]$, we must assign variables to $v$ if possible
  - $T_{a,b}(v) = \text{(Binomial coeff of) the number of variables hit by } v +$
\( T_{a,b} = \) Maximum equalities on variables entirely contained in \([a, b]\)

Given a value \( v \in [a, b] \), we must assign variables to \( v \) if possible

- \( T_{a,b}(v) = \) (Binomial coeff of) the number of variables hit by \( v \) +
- table content on the splitted parts: \( T_{a,v-1} + T_{v+1,b} \)
Dynamic Programming

- Time complexity: $O(nV^3)$
  - Table of size $V^2$, for each cell, go through the $V$ values and count occurrences ($O(n)$)
- Since we consider interval domains, $V$ can be very large!
Dynamic Programming

- **Time complexity**: $O(nV^3)$
  - Table of size $V^2$, for each cell, go through the $V$ values and count occurrences ($O(n)$)
- Since we consider interval domains, $V$ can be very large!
  - We can count occurrences of all values in $O(n + V)$
Dynamic Programming

- Time complexity: $O(nV^3)$
  - Table of size $V^2$, for each cell, go through the $V$ values and count occurrences ($O(n)$)
- Since we consider interval domains, $V$ can be very large!
  - We can count occurrences of all values in $O(n + V)$
  - Only maximal cliques matter
Dynamic Programming

- Time complexity: $O(nV^3)$
  - Table of size $V^2$, for each cell, go through the $V$ values and count occurrences ($O(n)$)

- Since we consider interval domains, $V$ can be very large!
  - We can count occurrences of all values in $O(n + V)$
  - Only maximal cliques matter
  - We can reformulate in an equivalent problem with as many values as maximal cliques
  - Interval graph, we can find all ($O(n)$) cliques in $O(n \log n)$ time
Dynamic Programming

- Time complexity: \( O(nV^3) \)
  - Table of size \( V^2 \), for each cell, go through the \( V \) values and count occurrences \( (O(n)) \)

- Since we consider interval domains, \( V \) can be very large!
  - We can count occurrences of all values in \( O(n + V) \)
  - Only maximal cliques matter
  - We can reformulate in an equivalent problem with as many values as maximal cliques
  - Interval graph, we can find all \( (O(n)) \) cliques in \( O(n \log n) \) time

- Algorithm in \( O(n^3) \) to find a solution and \( O(n^4) \) for BC
Assessing the Tradeoff

- If solving is NP-hard, then achieving $\Delta C$ is NP-hard
Assessing the Tradeoff

- If solving is **NP-hard**, then achieving $\Delta C$ is **NP-hard**
- Should we **decompose** into simpler constraints?
Assessing the Tradeoff

- If solving is **NP-hard**, then achieving $\Delta C$ is **NP-hard**
- Should we **decompose** into simpler constraints?
- There are alternatives
Assessing the Tradeoff

- If solving is **NP-hard**, then achieving $\Delta C$ is **NP-hard**
- Should we **decompose** into simpler constraints?
- There are alternatives
  - Achieve a **weaker** consistency (e.g., BC)
Assessing the Tradeoff

- If solving is **NP-hard**, then achieving $\Delta C$ is **NP-hard**
- Should we **decompose** into simpler constraints?
- There are alternatives
  - Achieve a **weaker** consistency (e.g., $BC$)
  - Approximate $\Delta C$
Assessing the Tradeoff

- If solving is NP-hard, then achieving AC is NP-hard
- Should we decompose into simpler constraints?
- There are alternatives
  - Achieve a weaker consistency (e.g., BC)
  - Approximate AC
  - Use an exponential algorithm (good case: Fixed Parameter Tractable)
Assessing the Tradeoff

- If solving is **NP-hard**, then achieving $\Delta C$ is **NP-hard**
- Should we **decompose** into simpler constraints?
- There are alternatives
  - Achieve a **weaker** consistency (e.g., $BC$)
  - Approximate $\Delta C$
  - Use an exponential algorithm (good case: **Fixed Parameter Tractable**)
- The best **tradeoff** might not be obvious
  - Theoretical comparison
  - Empirical comparison
Assessing the Tradeoff: The $\text{AtMostNValue}$ Constraint

