The Multi-Inter-Distance Constraint

Pierre Ouellet and Claude-Guy Quimper
Introduction

- The **MULTI-INTER-DISTANCE** constraint is a new global constraint.
- It is useful to model scheduling problems.
- We present a filtering algorithm achieving bounds consistency.
- The filtering algorithm relies on the theory of the shortest paths in a graph.
- We experimented on the runway scheduling problem.
Multi-Inter-Distance

- The constraint $\text{MULTI-INTER-DISTANCE}([X_1, \ldots X_n], m, p)$ is satisfied iff no more than $m$ variables are assigned to values lying in a window of $p$ consecutive values.
The constraint \textsc{Multi-Inter-Distance}([X_1, \ldots X_n], m, p) is satisfied iff no more than \( m \) variables are assigned to values lying in a window of \( p \) consecutive values.

Example: \textsc{Multi-Inter-Distance}([X_1, X_2, X_3, X_4, X_5], m = 2, p = 3)
\textbf{Multi-Inter-Distance}

- The constraint \textsc{Multi-Inter-Distance}([X_1, \ldots, X_n], m, p) is satisfied iff no more than \textit{m} variables are assigned to values lying in a window of \textit{p} consecutive values.

- Example: \textsc{Multi-Inter-Distance}([X_1, X_2, X_3, X_4, X_5], m = 2, p = 3)
**Multi-Inter-Distance**

- The constraint $\text{MULTI-INTER-DISTANCE}([X_1, \ldots, X_n], m, p)$ is satisfied iff no more than $m$ variables are assigned to values lying in a window of $p$ consecutive values.

- Example: $\text{MULTI-INTER-DISTANCE}([X_1, X_2, X_3, X_4, X_5], m = 2, p = 3)$

Variable count: 2
**MULTI-INTER-DISTANCE**

- The constraint \( \text{MULTI-INTER-DISTANCE}([X_1, \ldots X_n], m, p) \) is satisfied iff no more than \( m \) variables are assigned to values lying in a window of \( p \) consecutive values.

- Example: \( \text{MULTI-INTER-DISTANCE}([X_1, X_2, X_3, X_4, X_5], m = 2, p = 3) \)

Variable count: 2

\[
\begin{array}{cccccccc}
X_4 & & & & & & & X_3 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
\end{array}
\]

X_2, X_3, X_4}
**MULTI-INTER-DISTANCE**

- The constraint \( \text{MULTI-INTER-DISTANCE}([X_1, \ldots X_n], m, p) \) is satisfied iff no more than \( m \) variables are assigned to values lying in a window of \( p \) consecutive values.

- Example: \( \text{MULTI-INTER-DISTANCE}([X_1, X_2, X_3, X_4, X_5], m = 2, p = 3) \)

```
X4   X2   X5   X1   X3
1    2    3    4    5
```

Variable count: 1
Multi-Inter-Distance

- The constraint \text{Multi-Inter-Distance}([X_1, \ldots X_n], m, p) is satisfied iff no more than \textit{m} variables are assigned to values lying in a window of \textit{p} consecutive values.

- Example: \text{Multi-Inter-Distance}([X_1, X_2, X_3, X_4, X_5], m = 2, p = 3)

\begin{center}
\begin{tabular}{ccccccc}
X_4 & X_2 & X_5 & X_1 & X_3 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\end{tabular}
\end{center}

Variable count: 2
**MULTI-INTER-DISTANCE**

- The **MULTI-INTER-DISTANCE** constraint encodes a scheduling problem where the variables $X_i$ are the starting times of the task.
- Each task has a processing time $p$.
- There are $m$ identical resources.
The Multi-Inter-Distance constraint encodes a scheduling problem where the variables \( X_i \) are the starting times of the task.

- Each task has a processing time \( p \).
- There are \( m \) identical resources.
- This constraint encodes other constraints.
The MULTI-INTER-DISTANCE constraint encodes a scheduling problem where the variables $X_i$ are the starting times of the task.

- Each task has a processing time $p$.
- There are $m$ identical resources.
- This constraint encodes other constraints.
- For $m = 1$ and $p = 1$, the MULTI-INTER-DISTANCE constraint encodes the ALL-DIFFERENT constraint.
The **MULTI-INTER-DISTANCE** constraint encodes a scheduling problem where the variables $X_i$ are the starting times of the task.

- Each task has a processing time $p$.
- There are $m$ identical resources.

