Constraint Programming / Mathematical Programming Hybrid Methods for Combinatorial Optimization

Philippe Michelon

Université d’Avignon et des Pays de Vaucluse, Laboratoire d’Informatique d’Avignon, F-84911 Avignon

SBPO 2014, Salvador - Ba
History

- Issued from **Artificial Intelligence**
- First Language of Constraint Programming in 1963 (**I. Sutherland**)
- First application in O.R. in the 70s
- Great Success in **Planning, Scheduling, ...**
How does it works?

- CP was first designed for solving Satisfiability Problems: Find a value for each variable $x_i$ ($i = 1, \ldots, n$) such that

$$\begin{align*}
C_j(x) &\text{ is true } \forall j = 1, \ldots, m \\
x_i &\in D_i \ \forall i = 1, \ldots, n
\end{align*}$$

- $D_i$ is the set of possible values for $x_i$
- $C_j$ are constraints on variables $x_i$
How does it works?

- $D_i$ is called the domain of $x_i$
- To each of the constraints $C_j(x)$ is associated a Filtering Algorithm which aims to remove values of $D_i$ which are not possible anymore with respect to the constraint.
- Example: $3x_1 + 5x_2 + 7x_3 \leq 6 \quad x \in \{0, 1\}^3$
How does it work?

- \( D_i \) is called the **domain** of \( x_i \).
- To each of the constraints \( C_j(x) \) is associated a **Filtering Algorithm** which aims to remove values of \( D_i \) which are not possible anymore with respect to the constraint.
- Example: \( 3x_1 + 5x_2 + 7x_3 \leq 6 \) \( \quad x \in \{0, 1\}^3 \)

\[
\Rightarrow x_3 = 0
\]
Propagation

- Once a filtering algorithm succeeded in removing a value for a given variable, the filtering algorithms of the constraints containing this variable are successively called.

- This is the propagation phase
Example

\[
\begin{align*}
C_1 : & \quad 6x_1 + 4x_2 + 4x_3 \geq 5 \\
C_2 : & \quad 3x_1 + 5x_2 + 7x_3 \leq 6 \\
x & \in \{0, 1\}^3
\end{align*}
\]
Example

\[
\begin{align*}
C_1 : & \quad 6x_1 + 4x_2 + 4x_3 \geq 5 \\
C_2 : & \quad 3x_1 + 5x_2 + 7x_3 \leq 6 \\
x & \in \{0, 1\}^3
\end{align*}
\]

\( C_1 \Rightarrow \text{Nothing!} \)
Example

\[
\begin{align*}
C_1 : & \quad 6x_1 + 4x_2 + 4x_3 \geq 5 \\
C_2 : & \quad 3x_1 + 5x_2 + 7x_3 \leq 6 \\
\end{align*}
\]

\[x \in \{0, 1\}^3\]

- \(C_1 \Rightarrow \text{Nothing !}\)
- \(C_2 \Rightarrow x_3 = 0\)
Example

\[
\begin{align*}
C_1 & \quad 6x_1 + 4x_2 \geq 5 \\
C_2 & \quad 3x_1 + 5x_2 \leq 6
\end{align*}
\]
Example

\[
\begin{aligned}
\text{C}_1 & \quad 6x_1 + 4x_2 \geq 5 \\
\text{C}_2 & \quad 3x_1 + 5x_2 \leq 6
\end{aligned}
\]

\[C_1 \Rightarrow x_1 = 1\]
Example

\[
\begin{align*}
C_1 & : 6x_1 + 4x_2 \geq 5 \\
C_2 & : 3x_1 + 5x_2 \leq 6
\end{align*}
\]

\[C_1 \Rightarrow x_1 = 1\]

\[
\begin{align*}
C_1 & : 4x_2 \geq -1 \\
C_2 & : 5x_2 \leq 3
\end{align*}
\]
Example

\[
\begin{align*}
6x_1 + 4x_2 & \geq 5 \\
3x_1 + 5x_2 & \leq 6 \\
4x_2 & \geq -1 \\
5x_2 & \leq 3
\end{align*}
\]

\[C_1 \Rightarrow x_1 = 1\]
\[C_2 \Rightarrow x_2 = 0\]
Example

\[
\begin{align*}
C_1 & \Rightarrow x_1 = 1 \\
C_2 & \Rightarrow x_2 = 0
\end{align*}
\]

\[
\begin{align*}
\left\{ 
C_1 & \quad 6x_1 + 4x_2 \geq 5 \\
C_2 & \quad 3x_1 + 5x_2 \leq 6
\right. \\
\left\{ 
C_1 & \quad 4x_2 \geq -1 \\
C_2 & \quad 5x_2 \leq 3
\right.
\]
\]
Example

\[
\begin{align*}
\begin{cases}
C_1 & \quad 6x_1 + 4x_2 \geq 5 \\
C_2 & \quad 3x_1 + 5x_2 \leq 6
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\bullet & \quad C_1 \implies x_1 = 1 \\
\bullet & \quad C_2 \implies x_2 = 0
\end{align*}
\]

A solution has been found!
How does it works?

- If the propagation phase does not succeed in neither finding a solution nor proving that there is no solution, then a search tree is developed.
- It is based on a partition of the feasible set.
CP for Optimization

- \( \min \{ f(x)/C_j(x) \text{ is true } j = 1, \ldots, m \} \quad x_i \in D_i \quad i = 1, \ldots, n \)
Constraint Programming Basic Principles
Modelling with CP
Comparing Mathematical Programming and Constraint Programming
Common points and differences
Cooperative Schemes
Application to the Multi-Knapsack Problem
Conclusion

CP for Optimization

- min\{f(x)/C_j(x) \text{ is true } j = 1, \ldots, m \text{ } x_i \in D_i \text{ } i = 1, \ldots, n\}

\begin{align*}
\min & \quad Z \\
\text{s.t.} & \quad C_j(x) \text{ is true } j = 1, \ldots, m \\
& \quad Z = f(x) \\
& \quad x_i \in D_i \text{ } i = 1, \ldots, n \\
& \quad Z \in [lb, ub]
\end{align*}
CP for Optimization

\[
\begin{align*}
\min \{ f(x) &\mid C_j(x) \text{ is true } j = 1, \ldots, m \} \\
\text{s.t.} & \quad x_i \in D_i \quad i = 1, \ldots, n \\
& \quad Z = f(x) \\
& \quad Z \in [lb, ub]
\end{align*}
\]
CP for Optimization

- \( \min \{ f(x) / C_j(x) \text{ is true } j = 1, \ldots, m \} \quad x_i \in D_i \quad i = 1, \ldots, n \)

\[
\begin{align*}
\min & \quad Z \\
\text{s.t.} & \quad C_j(x) \text{ is true } \quad j = 1, \ldots, m \\
& \quad Z = f(x) \\
& \quad x_i \in D_i \quad i = 1, \ldots, n \\
& \quad Z \in [lb, ub]
\end{align*}
\]

- A filtering algorithm has to be designed for the objective function.
CP for Optimization

\[ \min \{ f(x) / C_j(x) \text{ is true } j = 1, \ldots, m \} x_i \in D_i \ i = 1, \ldots, n \]

\[
\begin{aligned}
\min & \quad Z \\
\text{s.t.} & \quad C_j(x) \text{ is true } j = 1, \ldots, m \\
& \quad Z = f(x) \\
& \quad x_i \in D_i \quad i = 1, \ldots, n \\
& \quad Z \in [lb, ub]
\end{aligned}
\]

- A filtering algorithm has to be designed for the objective function.
- It generally aims to remove values from:
CP for Optimization

- \( \min \{ f(x)/C_j(x) \text{ is true } j = 1, \ldots, m \, x_i \in D_i \, i = 1, \ldots, n \} \)

\[
\begin{align*}
\min & \quad Z \\
\text{s.t.} & \quad C_j(x) \text{ is true } j = 1, \ldots, m \\
& \quad Z = f(x) \\
& \quad x_i \in D_i \quad i = 1, \ldots, n \\
& \quad Z \in [lb, ub]
\end{align*}
\]

