ANAIS SBPO
XLVI SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL

# 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 Langage 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

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

- $D_i$ 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: $3x_1 + 5x_2 + 7x_3 \leq 6 \qquad x \in \{0,1\}^3$

# 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: $3x_1 + 5x_2 + 7x_3 \leq 6 \qquad x \in \{0,1\}^3$

$$\Longrightarrow x_3 = 0$$

# 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

- $$\begin{cases} C_1 : & 6x_1 + 4x_2 + 4x_3 \geq 5 \\ C_2 : & 3x_1 + 5x_2 + 7x_3 \leq 6 \\ & x \in \{0,1\}^3 \end{cases}$$

# Example

- $$\begin{cases} C_1 : & 6x_1 + 4x_2 + 4x_3 \geq 5 \\ C_2 : & 3x_1 + 5x_2 + 7x_3 \leq 6 \\ & x \in \{0,1\}^3 \end{cases}$$
- $C_1 \Rightarrow$ Nothing !

# Example

- $$\begin{cases} C_1 : & 6x_1 + 4x_2 + 4x_3 \geq 5 \\ C_2 : & 3x_1 + 5x_2 + 7x_3 \leq 6 \\ & x \in \{0,1\}^3 \end{cases}$$
- $C_1 \Rightarrow$ Nothing !
- $C_2 \Rightarrow x_3 = 0$

# Example

- $$\begin{cases} C_1 & 6x_1 + 4x_2 \geq 5 \\ C_2 & 3x_1 + 5x_2 \leq 6 \end{cases}$$

# Example

- $$\begin{cases} C_1 & 6x_1 + 4x_2 \geq 5 \\ C_2 & 3x_1 + 5x_2 \leq 6 \end{cases}$$
- $C_1 \Rightarrow x_1 = 1$

# Example

- $$\begin{cases} C_1 & 6x_1 + 4x_2 \geq 5 \\ C_2 & 3x_1 + 5x_2 \leq 6 \end{cases}$$
- $C_1 \Rightarrow x_1 = 1$
- $$\begin{cases} C_1 & 4x_2 \geq -1 \\ C_2 & 5x_2 \leq 3 \end{cases}$$

# Example

- $$\begin{cases} C_1 & 6x_1 + 4x_2 \geq 5 \\ C_2 & 3x_1 + 5x_2 \leq 6 \end{cases}$$
- $C_1 \Rightarrow x_1 = 1$
- $$\begin{cases} C_1 & 4x_2 \geq -1 \\ C_2 & 5x_2 \leq 3 \end{cases}$$
- $C_2 \Rightarrow x_2 = 0$

# Example

- $$\begin{cases} C_1 & 6x_1 + 4x_2 \geq 5 \\ C_2 & 3x_1 + 5x_2 \leq 6 \end{cases}$$
- $C_1 \Rightarrow x_1 = 1$
- $$\begin{cases} C_1 & 4x_2 \geq -1 \\ C_2 & 5x_2 \leq 3 \end{cases}$$
- $C_2 \Rightarrow x_2 = 0$

# Example

- $$\begin{cases} C_1 & 6x_1 + 4x_2 \geq 5 \\ C_2 & 3x_1 + 5x_2 \leq 6 \end{cases}$$
- $C_1 \Rightarrow x_1 = 1$
- $$\begin{cases} C_1 & 4x_2 \geq -1 \\ C_2 & 5x_2 \leq 3 \end{cases}$$
- $C_2 \Rightarrow x_2 = 0$

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

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

# CP for Optimization

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

# CP for Optimization

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

# CP for Optimization

- $\min\{f(x)/C_j(x) \text{ is true } j = 1, \ldots, m \; x_i \in D_i \; i = 1, \ldots, n\}$
- $$\begin{cases} \min & Z \\ \text{s.t.} & C_j(x) \text{is true} \quad j = 1, \ldots, m \\ & Z = f(x) \\ & x_i \in D_i \qquad i = 1, \ldots, n \\ & Z \in [lb, ub] \end{cases}$$
- A filtering algorithm has to be designed for the objective function.