This constraint encodes other constraints:

- For $m = 1$ and $p = 1$, the **MULTI-INTER-DISTANCE** constraint encodes the **ALL-DIFFERENT** constraint.
- When $m = 1$, the constraint specializes into the **INTER-DISTANCE** constraint.
• **Domain consistency** is NP-Hard to enforce as it is for the Inter-Distance constraint.
  [Artiouchine and Baptiste 2005]
Consistencies

- **Domain consistency** is NP-Hard to enforce as it is for the Inter-Distance constraint. [Artiouchine and Baptiste 2005]

- We show how to enforce **bounds consistency** in polynomial time.
  - We assume that the domain of a variable $X_i$ is an interval $[l_i, u_i]$.
  - We want to shrink this interval to remove all values that are not involved in any solution.
Test for Satisfiability

- The Multi-Inter-Distance constraint is satisfiable iff the following scheduling problem has a solution:
  - Task $i$ starts at or after time $l_i$ but before time $u_i$
  - Task $i$ is executed without preemption for $p$ units of time
  - Task $i$ does not overload one of the $m$ resources.
- This scheduling problem is solved in time $O(n^2 \min(1, p/m))$ [López-Ortiz & Quimper, 2011].
- We use this scheduling algorithm as a sub-routine in our filtering algorithm.
Scheduling Graph

**Multi-Inter-Distance**([X_1, \ldots, X_5], m = 2, p = 3)

X_1 \in [7, 9), X_2 \in [2, 4), X_3 \in [4, 7), X_4 \in [2, 7), X_5 \in [3, 5)

---

**Forward Edges**

Connect two time points that are **p** units of time apart with an edge of weight **m**.
Scheduling Graph

Multi-Inter-Distance([X_1, \ldots, X_5], m = 2, p = 3)

X_1 \in [7, 9), X_2 \in [2, 4), X_3 \in [4, 7), X_4 \in [2, 7), X_5 \in [3, 5)

Null Edges

Connect a time point with its predecessor with an edge of weight 0.
**Scheduling Graph**

Multi-Inter-Distance([X_1, \ldots, X_5], m = 2, p = 3)

X_1 \in [7, 9), \ X_2 \in [2, 4), \ X_3 \in [4, 7), \ X_4 \in [2, 7), \ X_5 \in [3, 5)

**Backward Edges**

Connect an upper bound with a lower bound. The absolute value of the weight is the number of domains contained in the interval spanned by the edge.
The Multi-Inter-Distance constraint is satisfiable if and only if the scheduling graph has no negative cycles.

[Dürr & Hurand 2009]
First Pruning Rule

- Consider a variable $X_i$ and its domaine $[l_i, u_i)$.

Scheduling graph: $G$
First Pruning Rule

- Consider a variable $X_i$ and its domain $[l_i, u_i)$.
- Reducing the upper bound leads to a new problem... and a new scheduling graph.
First Pruning Rule

- Consider a variable $X_i$ and its domaine $[l_i, u_i)$.
- Reducing the upper bound leads to a new problem... and a new scheduling graph.
- **Definition:** If the altered scheduling graph has a negative cycle, the interval $[l_i, v)$ is a **forbidden region**.
First Pruning Rule

- Consider a variable $X_i$ and its domain $[l_i, u_i)$.
- Reducing the upper bound leads to a new problem... and a new scheduling graph.

**Definition:** If the altered scheduling graph has a negative cycle, the interval $[l_i, v)$ is a **forbidden region**.

**Theorem:** The forbidden region is effective for all variables whose domain upper bound is greater than or equal to $u_i$. 
First Pruning Rule

Consider a variable $X_i$ and its domain $[l_i, u_i)$.

Reducing the upper bound leads to a new problem... and a new scheduling graph.

**Definition:** If the altered scheduling graph has a negative cycle, the interval $[l_i, v)$ is a **forbidden region**.

**Theorem:** The forbidden region is effective for all variables whose domain upper bound is greater than or equal to $u_i$.