- A filtering algorithm has to be designed for the objective function.
- It generally aims to remove values from:
  - The domain of \( Z \) by reducing the bounds.
CP for Optimization

- \( \min \{ f(x) / C_j(x) \text{ is true } j = 1, \ldots, m \} \) \( x_i \in D_i \text{ } i = 1, \ldots, n \)

\[
\begin{align*}
\min & \quad Z \\
\text{s.t.} & \quad C_j(x) \text{ is true } j = 1, \ldots, m \\
& \quad Z = f(x) \\
& \quad x_i \in D_i \quad i = 1, \ldots, n \\
& \quad Z \in [lb, ub]
\end{align*}
\]

- A filtering algorithm has to be designed for the objective function.
- It generally aims to remove values from:
  - The domain of \( Z \) by reducing the bounds.
  - \( D_i \) by removing values which are not consistent with the current upper and lower bounds.
Filtering Algorithm

- Input: Constraint data and domains of the variables.
- Output: List of variables whose domain has been reduced (and their new domain).
Filtering Algorithm

- Input: Constraint data and domains of the variables.
- Output: List of variables whose domain has been reduced (and their new domain).
- Example: knapsack constraint
  \[ \sum_{i=1}^{n} a_i x_i \leq b \]
Filtering Algorithm

- Input: Constraint data and domains of the variables.
- Output: List of variables whose domain has been reduced (and their new domain).
- Example: knapsack constraint
  \[ \sum_{i=1}^{n} a_i x_i \leq b \]
  \[ x_i \in [l_i, u_i] \] and integer.
Filtering Algorithm Example

- $a_1 x_1 \leq b - \sum_{i=2}^{n} a_i x_i$
Filtering Algorithm Example

- \( a_1 x_1 \leq b - \sum_{i=2}^{n} a_i x_i \)

- If \( a_1 \geq 0 \) then \( x_1 \leq \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} \)
Filtering Algorithm Example

- \( a_1 x_1 \leq b - \sum_{i=2}^{n} a_i x_i \)

- If \( a_1 \geq 0 \) then \( x_1 \leq \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} \)

- If \( a_1 \geq 0 \) then \( x_1 \leq \max\{\frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x_i \in [l_i, u_i]\} \)
Filtering Algorithm Example

- \( a_1 x_1 \leq b - \sum_{i=2}^{n} a_i x_i \)

- If \( a_1 \geq 0 \) then \( x_1 \leq \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} \)

- If \( a_1 \geq 0 \) then \( x_1 \leq \max\left\{ \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} \right\} \) for all \( x_i \in [l_i, u_i] \)

- If \( a_1 \geq 0 \) then \( x_1 \leq \left\lfloor \frac{b - \sum_{a_i \leq 0} a_i u_i - \sum_{a_i \geq 0} a_i l_i}{a_1} \right\rfloor = M \)
Constraint Programming Basic Principles
Modelling with CP
Comparing Mathematical Programming and Constraint Programming
Common points and differences
Cooperative Schemes
Application to the Multi-Knapsack Problem
Conclusion

Filtering Algorithm Example

- \( a_1 x_1 \leq b - \sum_{i=2}^{n} a_i x_i \)
- If \( a_1 \geq 0 \) then \( x_1 \leq \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} \)
- If \( a_1 \geq 0 \) then \( x_1 \leq \max\{ \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x_i \in [l_i, u_i]\} \)
- If \( a_1 \geq 0 \) then \( x_1 \leq \left\lfloor \frac{b - \sum_{a_i \leq 0} a_i u_i - \sum_{a_i \geq 0} a_i l_i}{a_1} \right\rfloor = M \)
- If \( a_1 \geq 0 \) and \( M < u_1 \) then \( u_1 = M \)
Filtering Algorithm Example

- We can do better by solving \( \max \{ \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x \text{ feasible} \} \)
  or any relaxation ...
Filtering Algorithm Example

- We can do better by solving \( \max \left\{ \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x \text{ feasible} \right\} \) or any relaxation ...
- It happens that more than one filtering algorithm are associated to the same constraint and are successively called if they make different deductions ...
Filtering Algorithm Example

- We can do better by solving $\max \left\{ \frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x \text{ feasible} \right\}$ or any relaxation ...

- It happens that more than one filtering algorithm are associated to the same constraint and are successively called if they make different deductions ...

- It also happens that different versions of the same algorithm, more or less powerful but more or less expensive to run, are considered and called appropriately according to some criteria.
Propagation Algorithm: principle

- \( \mathcal{L} \): a list of constraints to be filtered.
Propagation Algorithm: principle

- $\mathcal{L}$: a list of constraints to be filtered.
- At the root of the search tree, $\mathcal{L}$ is initialised with all the constraints of the problem.
Propagation Algorithm: principle

- \( \mathcal{L} \): a list of constraints to be filtered.
- At the root of the search tree, \( \mathcal{L} \) is initialised with all the constraints of problem.
- At any other node, \( \mathcal{L} \) is initialised with the constraint that has been added to the problem for branching (i.e. \( x = 1, x \leq y \), etc.) which is now a constraint of the problem.
Propagation Algorithm: principle

- $\mathcal{L}$: a list of constraints to be filtered.
- At the root of the search tree, $\mathcal{L}$ is initialised with all the constraints of problem.
- At any other node, $\mathcal{L}$ is initialised with the constraint that has been added to the problem for branching (i.e. $x = 1$, $x \leq y$, etc.) which is now a constraint of the problem.
- Choose a constraint $C$ in $\mathcal{L}$
Propagation Algorithm: principle

- $\mathcal{L}$: a list of constraints to be filtered.
- At the root of the search tree, $\mathcal{L}$ is initialised with all the constraints of problem.
- At any other node, $\mathcal{L}$ is initialised with the constraint that has been added to the problem for branching (i.e. $x = 1$, $x \leq y$, etc.) which is now a constraint of the problem.
- Choose a constraint $C$ in $\mathcal{L}$
- Call the Filtering Algorithm of $C$
Propagation Algorithm: principle

- \( \mathcal{L} \): a list of constraints to be filtered.
- At the root of the search tree, \( \mathcal{L} \) is initialised with all the constraints of the problem.
- At any other node, \( \mathcal{L} \) is initialised with the constraint that has been added to the problem for branching (i.e. \( x = 1, x \leq y \), etc.) which is now a constraint of the problem.
- Choose a constraint \( C \) in \( \mathcal{L} \)
- Call the Filtering Algorithm of \( C \)
- For each variable \( v \) whose domain has been reduced:
Propagation Algorithm: principle

- $\mathcal{L}$: a list of constraints to be filtered.
- At the root of the search tree, $\mathcal{L}$ is initialised with all the constraints of problem.
- At any other node, $\mathcal{L}$ is initialised with the constraint that has been added to the problem for branching (i.e. $x = 1$, $x \leq y$, etc.) which is now a constraint of the problem.
- Choose a constraint $C$ in $\mathcal{L}$
- Call the Filtering Algorithm of $C$
- For each variable $v$ whose domain has been reduced:
  - Add to $\mathcal{L}$, all the constraints where $v$ is involved and which are not already in $\mathcal{L}$. 

Philippe Michelon, Université d'Avignon
Available tools

- **CP-Optimizer** from IBM (ex-ILOG), works like CPLEX (CONCERT etc.). Very expensive but free for academics!

  [CP-Optimizer](http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/)

- **CHOCO**, open-source.