# CP for Optimization

- $\min\{f(x)/C_j(x) \text{ is true } j = 1, \ldots, m \; x_i \in D_i \; i = 1, \ldots, n\}$
- $$\begin{cases} \min & Z \\ \text{s.t.} & C_j(x) \text{is true} \quad j = 1, \ldots, m \\ & Z = f(x) \\ & x_i \in D_i \qquad i = 1, \ldots, n \\ & Z \in [lb, ub] \end{cases}$$
- A filtering algorithm has to be designed for the objective function.
- It generally aims to remove values from:

3360

# CP for Optimization

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

- $\min\{f(x)/C_j(x) \text{ is true } j = 1, \ldots, m \; x_i \in D_i \; i = 1, \ldots, n\}$
- $$\begin{cases} \min & Z \\ \text{s.t.} & C_j(x) \text{is true} \quad j = 1, \ldots, m \\ & Z = f(x) \\ & x_i \in D_i \qquad\qquad i = 1, \ldots, n \\ & Z \in [lb, ub] \end{cases}$$
- 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 $x_1 \leq \dfrac{b - \sum_{i=2}^{n} a_i x_i}{a_1}$
- If $a_1 \geq 0$ then $x_1 \leq \max\{\dfrac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x_i \in [l_i, u_i]\}$

# 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 $x_1 \leq \max\{\frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x_i \in [l_i, u_i]\}$
- If $a_1 \geq 0$ then $x_1 \leq \lfloor \frac{b - \sum_{a_i \leq 0} a_i u_i - \sum_{a_i \geq 0} a_i l_i}{a_1} \rfloor = M$

# 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 $x_1 \leq \lfloor \frac{b - \sum_{a_i \leq 0} a_i u_i - \sum_{a_i \geq 0} a_i l_i}{a_1} \rfloor = M$
- 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 $\max\{\frac{b - \sum_{i=2}^{n} a_i x_i}{a_1} / x \text{ feasible}\}$ or any relaxation ...
- It happens that more than one filtering algorithm are associated to the same constraint and are succesively called if they make different deductions ...

# 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 ...
- It happens that more than one filtering algorithm are associated to the same constraint and are succesively called if they make different deductions ...
- It also happens that different version 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 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 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 $\mathcal{L}$.

# Available tools

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

  www-01.ibm.com/softwre/commerce/optimization/cplex-cp-optimizer/

- CHOCO, open-source.

  www.emn.fr/z-info/choco-solver/

# 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.

# 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.
- $AllDifferent(x) \iff x_i \neq xj \qquad \forall 1 \leq i < j \leq n$
- 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.
- $AllDifferent(x) \iff x_i \neq xj \qquad \forall 1 \leq i < j \leq n$
- J-Ch Regin (94) designed a very efficient filtering algorithm for this global constraint.
- Example:
  $x_1 \in \{1,2,3\}$ $x_2 \in \{1,2\}$ $x_3 \in \{1,2,3,4\}$ $x_4 \in \{1,2\}$

# The AllDifferent Global Constraints

- AllDifferent should be used when n variables have to take a different value.
- $AllDifferent(x) \iff x_i \neq xj \qquad \forall 1 \leq i < j \leq n$
- J-Ch Regin (94) designed a very efficient filtering algorithm for this global constraint.
- Example:
  $x_1 \in \{1,2,3\}$ $x_2 \in \{1,2\}$ $x_3 \in \{1,2,3,4\}$ $x_4 \in \{1,2\}$

$AllDifferent(x) \implies x_1 = 3\ x_3 = 4.$

# 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.

# 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.

# 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.
- 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 !

# 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.

# 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.
-

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

# 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.
-

$$\begin{array}{ll} \min & \sum_{i=1}^{n-1} D_{p_i,p_{i+1}} + D_{p_n,p_1} \\ \text{s.t.} & AllDifferent(p) \text{ is true.} \end{array}$$

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

- Only one constraint !

# 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.
-

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

- 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) strenghen it by :

# 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) strenghen 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) strenghen 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) strenghen 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) strenghen 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.

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

# 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.
- 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.

# 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.
- 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.

# 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.
- 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

- Decison 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

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

$$
\begin{array}{ll}
\min & \sum_{i=1}^{n} C_{i,S_i} \\
\text{s.t.} & AllDifferent(S) \\
& S_i = j \Rightarrow l_j = l_i - r_i \ \forall i,j \\
& S_i = j \Rightarrow t_j \geq t_i + T_{ij} \ \forall i,j \\
& S_i \in \{1,\ldots,n\}/\{i\} \ \forall i \\
& l_i \in [0,K] \ \forall i \\
& t_i \in [a_i,b_i] \ \forall i
\end{array}
$$

# Modelling with CP : the VRPTW

Such a model has been used for the desing 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.