**Definition:** $\text{AtMostNValue}([x_1, \ldots, x_n], N) \iff$

$[x_1, \ldots, x_n]$ assigned using at most $N$ distinct values
Assessing the Tradeoff: The AtMostNValue Constraint

Definition: \texttt{AtMostNValue}([x_1, \ldots, x_n], N) ⇔

[x_1, \ldots, x_n] assigned using at most \( N \) distinct values

- NP-hard, equivalent to Minimum Hitting Set
Assessing the Tradeoff: The \textsc{AtMostNValue} Constraint

\textbf{Definition:} \textsc{AtMostNValue}([x_1, \ldots, x_n], N) ⇐⇒

\[ [x_1, \ldots, x_n] \text{ assigned using at most } N \text{ distinct values} \]

- NP-hard, equivalent to \textsc{Minimum Hitting Set}
Assessing the Tradeoff: The \texttt{AtMostNValue} Constraint

**Definition:** \texttt{AtMostNValue}([\(x_1, \ldots, x_n\)], \(N\)) ⇐⇒

\([x_1, \ldots, x_n]\) assigned using at most \(N\) distinct values

- NP-hard, equivalent to \textit{Minimum Hitting Set}
  - Domains = collection of sets \((D(x_i) = S_i)\)
Assessing the Tradeoff: The $\textsc{AtMostNValue}$ Constraint

**Definition:** $\textsc{AtMostNValue}([x_1, \ldots, x_n], N) \iff$ 

$[x_1, \ldots, x_n]$ assigned using at most $N$ distinct values

- NP-hard, equivalent to Minimum Hitting Set
  - Domains = collection of sets ($D(x_i) = S_i$)
  - $x_i = \text{an element in } H \cap S_i$
Assessing the Tradeoff: The **AtMostNValue** Constraint

**Definition:** \( \text{AtMostNValue}([x_1, \ldots, x_n], N) \iff \) 

\[ [x_1, \ldots, x_n] \text{ assigned using at most } N \text{ distinct values} \]

- NP-hard, equivalent to **Minimum Hitting Set**
  - Domains = collection of sets \( \mathcal{D}(x_i) = S_i \)
  - \( x_i \) = an element in \( H \cap S_i \)
  - \( N = |H| \)
Hitting Set

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]
Hitting Set

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]
Hitting Set

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]
Hitting Set

\[ \mathcal{D}(a) = \{2, 3\} \]
\[ \mathcal{D}(b) = \{3, 4\} \]
\[ \mathcal{D}(c) = \{1, 4, 5, 7\} \]
\[ \mathcal{D}(d) = \{5, 6\} \]
\[ \mathcal{D}(e) = \{6, 7\} \]
\[ \mathcal{D}(f) = \{2, 3, 7\} \]
**Hitting Set**

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]
Hitting Set, Clique Cover

Can we approximate Minimum Hitting Set to filter AtMostNValue?

▶ We need a lower bound; approximating MHS gives us an upper bound

▶ Another approach: Clique Cover of the Intersection Graph
Hitting Set, Clique Cover

Can we approximate Minimum Hitting Set to filter $\text{AtMostNValue}$?
Hitting Set, Clique Cover

- Can we approximate Minimum Hitting Set to filter \text{AtMostNValue}?
  - We need a lower bound; approximating MHS gives us an upper bound
  - Another approach: Clique Cover of the Intersection Graph
Intersection Graph: Discrete domains

\[ \mathcal{D}(a) = \{2, 3\} \]
\[ \mathcal{D}(b) = \{3, 4\} \]
\[ \mathcal{D}(c) = \{1, 4, 5, 7\} \]
\[ \mathcal{D}(d) = \{5, 6\} \]
\[ \mathcal{D}(e) = \{6, 7\} \]
\[ \mathcal{D}(f) = \{2, 3, 7\} \]
Intersection Graph: Discrete domains