  [CHOCO](http://www.emn.fr/z-info/choco-solver/)
Since CP does not use classical Mathematical Programming tools, it can be applied on any kind of model: no need for linearity, nor convexity, etc.
Modelling with CP

- Since CP does not use classical Mathematical Programming tools, it can be applied on any kind of model: no need for linearity, nor convexity, etc.
- It generally induces models which are more compact and more natural.
Modelling with CP

- Since CP does not use classical Mathematical Programming tools, it can be applied on any kind of model: no need for linearity, nor convexity, etc.
- It generally induces models which are more compact and more natural.
- Constraints might include disjunctions, implications, etc.
Modelling with CP

- Since CP does not use classical Mathematical Programming tools, it can be applied on any kind of model: no need for linearity, nor convexity, etc.
- It generally induces models which are more compact and more natural.
- Constraints might include disjunctions, implications, etc.
- Variables can be of any type (real, integer, set, object of any type)
Global Constraints

- CP is a local approach: it considers only one constraint at a time.
Global Constraints

- **CP is a local approach**: it considers only one constraint at a time.

- **Filtering algorithms** taking into account a whole set of constraints have been developed.
Global Constraints

- **CP is a local approach**: it considers only one constraint at a time.
- Filtering algorithms taking into account a whole set of constraints have been developed.
- **The whole set of constraints is then contracted in a unique constraint, called Global Constraint.**
The AllDifferent Global Constraints

- **AllDifferent** should be used when $n$ variables have to take a different value.

  $\text{AllDifferent}(x) \iff x_i \neq x_j \quad \forall 1 \leq i < j \leq n$

- J-Ch Regin (94) designed a very efficient filtering algorithm for this global constraint.
The AllDifferent Global Constraints

- **AllDifferent** should be used when \( n \) variables have to take a different value.

\[
\text{AllDifferent}(x) \iff x_i \neq x_j \quad \forall 1 \leq i < j \leq n
\]

- J-Ch Regin (94) designed a very efficient filtering algorithm for this global constraint.

- Example:
  \( x_1 \in \{1, 2, 3\} \quad x_2 \in \{1, 2\} \quad x_3 \in \{1, 2, 3, 4\} \quad x_4 \in \{1, 2\} \)
The AllDifferent Global Constraints

- **AllDifferent** should be used when \( n \) variables have to take a different value.

- **AllDifferent**\((x) \iff x_i \neq x_j \quad \forall 1 \leq i < j \leq n\)

- J-Ch Regin (94) designed a very efficient filtering algorithm for this global constraint.

- Example:
  \( x_1 \in \{1, 2, 3\} \quad x_2 \in \{1, 2\} \quad x_3 \in \{1, 2, 3, 4\} \quad x_4 \in \{1, 2\} \)

  \textit{AllDifferent}(x) \implies x_1 = 3 \; x_3 = 4.
Given a list of cities and their pairwise distances, the Travelling Salesman Problem consists in finding a shortest possible tour that visits each city exactly once.
Modelling with CP: the TSP

- Given a list of cities and their pairwise distances, the Travelling Salesman Problem consists in finding a shortest possible tour that visits each city exactly once.

Modelling with CP: the TSP

- Given a list of cities and their pairwise distances, the Travelling Salesman Problem consists in finding a shortest possible tour that visits each city exactly once.
- NP-Hard problem.
- Mathematical Programming models involve variables $x_{ij} = 1$ if the tour goes from city $i$ to city $j$ directly.
Given a list of cities and their pairwise distances, the Travelling Salesman Problem consists in finding a shortest possible tour that visits each city exactly once.

Np-Hard problem.

Mathematical Programming models involve variables $x_{ij} = 1$ if the tour goes from city $i$ to city $j$ directly.

and ... an exponential number of constraints to avoid subcycles.
Modelling with CP: the TSP

- The problem might be seen as finding a permutation.
Modelling with CP: the TSP

- The problem might be seen as finding a permutation.
- The CP tools allows to model the TSP with permutations!
The problem might be seen as finding a permutation.
The CP tools allows to model the TSP with permutations!
Let $p_i$ be the city in $i$-th position in the tour.
The problem might be seen as finding a permutation.
The CP tools allows to model the TSP with permutations!
Let $p_i$ be the city in $i$-th position in the tour.

\[
\begin{align*}
\min & \quad \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1} \\
\text{s.t.} & \quad \text{AllDifferent}(p) \text{ is true.} \\
 & \quad p_i \in \{1, \ldots, n\} \ \forall \ i \in \{1, \ldots, n\}
\end{align*}
\]
Modelling with CP: the TSP

- The problem might be seen as finding a permutation.
- The CP tools allows to model the TSP with permutations!
- Let $p_i$ be the city in $i$-th position in the tour.

\[
\begin{align*}
\text{min} & \quad \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1} \\
\text{s.t.} & \quad \text{AllDifferent}(p) \text{ is true.} \\
& \quad p_i \in \{1, \ldots, n\} \ \forall i \in \{1, \ldots, n\}
\end{align*}
\]

- Only one constraint!
The problem might be seen as finding a permutation.
The CP tools allows to model the TSP with permutations!
Let $p_i$ be the city in $i$-th position in the tour.

$$\min \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1}$$

s.t. \quad \text{AllDifferent}(p) \text{ is true.}

$$p_i \in \{1, \ldots, n\} \ \forall i \in \{1, \ldots, n\}$$

Only one constraint!
That does not make the problem easier!
Modelling with CP: the TSP

- Basic CP tools include a filtering algorithm for
  \[ Z = \sum_{i=1}^{n-1} D_{p_i, p_{i+1}} + D_{p_n, p_1}. \]
Modelling with CP: the TSP

- Basic CP tools include a filtering algorithm for
  \[ Z = \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1}. \]
- Some authors (Caseaux-Laburthe 97, Focacci-Lodi-Milano 99) strengthen it by:

---

Philippe Michelon, Université d’Avignon
Basic CP tools include a filtering algorithm for
\[ Z = \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1}. \]

Some authors (Caseaux-Laburthe 97, Focacci-Lodi-Milano 99) strengthen it by:
- Computing the Linear Assignment lower bound
Modelling with CP: the TSP

- Basic CP tools include a filtering algorithm for
  \[ Z = \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1}. \]
- Some authors (Caseaux-Laburthe 97, Focacci-Lodi-Milano 99) strengthen it by:
  - Computing the Linear Assignment lower bound
  - Fixing some variables by using the reduced costs
Modelling with CP: the TSP

- Basic CP tools include a filtering algorithm for
  \[ Z = \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1}. \]
- Some authors (Caseaux-Laburthe 97, Focacci-Lodi-Milano 99) strengthen it by:
  - Computing the Linear Assignment lower bound
  - Fixing some variables by using the reduced costs
Modelling with CP: the TSP

- Basic CP tools include a filtering algorithm for
  \[ Z = \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1}. \]

- Some authors (Caseaux-Laburthe 97, Focacci-Lodi-Milano 99) strengthen it by:
  - Computing the Linear Assignment lower bound
  - Fixing some variables by using the reduced costs

This is a cooperative scheme !!!
Modelling with CP: the VRPTW

- The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem seeking to service a number of customers with a fleet of vehicles.
The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem seeking to service a number of customers with a fleet of vehicles.

Proposed by Dantzig and Ramser in 1959, VRP is an important problem in the fields of transportation, distribution and logistics.
The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem seeking to service a number of customers with a fleet of vehicles.

Proposed by Dantzig and Ramser in 1959, VRP is an important problem in the fields of transportation, distribution and logistics.

The context is that of delivering goods located at a central depot to customers who have placed orders for such goods with the goal of minimizing the cost of distributing the goods.
The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem seeking to service a number of customers with a fleet of vehicles.

Proposed by Dantzig and Ramser in 1959, VRP is an important problem in the fields of transportation, distribution and logistics.

The context is that of delivering goods located at a central depot to customers who have placed orders for such goods with the goal of minimizing the cost of distributing the goods.

Sometimes, the goods can be only delivered in a time window: this is the VPRTW.
The vehicle routing problem (VRP) is a combinatorial optimization and integer programming problem seeking to service a number of customers with a fleet of vehicles.