# The Cumulative Global Constraint

- 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 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.
- Hence, at each time of the planning period, the set of activities currently performed must respect the capacity of each of the resources.

# The Cumulative Global Constraint

- This set of constraint is not so easy to express in a linear way.

# 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 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 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)

# 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
- $Cumulative(S, d, r, R) \iff \sum_{\{i \;\; S_i \leq t < S_i + d_i\}} r_i \leq R \; \forall \, t$

# The RCPSP

- 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 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 are renewable.

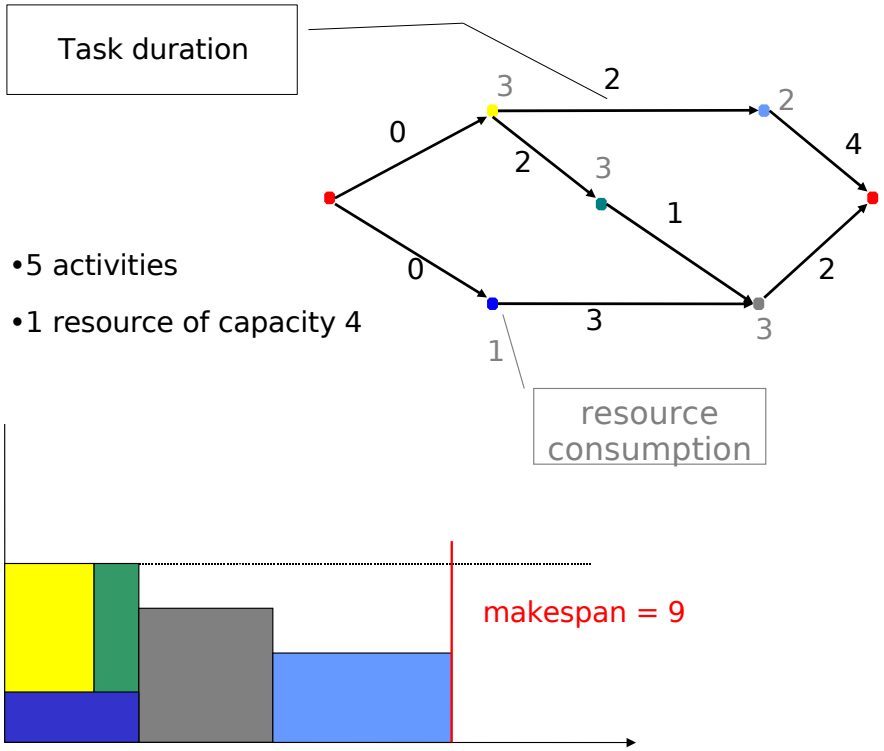
# The RCPSP

- 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 are renewable.
- The activities are not preemptive.

# 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 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

# 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. item

$$
\begin{array}{lll}
\min & C_{max} & \\
\text{s.t.:} & C_{max} \leq S_i + d_i & \forall i \\
 & S_j \geq S_i + d_i & \forall (i,j) \in E \\
 & Cumulative(S, d, r_{.k}, R_k) & \forall k \\
 & S \geq 0 &
\end{array}
$$

# 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.

# The Railway Rescheduling Problem

- Joint work with R. Acuña-Agost, S. Gueye, and D. Feillet.
- CP-AI-OR proceedings, Lecture Notes in Computer Sciences, V. 5547, 312-313, 2009.
- Deals with the the impact of disruptions in order to ensure the best possible service for the passengers.

# 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.

# 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.

# 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 $\leq$ 7h00 for the french network).

# Why studying a cooperative scheme for RRP ?

- Mathematical Programming based approaches are efficient ...
- for small or medium size instances (horizon $\leq$ 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 $\leq$ 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