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]
Intersection Graph: Discrete domains

\[ \mathcal{D}(a) = \{2, 3\} \]
\[ \mathcal{D}(b) = \{3, 4\} \]
\[ \mathcal{D}(c) = \{1, 4, 5, 7\} \]
\[ \mathcal{D}(d) = \{5, 6\} \]
\[ \mathcal{D}(e) = \{6, 7\} \]
\[ \mathcal{D}(f) = \{2, 3, 7\} \]
Intersection Graph: Discrete domains

\[ \mathcal{D}(a) = \{2, 3\} \]
\[ \mathcal{D}(b) = \{3, 4\} \]
\[ \mathcal{D}(c) = \{1, 4, 5, 7\} \]
\[ \mathcal{D}(d) = \{5, 6\} \]
\[ \mathcal{D}(e) = \{6, 7\} \]
\[ \mathcal{D}(f) = \{2, 3, 7\} \]
Intersection Graph: Discrete domains

\[ \mathcal{D}(a) = \{2, 3\} \]
\[ \mathcal{D}(b) = \{3, 4\} \]
\[ \mathcal{D}(c) = \{1, 4, 5, 7\} \]
\[ \mathcal{D}(d) = \{5, 6\} \]
\[ \mathcal{D}(e) = \{6, 7\} \]
\[ \mathcal{D}(f) = \{2, 3, 7\} \]
Intersection Graph: Discrete domains

$\mathcal{D}(a) = \{2, 3\}$
$\mathcal{D}(b) = \{3, 4\}$
$\mathcal{D}(c) = \{1, 4, 5, 7\}$
$\mathcal{D}(d) = \{5, 6\}$
$\mathcal{D}(e) = \{6, 7\}$
$\mathcal{D}(f) = \{2, 3, 7\}$

A Minimum Clique Cover is a lower bound
Intersection Graph: Discrete domains

\[ \mathcal{D}(a) = \{2, 3\} \]
\[ \mathcal{D}(b) = \{3, 4\} \]
\[ \mathcal{D}(c) = \{1, 4, 5, 7\} \]
\[ \mathcal{D}(d) = \{5, 6\} \]
\[ \mathcal{D}(e) = \{6, 7\} \]
\[ \mathcal{D}(f) = \{2, 3, 7\} \]

- A Minimum Clique Cover is a lower bound
- However, finding a Minimum Clique Cover is NP-hard
Intersection Graph: Interval domains

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]
Intersection Graph: Interval domains

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, \ldots, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, \ldots, 7\} \]
Intersection Graph: Interval domains

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, \ldots, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, \ldots, 7\} \]
Finding a Minimum Clique Cover is polynomial ($O(n \log n)$)
Finding a Minimum Clique Cover is polynomial ($O(n \log n)$)

A Minimum Clique Cover is an exact lower bound [Beldiceanu 2001]
Find an Independent Set

Independence Number of $G \leq$ Intersection Number of $\theta(G)$

$\mathcal{D}(a) = \{2, 3\}$
$\mathcal{D}(b) = \{3, 4\}$
$\mathcal{D}(c) = \{1, 4, 5, 7\}$
$\mathcal{D}(d) = \{5, 6\}$
$\mathcal{D}(e) = \{6, 7\}$
$\mathcal{D}(f) = \{2, 3, 7\}$
Independent Set

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]

- Find an Independent Set
- Indepedance Number of \( G \leq \) Intersection Number of \( \theta(G) \)
Independent Set

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]

- Find an Independent Set
- Independance Number of \( G \leq \) Intersection Number of \( \theta(G) \)
Independent Set

\[ D(a) = \{2, 3\} \]
\[ D(b) = \{3, 4\} \]
\[ D(c) = \{1, 4, 5, 7\} \]
\[ D(d) = \{5, 6\} \]
\[ D(e) = \{6, 7\} \]
\[ D(f) = \{2, 3, 7\} \]

- Find an Independent Set
- Indepedance Number of \( G \) \( \leq \) Intersection Number of \( \theta(G) \)
There are even simpler lower bounds

\[ \alpha(G) \leq \left\lceil \frac{n^2}{2m + n} \right\rceil \]
There are even simpler lower bounds