Proposed by Dantzig and Ramser in 1959, VRP is an important problem in the fields of transportation, distribution and logistics.

The context is that of delivering goods located at a central depot to customers who have placed orders for such goods with the goal of minimizing the cost of distributing the goods.

Sometimes, the goods can be only delivered in a time window: this is the VPRTW.

According to the context, additional constraints can be considered such as vehicle capacities.
Modelling with CP : the VRPTW

- data: $n$ customers, $C_{ij}$: travel cost from $i$ to $j$, $T_{ij}$: travel time from $i$ to $j$, $[a_i, b_i]$: time window of $i$, $r_i$: demand of $i$, $K$: capacity of the vehicles, $v$: number of vehicles.
Modelling with CP: the VRPTW

- data: \( n \) customers, \( C_{ij} \): travel cost from \( i \) to \( j \), \( T_{ij} \): travel time from \( i \) to \( j \), \([a_i, b_i]\): time window of \( i \), \( r_i \): demand of \( i \), \( K \): capacity of the vehicles, \( v \): number of vehicles.

- All the vehicles are used and a customer is served by exactly one vehicle.
Modelling with CP: the VRPTW

- data: \( n \) customers, \( C_{ij} \): travel cost from \( i \) to \( j \), \( T_{ij} \): travel time from \( i \) to \( j \), \([a_i, b_i]\): time window of \( i \), \( r_i \): demand of \( i \), \( K \): capacity of the vehicles, \( v \): number of vehicles.
- All the vehicles are used and a customer is served by exactly one vehicle.
- A vehicle can arrive before the time window, but it will have to wait.
Modelling with CP : the VRPTW

- Decision variables: $S_i$: customer visited after $i$, $t_i$: time delivery of $i$, $l_i$: load of vehicle serving $i$. 
Modelling with CP: the VRPTW

- Decision variables: $S_i$: customer visited after $i$, $t_i$: time delivery of $i$, $l_i$: load of vehicle serving $i$.

\[
\begin{align*}
\min & \quad \sum_{i=1}^{n} C_i S_i \\
\text{s.t.} & \quad \text{AllDifferent}(S) \\
& \quad S_i = j \Rightarrow l_j = l_i - r_i \quad \forall i, j \\
& \quad S_i = j \Rightarrow t_j \geq t_i + T_{ij} \quad \forall i, j \\
& \quad S_i \in \{1, \ldots, n\}/\{i\} \quad \forall i \\
& \quad l_i \in [0, K] \quad \forall i \\
& \quad t_i \in [a_i, b_i] \quad \forall i
\end{align*}
\]
Modelling with CP: the VRPTW

Such a model has been used for the design of heuristics for the VRPTW by:

- Caseau-Laburthe 99
- Feillet et al 01
- Pesant et al 98
- Rousseau 02
The Cumulative Global Constraint

- Scheduling problems consists in finding when (and, sometimes, where) a set of activities has to be executed.
Scheduling problems consists in finding when (and, sometimes, where) a set of activities has to be executed.

Each activity \(i\) requires a quantity \(r_{i,k}\) of a given resource \(k\) which cannot provide more than \(R_k\) units.
The Cumulative Global Constraint

- **Scheduling problems** consist in finding when (and, sometimes, where) a set of activities has to be executed.

- Each activity $i$ requires a quantity $r_{i,k}$ of a given resource $k$ which cannot provide more than $R_k$ units.

- Hence, at each time of the planning period, the set of activities currently performed must respect the capacity of each of the resources.
This set of constraint is not so easy to express in a linear way.
The Cumulative Global Constraint

- This set of constraints is not so easy to express in a linear way.
- The Cumulative Constraint, for which several different and complementary filtering algorithms have been developed, models this set of constraints. Among others, the filtering algorithms are:
The Cumulative Global Constraint

- This set of constraint is not so easy to express in a linear way.
- The Cumulative Constraint, for which several different and complementary filtering algorithm have been developed, models this set of constraints. Among others, the filtering algorithms are:
  - The time table constraint (folklore)
The Cumulative Global Constraint

- This set of constraints is not so easy to express in a linear way.
- The Cumulative Constraint, for which several different and complementary filtering algorithms have been developed, models this set of constraints. Among others, the filtering algorithms are:
  - The time table constraint (folklore)
  - The energetic rules (Lopez 97, Baptiste-LePape 00)
The Cumulative Global Constraint

- This set of constraint is not so easy to express in a linear way.
- The Cumulative Constraint, for which several different and complementary filtering algorithm have been developed, models this set of constraints. Among others, the filtering algorithms are:
  - The time table constraint (folklore)
  - The energetic rules (Lopez 97, Baptiste-LePape 00)
  - Edge Finding (Carlier-Pinson 90)
The Cumulative Global Constraint

- Given a set of $n$ activities and a resource of capacity $R$, let:
The Cumulative Global Constraint

- Given a set of $n$ activities and a resource of capacity $R$, let:
  
  - $r = (r_1, r_2, \ldots, r_n)$ the vector of demands
The Cumulative Global Constraint

- Given a set of $n$ activities and a resource of capacity $R$, let:
  - $r = (r_1, r_2, \ldots, r_n)$ the vector of demands
  - $d = (d_1, d_2, \ldots, d_n)$ the vector of durations
The Cumulative Global Constraint

- Given a set of $n$ activities and a resource of capacity $R$, let:
  - $r = (r_1, r_2, \ldots, r_n)$ the vector of demands
  - $d = (d_1, d_2, \ldots, d_n)$ the vector of durations
  - $S = (S_1, S_2, \ldots, S_n)$ the vector of starting times
The Cumulative Global Constraint

- Given a set of $n$ activities and a resource of capacity $R$, let:
  - $r = (r_1, r_2, \ldots, r_n)$ the vector of demands
  - $d = (d_1, d_2, \ldots, d_n)$ the vector of durations
  - $S = (S_1, S_2, \ldots, S_n)$ the vector of starting times

- $Cumulative(S, d, r, R) \iff \sum_{\{i \mid S_i \leq t < S_i + d_i\}} r_i \leq R \forall t$
The Resources Constrained Project Scheduling Problem deals with scheduling a project of $n$ activities which are linked by precedence constraints.
The RCPSP

- The Resources Constrained Project Scheduling Problem deals with scheduling a project of \( n \) activities which are linked by precedence constraints.

- Each activity requires a given quantity of \( K \) resources available in a limited quantity.
The Resources Constrained Project Scheduling Problem deals with scheduling a project of $n$ activities which are linked by precedence constraints.

Each activity requires a given quantity of $K$ resources available in a limited quantity.

The resources are renewable.
The Resources Constrained Project Scheduling Problem deals with scheduling a project of $n$ activities which are linked by precedence constraints.

Each activity requires a given quantity of $K$ resources available in a limited quantity.

The resources are renewable.

The activities are not preemptive.
The Resources Constrained Project Scheduling Problem deals with scheduling a project of $n$ activities which are linked by precedence constraints.

Each activity requires a given quantity of $K$ resources available in a limited quantity.

The resources are renewable.

The activities are not preemptive.

The objective is to minimize the makespan of the project.
Modelling with CP: the RCPSP

- 5 activities
- 1 resource of capacity 4

Task duration

makespan = 9

resource consumption
Modelling with CP: the RCPSP

- $K$ resources of cap. $R_k$.
- $n$ activities of duration $d_i$ and resource consumption $r_{ik}$.
- Precedence constraints $G=(V,E)$.
- Decision variables: $S_i$, Starting time of $i$, $C_{max}(=S_{n+1})$, total duration of the project.
Modelling with CP: the RCPSP

- $K$ resources of cap. $R_k$.
- $n$ activities of duration $d_i$ and resource consumption $r_{ik}$.
- Precedence constraints $G=(V,E)$.
- Decision variables: $S_i$, Starting time of $i$, $C_{max}(=S_{n+1})$, total duration of the project.

\[
\begin{align*}
\min \quad & C_{max} \\
\text{s.t.:} \quad & C_{max} \leq S_i + d_i \quad \forall i \\
& S_j \geq S_i + d_i \quad \forall (i,j) \in E \\
& \text{Cumulative}(S, d, r_k, R_k) \quad \forall k \\
& S \geq 0
\end{align*}
\]
Solving the RCPSP with a pure CP approach

- The filtering algorithm associated with the precedence constraints is a longest path algorithm.
- Many authors solved the RCPSP with a pure CP approach:
  - Klein-Scholl 99
  - Baptiste-Le Pape 00
  - Liess-Michelon 03
  - Etc.
Joint work with R. Acuña-Agost, S. Gueye, and D. Feillet.