**Rule:** Lower bounds in that forbidden region should be increased to $v$. 

\[ \text{Scheduling graph: } G \]

\[ \text{Altered scheduling graph: } G_{i}^{v} \]
Second Pruning Rule

- Consider a variable domain $\text{dom}(X_i) = [l_i, u_i)$
Second Pruning Rule

- Consider a variable domain $\text{dom}(X_i) = [l_i, u_i)$
- Let $u^*$ be the smallest domain upper bound greater than $l_i$. 
Second Pruning Rule

- Consider a variable domain \( \text{dom}(X_i) = [l_i, u_i] \)
- Let \( u^* \) be the smallest domain upper bound greater than \( l_i \).
- If the altered scheduling graph \( G_{i}^{nu^*} \) has no negative cycles, there exists a solution with \( X_i \in [l_i, u^*] \).
Second Pruning Rule

- Consider a variable domain \( \text{dom}(X_i) = [l_i, u_i] \)
- Let \( u^* \) be the smallest domain upper bound greater than \( l_i \).
- If the altered scheduling graph \( G_{i}^{nu^*} \) has no negative cycles, there exists a solution with \( X_i \in [l_i, u^*) \).
- What is the smallest value in \([l_i, u^*)\) that has a support?
Second Pruning Rule

- Consider a variable domain \( \text{dom}(X_i) = [l_i, u_i] \)
- Let \( u^* \) be the smallest domain upper bound greater than \( l_i \).
- If the altered scheduling graph \( G_{i}^{nu^*} \) has no negative cycles, there exists a solution with \( X_i \in [l_i, u^*) \).
- What is the smallest value in \( [l_i, u^*) \) that has a support?
- **Theorem:** The smallest value that has a support in \( \text{dom}(X_i) \) is the largest value that is at distance 0 from \( l_i \) in \( G_{i}^{nu^*} \).
Second Pruning Rule

• Consider a variable domain \( \text{dom}(X_i) = [l_i, u_i] \)

• Let \( u^* \) be the smallest domain upper bound greater than \( l_i \).

• If the altered scheduling graph \( G^n_{u^*_i} \) has no negative cycles, there exists a solution with \( X_i \in [l_i, u^*) \).

• What is the smallest value in \( [l_i, u^*) \) that has a support?

• **Theorem:** The smallest value that has a support in \( \text{dom}(X_i) \) is the largest value that is at distance 0 from \( l_i \) in \( G^n_{u^*_i} \).

• **Rule:** Compute the shortest paths from \( l_i \) to all the other nodes. Set the new lower bound to the largest value that is at distance 0 from \( l_i \).
Filtering Algorithm

We process the variables in non-decreasing order of upper bounds.

1. Let the interval \([l_i, u_i)\) be the domain of the variable \(X_i\).

2. Let \(u^*\) be the smallest domain upper bound greater than \(l_i\).

3. If the altered scheduling graph \(G^u_i\) has a negative cycle, the interval \([l_i, u^*)\) is a forbidden region and we prune the domains accordingly. Go to 2.

4. If the altered scheduling graph has no negative cycles, let \(v\) be the largest value at distance 0 from \(l_i\).

5. Set the lower bound of \(X_i\) to \(v\).

Running Time Complexity

- Computing a shortest path: $O(n^2 \min(1, \frac{p}{m}))$
- Maximum number of shortest path computations: $2n$
- Total running time complexity: $O(n^3 \min(1, \frac{p}{m}))$
Runway scheduling problem

Landing time intervals
One runway

Number of instances solved vs time

### One runway

**Number of instances solved vs time**

<table>
<thead>
<tr>
<th>MID</th>
<th>Multi-Inter-Distance</th>
<th>$O(n^3 \min(1, \frac{p}{m}))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MVH</td>
<td>Edge-Finder</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>OC</td>
<td>Overload Checking</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>

[Mercier & Van Hentenryck]
Two or Three Runways

Number of instances solved vs time

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>MID</td>
<td>Multi-Inter-Distance</td>
<td>$O(n^3 \min(1, \frac{p}{m}))$</td>
</tr>
<tr>
<td>MVH</td>
<td>Edge-Finder</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>[Mercier &amp; Van Hentenryck]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td>Overload Checking</td>
<td>$O(n \log n)$</td>
</tr>
</tbody>
</table>

The graph shows the number of instances solved over time for different methods, with the following complexities:

- **MID**: Multi-Inter-Distance
- **MVH**: [Mercier & Van Hentenryck]
- **OC**: Overload Checking
Conclusion

• The Multi-Inter-Distance constraint is a new constraint that models certain scheduling problems.

• We showed how to enforce bounds consistency in polynomial time.

• The filtering algorithm relies on the properties of shortest paths in the scheduling graph.

• Experiments on the runway scheduling problem proved that a strong consistency is necessary to efficiently solve the problem.