- $x_k^{begin} \geq 0$ start time of event $k$

# Decision Variables for the Mathematical Programming Model

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

# Decision Variables for the Mathematical Programming Model

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

# Decision Variables for the Mathematical Programming Model

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

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

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

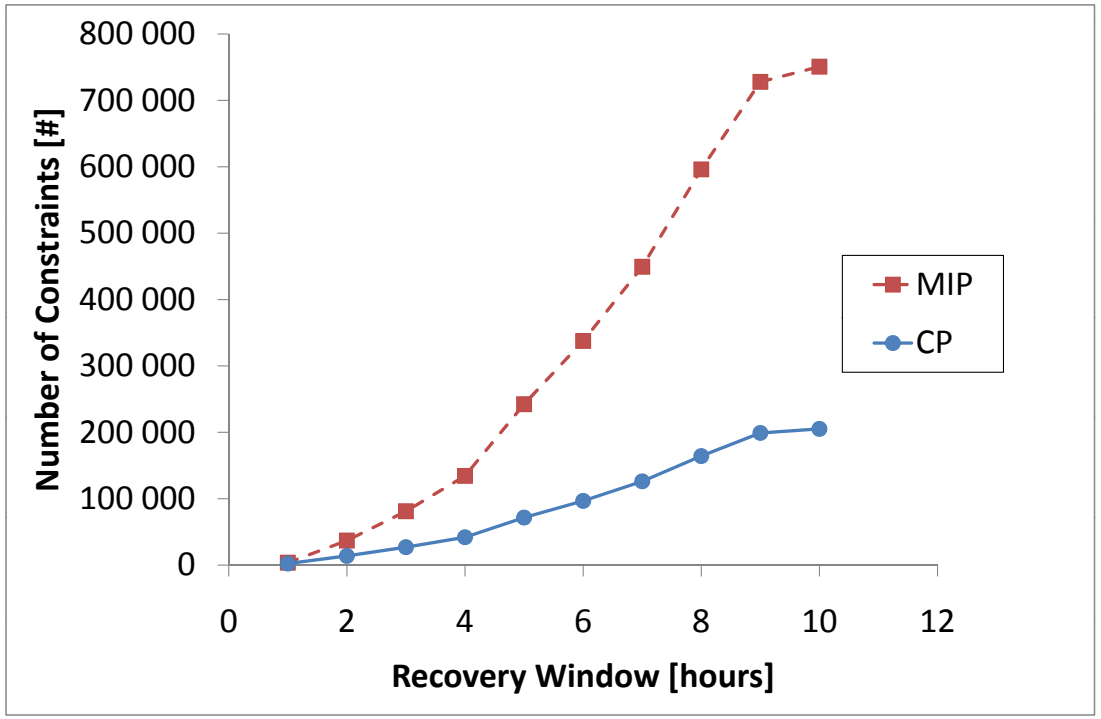
# Decision Variables for the Mathematical Programming Model

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

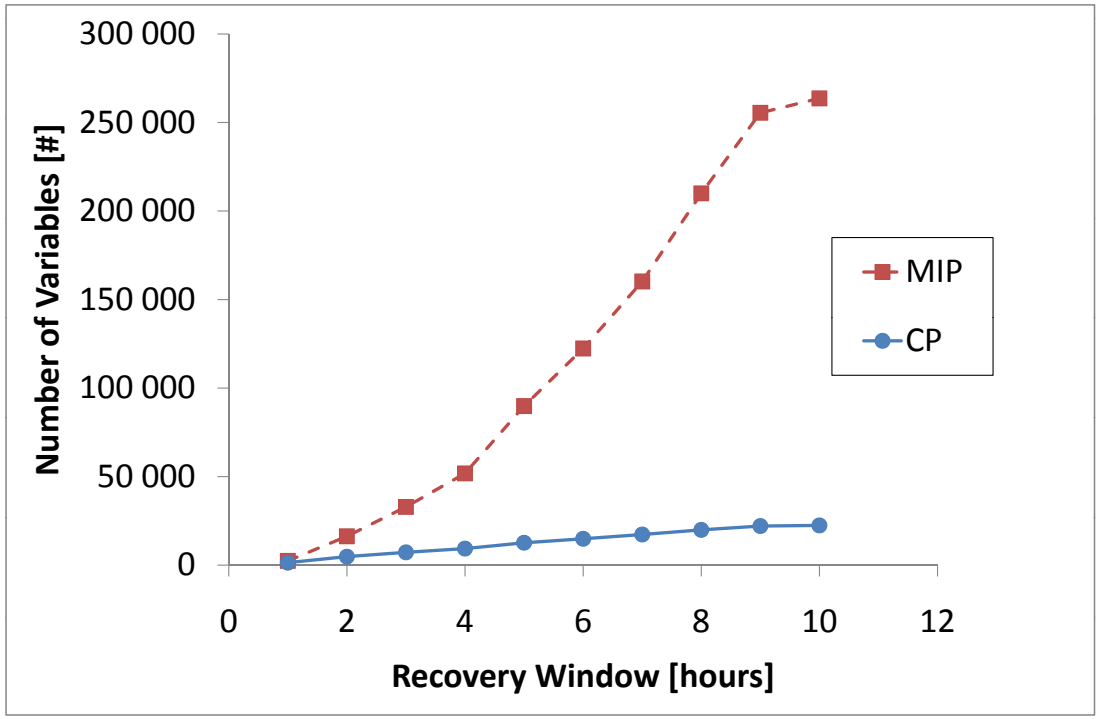
# Constraint Programming Decisions Variables