Deals with the impact of disruptions in order to ensure the best possible service for the passengers.
The Rescheduling Problem aims to restore an original plan made infeasible by some incidents.

- It is to be solved in (almost) real time.
The Rescheduling Problem aims to restore an original plan made infeasible by some incidents.

- It is to be solved in (almost) real time.
- It considers both Timetabling and Platforms Assignment Problems.
The Railway Rescheduling Problem

The Rescheduling Problem aims to restore an original plan made infeasible by some incidents.

- It is to be solved in (almost) real time.
- It considers both Timetabling and Platforms Assignment Problems.
- **Input**: Original Plan and the incidents (with the associated delays).
The Railway Rescheduling Problem: Constraints

- Assignment constraints.
The Railway Rescheduling Problem: Constraints

- Assignment constraints.

- Traveling and stopping constraints.
The Railway Rescheduling Problem: Constraints

- Assignment constraints.
- Traveling and stopping constraints.
- Utilization of tracks and platforms constraints.
The Railway Rescheduling Problem: Constraints

- Assignment constraints.
- Traveling and stopping constraints.
- Utilization of tracks and platforms constraints.
- **Scheduling constraints.**
The Railway Rescheduling Problem: Constraints

- Assignment constraints.
- Traveling and stopping constraints.
- Utilization of tracks and platforms constraints.
- Scheduling constraints.
- Connections of trains constraints.
The Railway Rescheduling Problem: Constraints

- Assignment constraints.
- Traveling and stopping constraints.
- Utilization of tracks and platforms constraints.
- Scheduling constraints.
- Connections of trains constraints.
- Unplanned stops constraints
The Railway Rescheduling Problem: Objective Function

- Minimization of the overall delay.
The Railway Rescheduling Problem: Objective Function

- Minimization of the overall delay.
- Minimization of the changes in assignments.
The Railway Rescheduling Problem: Objective Function

- Minimization of the overall delay.
- Minimization of the changes in assignments.
- Minimization of unplanned stops.
The Railway Rescheduling Problem: Objective Function

- Minimization of the overall delay.
- Minimization of the changes in assignments.
- Minimization of unplanned stops.
The Railway Rescheduling Problem: Objective Function

- Minimization of the overall delay.
- Minimization of the changes in assignments.
- Minimization of unplanned stops.

These objectives are reduced to only one objective by a weighted linear combination.
Why studying a cooperative scheme for RRP?

- Mathematical Programming based approaches are efficient ...
Why studying a cooperative scheme for RRP?

- Mathematical Programming based approaches are efficient ...
- for small or medium size instances (horizon $\leq 7\text{h}00$ for the french network).
Why studying a cooperative scheme for RRP?

- Mathematical Programming based approaches are efficient...
- for small or medium size instances (horizon ≤ 7h00 for the french network).
- They are limited by the memory they require (150 000 variables and 350 000 constraints for a 6h00 horizon time).
Why studying a cooperative scheme for RRP?

- Mathematical Programming based approaches are efficient ...
- for small or medium size instances (horizon ≤ 7h00 for the french network).
- They are limited by the memory they require (150 000 variables and 350 000 constraints for a 6h00 horizon time).
- Since Constraints Programming Models are "smaller", it may be usefull to try to use CP ...
Decision Variables for the Mathematical Programming Model

- $x_k^{\text{begin}} \geq 0$ start time of event $k$
Decision Variables for the Mathematical Programming Model

- $x_k^{\text{begin}} \geq 0$ start time of event $k$
- $x_k^{\text{end}} \geq 0$ end time of event $k$
Decision Variables for the Mathematical Programming Model

- $x_k^{\text{begin}} \geq 0$ start time of event $k$
- $x_k^{\text{end}} \geq 0$ end time of event $k$
- $z_k \geq 0$ delay of event $k$

The Railway Rescheduling Problem

Constraint Programming Basic Principles
Modelling with CP
Comparing Mathematical Programming and Constraint Programming
Common points and differences
Cooperative Schemes
Application to the Multi-Knapsack Problem
Conclusion
Decision Variables for the Mathematical Programming Model

- $x_k^{\text{begin}} \geq 0$ start time of event $k$
- $x_k^{\text{end}} \geq 0$ end time of event $k$
- $z_k \geq 0$ delay of event $k$
- $q_{kt} \in \{0, 1\} = 1$ if event $k$ uses track $t$
Decision Variables for the Mathematical Programming Model

- $x_k^{\text{begin}} \geq 0$ start time of event $k$
- $x_k^{\text{end}} \geq 0$ end time of event $k$
- $z_k \geq 0$ delay of event $k$
- $q_{kt} \in \{0, 1\} = 1$ if event $k$ uses track $t$
- $y_k = 1$ if event $k$ is an unplanned stop
Decision Variables for the Mathematical Programming Model

- $x_k^{\text{begin}} \geq 0$ start time of event $k$
- $x_k^{\text{end}} \geq 0$ end time of event $k$
- $z_k \geq 0$ delay of event $k$
- $q_{kt} \in \{0, 1\} = 1$ if event $k$ uses track $t$
- $y_k = 1$ if event $k$ is an unplanned stop
- $\gamma_k \hat{k} = 1$ if event $k$ occurs before event $\hat{k}$ as planned originally.
Decision Variables for the Mathematical Programming Model

- $x_k^{\text{begin}} \geq 0$ start time of event $k$
- $x_k^{\text{end}} \geq 0$ end time of event $k$
- $z_k \geq 0$ delay of event $k$
- $q_{kt} \in \{0, 1\} = 1$ if event $k$ uses track $t$
- $y_k = 1$ if event $k$ is an unplanned stop
- $\gamma_k \hat{k} = 1$ if event $k$ occurs before event $\hat{k}$ as planned originally.
- $\lambda_k \hat{k} = 1$ if event $k$ occurs after event $\hat{k}$, not as planned originally.
Constraint Programming Decisions Variables

- $Delay_{in}$: Delay of train $i$ at node $n$.
- $Start_{in}$: Arrival time of train $i$ at node $n$.
- $Track_{in}$: Track used by train $i$ at node $n$.
- $UP_{in}$: 1, if train $i$ performs an unplanned stop in the node $n$; 0 otherwise.
Comparison of the Models

The Railway Rescheduling Problem

Philippe Michelon, Université d'Avignon
Comparison of the Models

The Railway Rescheduling Problem

Constraint Programming Basic Principles
Modelling with CP
Comparing Mathematical Programming and Constraint Programming
Common points and differences
Cooperative Schemes
Application to the Multi-Knapsack Problem
Conclusion
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
- They are using quite different models and tools.
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
- They are using quite different models and tools.
- CP is a "local" method while MP is "global".
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
- They are using quite different models and tools.
- CP is a "local" method while MP is "global".
- CP cannot be guided by an approximate solution.
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
- They are using quite different models and tools.
- CP is a “local” method while MP is ”global”.
- CP cannot be guided by an approximate solution.
- CP fully takes into account the domains of the variables while MP does not.
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
- They are using quite different models and tools.
- CP is a “local” method while MP is “global”.
- CP cannot be guided by an approximate solution.
- CP fully takes into account the domains of the variables while MP does not.
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
- They are using quite different models and tools.
- CP is a "local" method while MP is "global".
- CP cannot be guided by an approximate solution.
- CP fully takes into account the domains of the variables while MP does not.