- [Turán 1941]
- [Favaron et al. 1993]

\[
\alpha(G) \leq \left\lfloor \frac{n^2}{2m + n} \right\rfloor \leq \left\lfloor 2n - \frac{2m}{\left\lfloor \frac{2m}{n} \right\rfloor + 1} \right\rfloor
\]
Linear Program

\[
\text{minimize } \sum_{v \in D} y_v \\
\text{subject to } \sum_{v \in x_i} y_v \geq 1, x_i \in X \\
y_v \geq 0, v \in D
\]

number of values each variable takes a value

An integral solution is a Hitting Set

The linear relaxation gives a lower bound
An integral solution is a Hitting Set

minimize \[ \sum_{v \in \mathcal{D}} y_v \]
subject to \[ \sum_{v \in x_i} y_v \geq 1, \ x_i \in \mathcal{X} \]
\[ y_v \geq 0 \quad , \ v \in \mathcal{D} \]

number of values

each variable takes a value
Linear Program

\[
\text{minimize } \sum_{v \in \mathcal{D}} y_v \\
\text{subject to } \sum_{v \in \mathcal{X}_i} y_v \geq 1, \quad x_i \in \mathcal{X} \\
y_v \geq 0, \quad v \in \mathcal{D}
\]

- An integral solution is a Hitting Set
- The linear relaxation gives a lower bound
Tradeoff: Theory

- Which solution is best?
Tradeoff: Theory

- Which solution is best?
  - Better time complexity vs. better propagation

Tóran: MD

experiments
Tradeoff: Theory

- Which solution is best?
  - Better time complexity vs. better propagation
- Comparing propagation $\phi$ against $\psi$, constraint $C$ domain $D$

experiments
Tradeoff: Theory

- Which solution is best?
  - Better time complexity vs. better propagation
- Comparing propagation $\phi$ against $\psi$, constraint $C$ domain $D$
  - $\phi(C,D)$ the domain representation / set of solutions after applying $\phi$
  - $\psi(C,D)$ the domain representation / set of solutions after applying $\psi$
Tradeoff: Theory

- Which solution is best?
  - Better time complexity vs. better propagation

- Comparing propagation $\phi$ against $\psi$, constraint $C$ domain $D$
  - $\phi(C, D)$ the domain representation / set of solutions after applying $\phi$
  - $\psi(C, D)$ the domain representation / set of solutions after applying $\psi$

- $\phi$ is as strong as $\psi$ is $\phi(C, D) \subseteq \psi(C, D)$
Tradeoff: Theory

- Which solution is best?
  - Better time complexity vs. better propagation

- Comparing propagation $\phi$ against $\psi$, constraint $C$ domain $D$
  - $\phi(C, D)$ the domain representation / set of solutions after applying $\phi$
  - $\psi(C, D)$ the domain representation / set of solutions after applying $\psi$

- $\phi$ is as strong as $\psi$ is $\phi(C, D) \subseteq \psi(C, D)$
Tradeoff: Theory

- Which solution is best?
  - Better time complexity vs. better propagation

- Comparing propagation $\phi$ against $\psi$, constraint $C$ domain $D$
  - $\phi(C, D)$ the domain representation / set of solutions after applying $\phi$
  - $\psi(C, D)$ the domain representation / set of solutions after applying $\psi$

- $\phi$ is as strong as $\psi$ is $\phi(C, D) \subseteq \psi(C, D)$

**Diagram**

Túran \( \neq \) MD

Túran \( \neq \) LP

Túran \( \neq \) BC

**Complexity**

- Túran amortised
- BC \( O(n \log n) \)
- MD \( O(n^2d) \)
- LP \( O(nd^4) \)
- AC NP-hard
Conclusion

- When **solving** a problem: find the algorithm with best complexity
When solving a problem: find the algorithm with best complexity

- Here we also want to achieve the highest possible consistency
  - $AC$? $BC$? $MDD-C$? (SAC, MaxRPC, ...)

Even something undefined when achieving a given consistency is NP-hard

- Consider a lower consistency
- Consider approximations
- Consider parameterized complexity

Other related questions:

- Is it decomposable?
- What consistency/decomposition is the strongest?