They are complementary methods!
Comparing Mathematical Programming and Constraint Programming

- Both methods use search trees.
- They are using quite different models and tools.
- CP is a ”local” method while MP is ”global”.
- CP cannot be guided by an approximate solution.
- CP fully takes into account the domains of the variables while MP does not.

- They are complementary methods !
- Cooperative schemes should be useful !
The first cooperative schemes to appear in the literature were based on Mathematical Programming decomposition.

Benoit et al 02 solved with a pure CP approach the master problem in a Bender’s decomposition approach.

Rousseau et al 02 solved with a pure CP approach the sub problem in a column generation method.
Cooperative Schemes

Perform the CP propagation

Compute the MP bound
The Reduced Cost Constraint

\[
\begin{align*}
\min & \quad Z = \sum_{i=1}^{n} c_i x_i \\
\text{s.t.} & \quad \sum_{i=1}^{n} a_{j,i} x_i + s_j = b_j \\
& \quad x \in \{0, 1\}^n \\
& \quad s \geq 0
\end{align*}
\]
The Reduced Cost Constraint

\[ \min Z = \sum_{i=1}^{n} c_i x_i \]
\[ \text{s.t.} \quad \sum_{i=1}^{n} a_{j,i} x_i + s_j = b_j \]
\[ x \in \{0, 1\}^n \]
\[ s \geq 0 \]

At the optimum of the linear relaxation, the objective function is expressed as:

\[ Z = lb + \sum_{i \in N_0} \bar{c}_i x_i + \sum_{i \in N_1} \bar{c}_i (1 - x_i) + \sum_j \bar{d}_j s_j \]
If a feasible solution is known, we are not interested in solutions worst than this solution:

\[ \text{lb} + \sum_{i \in N_0} \bar{c}_i x_i + \sum_{i \in N_1} \bar{c}_i (1 - x_i) + \sum_j \bar{d}_j s_j \leq \text{ub} \]
The Reduced Cost Constraint

- If a feasible solution is known, we are not interested in solutions worst than this solution:

\[ lb + \sum_{i \in N_0} \bar{c}_i x_i + \sum_{i \in N_1} \bar{c}_i (1 - x_i) + \sum_j \bar{d}_j s_j \leq ub \]

- This is the so called Reduced Cost Constraint.
If a feasible solution is known, we are not interested in solutions worst than this solution:

\[ lb + \sum_{i \in N_0} \bar{c}_i x_i + \sum_{i \in N_1} \bar{c}_i (1 - x_i) + \sum_j \bar{d}_j s_j \leq ub \]

This is the so called Reduced Cost Constraint.

By adding this constraint to the pool of constraints to be filtered, we can:
If a feasible solution is known, we are not interested in solutions worst than this solution:

\[ lb + \sum_{i \in \mathcal{N}_0} \bar{c}_i x_i + \sum_{i \in \mathcal{N}_1} \bar{c}_i (1 - x_i) + \sum_j d_j s_j \leq ub \]

- This is the so called Reduced Cost Constraint.
- By adding this constraint to the pool of constraints to be filtered, we can:
  - fix some variables.
The Reduced Cost Constraint

- If a feasible solution is known, we are not interested in solutions worst than this solution:

\[
lb + \sum_{i \in N_0} \bar{c}_i x_i + \sum_{i \in N_1} \bar{c}_i (1 - x_i) + \sum_j \bar{d}_j s_j \leq ub
\]

- This is the so-called Reduced Cost Constraint.
- By adding this constraint to the pool of constraints to be filtered, we can:
  - fix some variables.
  - find some upper bounds on the slack variables.
Filtering algorithm:

- \( s_j \leq \frac{ub - lb}{d_j} \)
The Reduced Cost Constraint

Filtering algorithm:

1. \( s_j \leq \frac{ub - lb}{d_j} \)
2. For \( i \in N_1, \ ub - lb \leq c_i \implies x_i = 1 \)
The Reduced Cost Constraint

Filtering algorithm:

- \( s_j \leq \frac{ub - lb}{d_j} \)
- For \( i \in N_1, \) \( ub - lb \leq \overline{c}_i \Rightarrow x_i = 1 \)
- For \( i \in N_0, \) \( ub - lb \leq \overline{c}_i \Rightarrow x_i = 0 \)
The Reduced Cost Constraint

Filtering algorithm:

- \( s_j \leq \frac{ub - lb}{d_j} \)
- For \( i \in N_1, \) \( ub - lb \leq c_i \rightarrow x_i = 1 \)
- For \( i \in N_0, \) \( ub - lb \leq c_i \rightarrow x_i = 0 \)

\( u_b \) over the slack variable allows to transform the initial constraint into:

\[ u_j \leq \sum_{i=1}^{n} a_{j,i} x_i \leq b_j \]
The Reduced Cost Constraint

Filtering algorithm:

- \( s_j \leq \frac{ub - lb}{d_j} \)
- For \( i \in N_1, \) \( ub - lb \leq \bar{c}_i \) \( \implies x_i = 1 \)
- For \( i \in N_0, \) \( ub - lb \leq \bar{c}_i \) \( \implies x_i = 0 \)

An upper bound \( u_j \) over the slack variable allows to transform the initial constraint into:

\[
u_j \leq \sum_{i=1}^{n} a_{j,i} x_i \leq b_j\]
The Multi-Knapsack Problem

- Generalisation of the classical knapsack Problem.

\[
\begin{align*}
\text{max} & \quad c^T x \\
\text{s.t.:} & \quad Ax \leq b, \\
& \quad x \in \{0, 1\}^n,
\end{align*}
\]
The Multi-Knapsack Problem

- Generalisation of the classical knapsack Problem.

\[
\begin{align*}
\text{max} & \quad c^T x \\
\text{s.t.:} & \quad Ax \leq b, \\
& \quad x \in \{0, 1\}^n,
\end{align*}
\]

- where \( A \geq 0^{m \times n} \), \( b \geq 0^m \), \( c \geq 0^n \)
The Multi-Knapsack Problem

- NP-hard problem
The Multi-Knapsack Problem

- NP-hard problem
- Widely studied (Fayard-Plateau 80, Freville-Plateau 94, James et al 05, Vasquez-Hao 01, Boussier et al 09, etc.).
The Multi-Knapsack Problem

- NP-hard problem
- Widely studied (Fayard-Plateau 80, Freville-Plateau 94, James et al 05, Vasquez-Hao 01, Boussier et al 09, etc.).
- No particular structure!
The Multi-Knapsack Problem

- NP-hard problem
- Widely studied (Fayard-Plateau 80, Freville-Plateau 94, James et al 05, Vasquez-Hao 01, Boussier et al 09, etc.).
- No particular structure!
- Joint work with C.D. Rodrigues and M. Campelo, published in the CP proceeding, 08.
Application to the Multi-Knapsack Problem

- Mathematical approaches for the (MKP) compute an upper bound by solving the linear relaxation.
Application to the Multi-Knapsack Problem

- Mathematical approaches for the (MKP) compute an upper bound by solving the linear relaxation.
- Vasquez-Hao introduce a cardinality constraint which strongly improve this bound!
Application to the Multi-Knapsack Problem

- Mathematical approaches for the (MKP) compute an upper bound by solving the linear relaxation.
- Vasquez-Hao introduce a cardinality constraint which strongly improve this bound!

\[
\begin{align*}
\max & \quad \sum_{i=1}^{n} x_i \\
\text{s.t.:} & \quad Ax \leq b, \\
& \quad c^t x \geq lb \\
& \quad x \in [0, 1]^n,
\end{align*}
\]
Application to the Multi-Knapsack Problem

- Mathematical approaches for the (MKP) compute an upper bound by solving the linear relaxation.
- Vasquez-Hao introduce a cardinality constraint which strongly improve this bound!