Experimental tradeoff!
When solving a problem: find the algorithm with best complexity

- Here we also want to achieve the highest possible consistency
  - AC? BC? MDD-C? (SAC, MaxRPC, ...)
  - Even something undefined
Conclusion

- When **solving** a problem: find the algorithm with best complexity
  - Here we also want to achieve the highest possible consistency
    - $AC$? $BC$? MDD-C? (SAC, MaxRPC, ...)
    - Even something undefined
- When achieving a given consistency is NP-hard
  - Consider a lower consistency
  - Consider approximations
  - Consider parameterized complexity
- Other related questions:
  - Is it decomposable?
  - What consistency/decomposition is the **strongest**?
- Experimental tradeoff!
Questions?
Experimental Evaluation (120 garments, 3 machines, 7 colors)
# Experimental results

## Table: Car-sequencing

<table>
<thead>
<tr>
<th>Models</th>
<th>G1 (70 × 34 × 5)</th>
<th>G2 (4 × 34 × 5)</th>
<th>G3 (5 × 34 × 5)</th>
<th>G4 (7 × 34 × 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#sol</td>
<td>time</td>
<td>#sol</td>
<td>time</td>
</tr>
<tr>
<td>sum</td>
<td>8480</td>
<td>13.93</td>
<td>95</td>
<td>76.60</td>
</tr>
<tr>
<td>gsc</td>
<td>11218</td>
<td>3.60</td>
<td>325</td>
<td>110.99</td>
</tr>
<tr>
<td>amsc</td>
<td>10702</td>
<td>4.43</td>
<td>360</td>
<td>72.00</td>
</tr>
<tr>
<td>amsc+gsc</td>
<td><strong>11243</strong></td>
<td>3.43</td>
<td>339</td>
<td>106.53</td>
</tr>
</tbody>
</table>

Sum: simple decomposition

GSC (Global Sequencing Constraint) is the same as AtMostSeqCard for an option + demand for each type of car requiring this option. It is NP-hard, and there is an approximate algorithm.
## Experimental results

Table: Car-sequencing

<table>
<thead>
<tr>
<th>Models</th>
<th>G1 (70 × 34 × 5)</th>
<th>G2 (4 × 34 × 5)</th>
<th>G3 (5 × 34 × 5)</th>
<th>G4 (7 × 34 × 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#sol</td>
<td>time</td>
<td>#sol</td>
<td>time</td>
</tr>
<tr>
<td>sum</td>
<td>8480</td>
<td>13.93</td>
<td>95</td>
<td>76.60</td>
</tr>
<tr>
<td>gsc</td>
<td>11218</td>
<td>3.60</td>
<td>325</td>
<td>110.99</td>
</tr>
<tr>
<td>amsc</td>
<td>10702</td>
<td>4.43</td>
<td>360</td>
<td>72.00</td>
</tr>
<tr>
<td>amsc+gsc</td>
<td>11243</td>
<td>3.43</td>
<td>339</td>
<td>106.53</td>
</tr>
</tbody>
</table>

- **Sum**: simple decomposition
## Experimental results

**Table:** Car-sequencing

<table>
<thead>
<tr>
<th>Models</th>
<th>G1 (70 × 34 × 5)</th>
<th>G2 (4 × 34 × 5)</th>
<th>G3 (5 × 34 × 5)</th>
<th>G4 (7 × 34 × 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11900</td>
<td>680</td>
<td>850</td>
<td>1190</td>
</tr>
<tr>
<td>sum</td>
<td>#sol</td>
<td>time</td>
<td>#sol</td>
<td>time</td>
</tr>
<tr>
<td>gsc</td>
<td>8480</td>
<td>13.93</td>
<td>95</td>
<td>76.60</td>
</tr>
<tr>
<td>amsc</td>
<td>11218</td>
<td>3.60</td>
<td>325</td>
<td>110.99</td>
</tr>
<tr>
<td>amsc + gsc</td>
<td>10702</td>
<td>4.43</td>
<td>360</td>
<td>72.00</td>
</tr>
<tr>
<td>sum</td>
<td>11243</td>
<td>3.43</td>
<td>339</td>
<td>106.53</td>
</tr>
</tbody>
</table>

- **Sum:** simple decomposition
- **GSC** (Global Sequencing Constraint)
Experimental results

[label=expeamsc]

<table>
<thead>
<tr>
<th>Models</th>
<th>G1 (70 × 34 × 5)</th>
<th>G2 (4 × 34 × 5)</th>
<th>G3 (5 × 34 × 5)</th>
<th>G4 (7 × 34 × 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum</td>
<td>#sol  8480  13.93</td>
<td>#sol  95  76.60</td>
<td>#sol  0  &gt; 1200</td>
<td>#sol  64  43.81</td>
</tr>
<tr>
<td>gsc</td>
<td>11218  3.60</td>
<td>325  110.99</td>
<td>31  276.06</td>
<td>140  56.61</td>
</tr>
<tr>
<td>amsc</td>
<td>10702  4.43</td>
<td>360  72.00</td>
<td>16  8.62</td>
<td>153  33.56</td>
</tr>
<tr>
<td>amsc+gsc</td>
<td><strong>11243</strong>  <strong>3.43</strong></td>
<td>339  106.53</td>
<td><strong>32</strong>  <strong>285.43</strong></td>
<td>147  66.45</td>
</tr>
</tbody>
</table>

- **SUM**: simple decomposition
- **GSC** (Global Sequencing Constraint)
  - same as **AtMostSeqCard** for an option + demand for each type of car requiring this option
  - NP-hard, approximation
**Experimental results**

**Table: Crew-Rostering**

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Lexicographic</th>
<th>MultiAtMostSeqCard: several capacity constraints together</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>satifiable (1140)</td>
<td>unsatisfiable (385)</td>
</tr>
<tr>
<td>sum</td>
<td>#sol</td>
<td>time</td>
</tr>
<tr>
<td>gsc</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>amsc</td>
<td>25</td>
<td>308.93</td>
</tr>
<tr>
<td>mamsc</td>
<td>125</td>
<td>164.36</td>
</tr>
<tr>
<td></td>
<td>534</td>
<td>87.29</td>
</tr>
</tbody>
</table>
### Experimental results

#### Table: Crew-Rostering

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Lexicographic</th>
<th>Model</th>
<th>satifiable (1140)</th>
<th>unsatisfiable (385)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>#sol</td>
<td>time</td>
<td>avg bts</td>
</tr>
<tr>
<td>sum</td>
<td></td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>gsc</td>
<td></td>
<td>25</td>
<td>308.93</td>
<td>74074</td>
</tr>
<tr>
<td>amsc</td>
<td></td>
<td>125</td>
<td>164.36</td>
<td>1828347</td>
</tr>
<tr>
<td>mamsc</td>
<td></td>
<td>534</td>
<td>87.29</td>
<td>685720</td>
</tr>
</tbody>
</table>

- **MultiAtMostSeqCard**: several capacity constraints together
Linear time algorithm

We need to do each of the following in $O(1)$:

- access the cardinality of the $j$th subsequence containing $x_i$: $\text{card}(i, j)$

\[
\begin{array}{cccccccccccc}
1 & 0 & 0 & 1 & 1 & 0 & . & 0 & . & 0 & 1 & . & . & 1 & . & . & . & . & . & . & . & . & . & 1
\end{array}
\]
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i,j)$

\[
\begin{align*}
1 & 0 0 1 1 0 . 0 . 0 1 . . 1 . . . . . . . 1 \\
\end{align*}
\]
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i, j)$
    - jump from $x_i$ to $x_{i+1}$

1 0 0 1 1 0 0 0 . 0 1 . . 1 . . . . . . . . . . . 1

- $\text{card}(7, 0) = 2$
- $\text{card}(7, 1) = 1$
- $\text{card}(7, 2) = 0$
- $\text{card}(7, 3) = 0$
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i,j)$
    - jump from $x_i$ to $x_{i+1}$