\[
\begin{align*}
\max (\text{resp } \min) & \quad \sum_{i=1}^{n} x_i \\
\text{s.t.: } & \quad Ax \leq b, \\
& \quad c^t x \geq lb \\
& \quad x \in [0, 1]^n,
\end{align*}
\]

- Let \( \bar{k} \) and \( k \) be the optimal values of the above problem.
Application to the Multi-Knapsack Problem

For \( k \in \{k, \ldots, \overline{k}\} \), create a subproblem :

\[
\begin{align*}
\text{max} & \quad c^T x \\
\text{s.t.} & \quad Ax \leq b, \\
& \quad \sum_{i=1}^{n} x_i = k \\
& \quad c^T x \geq lb \\
& \quad x \in \{0, 1\}^n,
\end{align*}
\]
Problem Definition
The KnapSum Constraint
Constraint aggregation

Application to the Multi-Knapsack Problem

For $k \in \{k, \ldots, \bar{k}\}$, create a subproblem:

\[
\begin{align*}
\text{max} & \quad c^T x \\
\text{s.t.:} & \quad Ax \leq b, \\
& \quad \sum_{i=1}^{n} x_i = k, \\
& \quad c^T x \geq lb, \\
& \quad x \in \{0, 1\}^n,
\end{align*}
\]

- Apply an Hybrid CP/MP method to each of them.
The KnapSum Constraint

- At each node of the search tree, the CP lawyer has to tackle constraints of the form: \( \sum_{i=1}^{n} x_i = k \) and \( \sum_{i=1}^{n} a_{ji} x_i \leq \bar{b}_j \)
At each node of the search tree, the CP lawyer has to tackle constraints of the form:
\[
\sum_{i=1}^{n} x_i = k \quad \text{and} \quad \sum_{i=1}^{n} a_{ji} x_i \leq \bar{b}_j
\]
- They can be grouped into Global Constraints!
The KnapSum Constraint

- At each node of the search tree, the CP lawyer has to tackle constraints of the form: \( \sum_{i=1}^{n} x_i = k \) and \( \sum_{i=1}^{n} a_{ji} x_i \leq \bar{b}_j \)
- They can be grouped into Global Constraints!

\[
\text{knapSum}(a_j, b_j, k, x) \iff b_i \leq \sum_{i=1}^{n} a_{ji} x_i \leq \bar{b}_i; \quad \sum_{j=1}^{n} x_j = k
\]
Let’s assume that for a constraint $\text{knapSum}(a_i, b_i, k, x)$ the coefficients $a_{ij}$ are ordered from the biggest to the smallest.
The KnapSum Constraint Filtering Algorithm

- Let’s assume that for a constraint $\text{knapSum}(a_i, b_i, k, x)$ the coefficients $a_{ij}$ are ordered from the biggest to the smallest.
- **Phase 1:** checking the feasibility.
The KnapSum Constraint Filtering Algorithm

Let’s assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coefficients $a_{ij}$ are ordered from the biggest to the smallest.

Phase 1: checking the feasibility.

Check $\tilde{S} = \sum_{j=1}^{k} a_{ij} \geq b_i$ and $S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i$ (1)
Let’s assume that for a constraint \( knapSum(a_i, b_i, k, x) \) the coefficients \( a_{ij} \) are ordered from the biggest to the smallest.

**Phase 1: checking the feasibility.**

\[
\text{Check } \bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i \quad \text{and} \quad S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \quad (1)
\]
The KnapSum Constraint Filtering Algorithm

- Let’s assume that for a constraint \( knapSum(a_i, b_i, k, x) \) the coefficients \( a_{ij} \) are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  - Check \( \bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i \) and \( S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \) (1)

- Phase 2: fixing variables
The KnapSum Constraint Filtering Algorithm

Let’s assume that for a constraint \( knapSum(a_i, b_i, k, x) \) the coefficients \( a_{ij} \) are ordered from the biggest to the smallest.

Phase 1: checking the feasibility.

- **Check** \( \bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i \) and \( S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \) (1)

Phase 2: fixing variables

- Set \( x_1 \) to 0, and check if \( \bar{S} < b_i \Rightarrow \) Fix \( x_1 = 1 \)
Let’s assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coefficients $a_{ij}$ are ordered from the biggest to the smallest.

Phase 1: checking the feasibility.

Let $\bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i$ and $S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i$.

Phase 2: fixing variables

- Set $x_1$ to 0, and check if $\bar{S} < \bar{b}_i \Rightarrow$ Fix $x_1 = 1$
The KnapSum Constraint Filtering Algorithm

- Let’s assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coefficients $a_{ij}$ are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  
  \[
  \text{Check } \bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i \quad \text{and} \quad S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \quad (1)
  \]

- Phase 2: fixing variables
  - Set $x_1$ to 0, and check if $\bar{S} < b_i \Rightarrow$ Fix $x_1 = 1$
  - Set $x_n$ to 0, and check if $S > \bar{b}_i \Rightarrow$ Fix $x_n = 1$
Problem Definition
The KnapSum Constraint
Constraint aggregation

The KnapSum Constraint Filtering Algorithm

- Let’s assume that for a constraint \( \text{knapSum}(a_i, b_i, k, x) \) the coefficients \( a_{ij} \) are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  
  \[
  \text{Check } \bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i \quad \text{and} \quad \bar{S} = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \tag{1}
  \]

- Phase 2: fixing variables
  - Set \( x_1 \) to 0, and check if \( \bar{S} < \bar{b}_i \) \( \Rightarrow \) Fix \( x_1 = 1 \)
  - Set \( x_n \) to 0, and check if \( \bar{S} > \bar{b}_i \) \( \Rightarrow \) Fix \( x_n = 1 \)
The KnapSum Constraint Filtering Algorithm

- Let’s assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coefficients $a_{ij}$ are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  - Check $\bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i$ and $S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i$ (1)
- Phase 2: fixing variables
  - Set $x_1$ to 0, and check if $\bar{S} < b_i \Rightarrow$ Fix $x_1 = 1$
  - Set $x_n$ to 0, and check if $S > \bar{b}_i \Rightarrow$ Fix $x_n = 1$
  - Set $x_1$ to 1, and check if $S > \bar{b}_i \Rightarrow$ Fix $x_1 = 0$
The KnapSum Constraint Filtering Algorithm

- Let’s assume that for a constraint $\text{knapSum}(a_i, b_i, k, x)$ the coefficients $a_{ij}$ are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.

  $\bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i$ and $S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i$ (1)

- Phase 2: fixing variables
  - Set $x_1$ to 0, and check if $\bar{S} < \bar{b}_i \Rightarrow$ Fix $x_1 = 1$
  - Set $x_n$ to 0, and check if $S > \bar{b}_i \Rightarrow$ Fix $x_n = 1$
  - Set $x_1$ to 1, and check if $S > \bar{b}_i \Rightarrow$ Fix $x_1 = 0$
The KnapSum Constraint Filtering Algorithm

- Let’s assume that for a constraint \(\text{knapSum}(a_i, b_i, k, x)\) the coefficients \(a_{ij}\) are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.

\[
\text{Check } \bar{S} = \sum_{j=1}^{k} a_{ij} \geq b_i \quad \text{and} \quad S = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \quad (1)
\]

- Phase 2: fixing variables
  - Set \(x_1\) to 0, and check if \(\bar{S} < b_i \Rightarrow \text{Fix } x_1 = 1\)
  - Set \(x_n\) to 0, and check if \(S > \bar{b}_i \Rightarrow \text{Fix } x_n = 1\)
  - Set \(x_1\) to 1, and check if \(S > \bar{b}_i \Rightarrow \text{Fix } x_1 = 0\)
  - Set \(x_n\) to 1, and check if \(\bar{S} < b_i \Rightarrow \text{Fix } x_n = 0\)
Summary of the Hybrid Method for the MKP

1. Compute the **MP** bound
2. Add Reduced Cost Constraint
3. Perform the **CP** propagation:
   - knapSum & Reduced Costs Constraint

Philippe Michelon, Université d’Avignon
Combining equality constraints

- Given two equality constrains with integer coefficients:
  \[ a_1x = b_1 \quad \text{and} \quad a_2x = b_2 \]

where \( x \) is an integer variable.
Combining equality constraints

- Given two equality constraints with integer coefficients:

\[ a_1x = b_1 \quad \text{and} \quad a_2x = b_2 \]

where \( x \) is an integer variable.