$\begin{align*}
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & . & 0 & 1 & . & 1 & . & . & . & . & . & . & . & . & . & . & . & . & . & . & 1
\end{align*}$

- $\text{card}(7, 0) = 2$
- $\text{card}(7, 1) = 1$
- $\text{card}(7, 2) = 0$
- $\text{card}(7, 3) = 0$
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i,j)$
  - jump from $x_i$ to $x_{i+1}$

\[
\text{card}(8, 0) = 1 \\
\text{card}(8, 1) = 0 \\
\text{card}(8, 2) = 0 \\
\text{card}(8, 3) = 1
\]
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i, j)$
  - jump from $x_i$ to $x_{i+1}$

```
1 0 0 1 1 0 0 0 . 0 1 . . 1 . . . . . . . . 1
```

- $\text{card}(7, 0) = 2$
- $\text{card}(7, 1) = 1$
- $\text{card}(7, 2) = 0$
- $\text{card}(7, 3) = 0$
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i, j)$
  - jump from $x_i$ to $x_{i+1}$

1 0 0 1 1 0 0 0 . 0 1 . . 1 . . . . . . . . . . . 1

- $\text{card}(7, 0) = 3 - 1 = 2$
- $\text{card}(7, 1) = 3 - 2 = 1$
- $\text{card}(7, 2) = 3 - 3 = 0$
- $\text{card}(7, 3) = 3 - 3 = 0$
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: \( \text{card}(i, j) \)
  - jump from $x_i$ to $x_{i+1}$

\[
\begin{array}{ccccccccccccccccccc}
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & . & 0 & 1 & . & . & 1 & . & . & . & . & . & 1
\end{array}
\]

- \( \text{card}(8, 0) = 3 - 2 = 1 \)
- \( \text{card}(8, 1) = 3 - 3 = 0 \)
- \( \text{card}(8, 2) = 3 - 3 = 0 \)
- \( \text{card}(8, 3) = 3 - 2 = 1 \)

\( \text{card}(i, j) = \sum_{k=1}^{i} c[i + j \mod q] \)
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i, j)$
    - jump from $x_i$ to $x_{i+1}$
    - increment all subsequences containing $x_i$

1 0 0 1 1 0 0 0 . 0 1 . . . . . . . . 1

- $\text{card}(8, 0) = 3 - 2 = 1$
- $\text{card}(8, 1) = 3 - 3 = 0$
- $\text{card}(8, 2) = 3 - 3 = 0$
- $\text{card}(8, 3) = 3 - 2 = 1$

- $\text{card}(i, j) = \sum_{k=1}^{i} + c[i + j \mod q]$
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i,j)$
    - jump from $x_i$ to $x_{i+1}$
    - increment all subsequences containing $x_i$

1 0 0 1 1 0 0 0 1 0 1 . . 1 . . . . . . . 1

- $\text{card}(8, 0) = 3 - 2 = 1$
- $\text{card}(8, 1) = 3 - 3 = 0$
- $\text{card}(8, 2) = 3 - 3 = 0$
- $\text{card}(8, 3) = 3 - 2 = 1$

- $\text{card}(i,j) = \sum_{k=1}^{i} + c[i + j \mod q]$
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - access the cardinality of the $j^{th}$ subsequence containing $x_i$: $\text{card}(i, j)$
    - jump from $x_i$ to $x_{i+1}$
    - increment all subsequences containing $x_i$

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]

- $\text{card}(8, 0) = 4 - 2 = 2$
- $\text{card}(8, 1) = 4 - 3 = 1$
- $\text{card}(8, 2) = 4 - 3 = 1$
- $\text{card}(8, 3) = 4 - 2 = 2$

- $\text{card}(i, j) = \sum_{k=1}^{i} c[i + j \mod q]$
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - check whether there exists a subsequence of cardinality $u$: $\#\text{sub}(i, k)$
    - jump from $x_i$ to $x_{i+1}$
    - increment all subsequences containing $x_i$
Linear time algorithm

We need to do each of the following in \( O(1) \):
- check whether there exists a subsequence of cardinality \( u \): \( \# \text{sub}(i, k) \)
  - jump from \( x_i \) to \( x_{i+1} \)
  - increment all subsequences containing \( x_i \)
Linear time algorithm