- Then, there exists an equality constraint of the form:

\[ \alpha(a_1x) + a_2x = \alpha b_1 + b_2 \]

which is equivalent to the 2 above equations

\( (\alpha > \max\{|a_2x - b_2|\}) \)
Combining equality constraints

- Given two equality constraints with integer coefficients:
  
  \[ a_1x = b_1 \quad \text{and} \quad a_2x = b_2 \]

  where \( x \) is an integer variable.

- Then, there exists an equality constraint of the form:
  
  \[ \alpha(a_1x) + a_2x = \alpha b_1 + b_2 \]

  which is equivalent to the 2 above equations
  
  \[ \alpha > \max\{|a_2x - b_2|\} \]

- Old known result.
Combining equality constraints

- Given two equality constraints with integer coefficients:

\[ a_1x = b_1 \quad \text{and} \quad a_2x = b_2 \]

where \( x \) is an integer variable.

- Then, there exists an equality constraint of the form:

\[ \alpha(a_1x) + a_2x = \alpha b_1 + b_2 \]

which is equivalent to the 2 above equations

\[ (\alpha > \max\{|a_2x - b_2|\}) \]

- Old known result.

- Makes linear programming models weaker
Combining equality constraints

- Given two equality constraints with integer coefficients:

\[ a_1 x = b_1 \quad \text{and} \quad a_2 x = b_2 \]

where \( x \) is an integer variable.

- Then, there exists an equality constraint of the form:

\[ \alpha (a_1 x) + a_2 x = \alpha b_1 + b_2 \]

which is equivalent to the 2 above equations

\( \alpha > \max\{ |a_2 x - b_2| \} \)

- Old known result.

- Makes linear programming models weaker

- ... but could be useful for Constraint Programming.
Applying to Knapsack constraints

- Knapsack constraints are inequalities: $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$
Applying to Knapsack constraints

- Knapsack constraints are inequalities: \( \sum_{j=1}^{n} a_{ij} x_j \leq b_i \)
- We will use the slack variables \( f_i \) given by the LP and turn into equality constraints: \( \sum_{j=1}^{n} a_{ij} x_j + S_i = b_i \)
Applying to Knapsack constraints

- Knapsack constraints are inequalities: \( \sum_{j=1}^{n} a_{ij} x_j \leq b_i \)
- We will use the slack variables \( f_i \) given by the LP and turn into equality constraints: \( \sum_{j=1}^{n} a_{ij} x_j + S_i = b_i \)
- And apply the aggregation result.
Applying to Knapsack constraints

- Knapsack constraints are inequalities: \( \sum_{j=1}^{n} a_{ij} x_j \leq b_i \)
- We will use the slack variables \( f_i \) given by the LP and turn into equality constraints: \( \sum_{j=1}^{n} a_{ij} x_j + S_i = b_i \)
- And apply the aggregation result.
- These aggregated constraints will only be used in the CP layer of our process.
Applying to Knapsack constraints

- Knapsack constraints are inequalities: \( \sum_{j=1}^{n} a_{ij} x_j \leq b_i \)

- We will use the slack variables \( f_i \) given by the LP and turn into equality constraints: \( \sum_{j=1}^{n} a_{ij} x_j + S_i = b_i \)

- And apply the aggregation result.

- These aggregated constraints will only be used in the CP layer of our process.

- Several aggregations were tested but the most efficient was to aggregate each of initial constraints with a constraint requiring that the objective function to be greater than the best known solution.
Numerical Results for the MKP

- We compared our methods with two other methods: CPLEX and the algorithm presented in Vimont, Boussier and Vasquez [2007].

<table>
<thead>
<tr>
<th>Instance Set</th>
<th>CPLEX</th>
<th>VBV</th>
<th>KPS</th>
<th>KPS + Combo</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.100</td>
<td>5.16</td>
<td>0.19</td>
<td>0.06</td>
<td>0.05</td>
</tr>
<tr>
<td>5.250</td>
<td>244.02</td>
<td>99.73</td>
<td>1.5</td>
<td>1.23</td>
</tr>
<tr>
<td>5.500</td>
<td>4219.56</td>
<td>926.7</td>
<td>19.09</td>
<td>14.35</td>
</tr>
<tr>
<td>10.100</td>
<td>39.53</td>
<td>6.37</td>
<td>2.54</td>
<td>2.07</td>
</tr>
<tr>
<td>10.250</td>
<td>N/A</td>
<td>8536.26</td>
<td>3050.51</td>
<td>2392.99</td>
</tr>
<tr>
<td>30.100</td>
<td>3469.27</td>
<td>1376.09</td>
<td>587.5</td>
<td>380.83</td>
</tr>
</tbody>
</table>

Table: Average time for each instance set in seconds.
CP and MP (or IP) are complementary methods!
Conclusion

- CP and MP (or IP) are complementary methods!
- Actually, most of the preprocessing in MP-based Branch-and-Bounds are, at least, a light CP process, so that hybrid CP-MP methods are widely used ...
CP and MP (or IP) are complementary methods!

Actually, most of the preprocessing in MP-based Branch-and-Bounds are, at least, a light CP process, so that hybrid CP-MP methods are widely used ...

... but they can still be improved.
**Conclusion**

- CP and MP (or IP) are complementary methods!
- Actually, most of the preprocessing in MP-based Branch-and-Bounds are, at least, a light CP process, so that hybrid CP-MP methods are widely used ...  
- ... but they can still be improved.
- People working with ”pure” CP approach have developed very efficient rules and techniques for ”branching”.

**Philippe Michelon, Université d’Avignon**
CP and MP (or IP) are complementary methods!

Actually, most of the preprocessing in MP-based Branch-and-Bounds are, at least, a light CP process, so that hybrid CP-MP methods are widely used ...

... but they can still be improved.

People working with ”pure” CP approach have developed very efficient rules and techniques for ”branching”.

It might be an interesting and fruitful research direction to see how update these rules for ”pure” IP or Hybrid approach.
THAT’S IT!
Something else ...

- The Universities of Avignon, Federal of Ceará (Manoel Camplo), Federal of Rio de Janeiro (Nelson Maculan) and Federal Fluminense (Luiz Satoru Ochi) are collaborating in a PVE ("Ciência sem Fronteira") project.
The Universities of Avignon, Federal of Ceará (Manoel Camplo), Federal of Rio de Janeiro (Nelson Maculan) and Federal Fluminense (Luiz Satoru Ochi) are collaborating in a PVE ("Ciência sem Fronteira") project.

This project includes 3 "cotutelle" Ph.D. grants (one year in France).
Something else ...

- The Universities of Avignon, Federal of Ceará (Manoel Camplo), Federal of Rio de Janeiro (Nelson Maculan) and Federal Fluminense (Luiz Satoru Ochi) are collaborating in a PVE ("Ciência sem Fronteira") project.
- This project includes 3 "cotutelle" Ph.D. grants (one year in France).
- We are looking for good candidates ...
The Universities of Avignon, Federal of Ceará (Manoel Camplo), Federal of Rio de Janeiro (Nelson Maculan) and Federal Fluminense (Luiz Satoru Ochi) are collaborating in a PVE ("Ciência sem Fronteira") project.

This project includes 3 "cotutelle" Ph.D. grants (one year in France).

We are looking for good candidates ...

Besides this program, I am also looking for Ph.D. students who would like to make an entire Ph.D. in Avignon.