We need to do each of the following in $O(1)$:

- check whether there exists a subsequence of cardinality $u$: $\#\text{sub}(i, k)$
  - jump from $x_i$ to $x_{i+1}$ $\rightarrow$ OK using $\text{card}(i, 0)$ and $\text{card}(i + 1, 4)$
  - increment all subsequences containing $x_i$

```
0 0 0 0 0 0 3 1 2 0
```

```
0 1 2 3
```
Linear time algorithm

We need to do each of the following in $O(1)$:

- check whether there exists a subsequence of cardinality $u$: $\#\text{sub}(i, k)$
  - jump from $x_i$ to $x_{i+1}$ → OK using $\text{card}(i, 0)$ and $\text{card}(i + 1, 4)$
  - increment all subsequences containing $x_i$ → decrement pointer

```
0 0 0 0 0 0 3 1 2 0
```

```
0 1 2 3
```
Linear time algorithm

- We need to do each of the following in $O(1)$:
  - check whether there exists a subsequence of cardinality $u$: $\# sub(i, k)$
    - jump from $x_i$ to $x_{i+1}$ → OK using $card(i, 0)$ and $card(i + 1, 4)$
    - increment all subsequences containing $x_i$ → decrement pointer

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>3</th>
<th>1</th>
<th>2</th>
<th>0</th>
</tr>
</thead>
</table>

0 1 2 3
Tradeoff: In Practice

- Random instances
- Dominating set of the Queens Graph of order $n$
  - One vertex per square of the chessboard ($n^2$)
  - An edge iff if the two squares are attacked (same row, column or diagonal)
Tradeoff: In Practice

- Random instances
- Dominating set of the Queens Graph of order $n$
  - One vertex per square of the chessboard ($n^2$)
  - An edge iff if the two squares are attacked (same row, column or diagonal)

- minimum number of Queens to attack the whole chessboard

<table>
<thead>
<tr>
<th>problem</th>
<th>Túran</th>
<th>BC</th>
<th>MD</th>
<th>LP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random $n = 50, d = 15$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queens $(9 \times 9)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tradeoff: In Practice

- Random instances
- Dominating set of the Queens Graph of order $n$
  - One vertex per square of the chessboard ($n^2$)
  - An edge iff if the two squares are attacked (same row, column or diagonal)
- minimum number of Queens to attack the whole chessboard

<table>
<thead>
<tr>
<th>problem</th>
<th>Túran</th>
<th>BC</th>
<th>MD</th>
<th>LP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random $n = 50, d = 15$</td>
<td>7.2</td>
<td>6.9</td>
<td>2.5</td>
<td>15.8</td>
</tr>
<tr>
<td></td>
<td>90786</td>
<td>72773</td>
<td>18885</td>
<td>17866</td>
</tr>
<tr>
<td>Queens (9 $\times$ 9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tradeoff: In Practice

- Random instances
- Dominating set of the Queens Graph of order $n$
  - One vertex per square of the chessboard ($n^2$)
  - An edge iff if the two squares are attacked (same row, column or diagonal)
- minimum number of Queens to attack the whole chessboard

<table>
<thead>
<tr>
<th>problem</th>
<th>Túran</th>
<th>BC</th>
<th>MD</th>
<th>LP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time</td>
<td>time</td>
<td>time</td>
<td>time</td>
</tr>
<tr>
<td></td>
<td>bkt</td>
<td>bkt</td>
<td>bkt</td>
<td>bkt</td>
</tr>
<tr>
<td>Random $n = 50, d = 15$</td>
<td>7.2</td>
<td>6.9</td>
<td>2.5</td>
<td>15.8</td>
</tr>
<tr>
<td></td>
<td>90786</td>
<td>72773</td>
<td>18885</td>
<td>17866</td>
</tr>
<tr>
<td>Queens (9 × 9)</td>
<td>-</td>
<td>-</td>
<td>203</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>-</td>
<td>880669</td>
<td>2243</td>
</tr>
</tbody>
</table>