- $Delay_{in}$ : 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



# Comparison of the Models



# 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

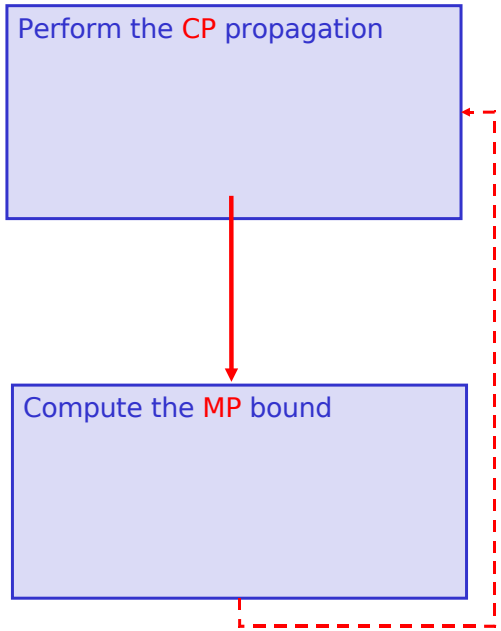
- Both methods use search trees.
- They are using quite different models and tools.
- CP is a "local" method while MP is "global".
- CP cannot be guided by an approximate solution.
- CP fully takes into account the domains of the variables while MP does not.

- They are complementary methods !
- Cooperative schemes should be useful !

# Cooperative Schemes

- 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



# The Reduced Cost Constraint

- $$\begin{array}{ll} \min & Z = \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} & \sum_{i=1}^{n} a_{j,i} x_i + s_j = b_j \\ & x \in \{0,1\}^n \\ & s \geq 0 \end{array}$$

# The Reduced Cost Constraint

- $$\begin{array}{ll} \min & Z = \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} & \sum_{i=1}^{n} a_{j,i} x_i + s_j = b_j \\ & x \in \{0,1\}^n \\ & s \geq 0 \end{array}$$

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

$$Z = lb + \sum_{i \in N_0} \overline{c}_i x_i + \sum_{i \in N_1} \overline{c}_i (1 - x_i) + \sum_j \overline{d}_j s_j$$

# The Reduced Cost Constraint

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

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

# The Reduced Cost Constraint

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

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

- This is the so called Reduced Cost Constraint.

# The Reduced Cost Constraint

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

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

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

# The Reduced Cost Constraint

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

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

- 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:

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

- 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.

# The Reduced Cost Constraint

Filtering algorithm:

- $s_j \leq \frac{ub - lb}{\overline{d}_j}$

# The Reduced Cost Constraint

Filtering algorithm:

- $s_j \leq \frac{ub-lb}{\overline{d}_j}$
- For $i \in N_1$, $ub - lb \leq \overline{c}_i \Longrightarrow x_i = 1$

# The Reduced Cost Constraint

Filtering algorithm:

- $s_j \leq \frac{ub-lb}{\overline{d}_j}$
- For $i \in N_1$, $ub - lb \leq \overline{c}_i \Longrightarrow x_i = 1$
- For $i \in N_0$, $ub - lb \leq \overline{c}_i \Longrightarrow x_i = 0$

# The Reduced Cost Constraint

Filtering algorithm:

- $s_j \leq \frac{ub-lb}{\overline{d}_j}$
- For $i \in N_1$, $ub - lb \leq \overline{c}_i \Longrightarrow x_i = 1$
- For $i \in N_0$, $ub - lb \leq \overline{c}_i \Longrightarrow x_i = 0$

# The Reduced Cost Constraint

Filtering algorithm:

- $s_j \leq \frac{ub-lb}{\overline{d}_j}$
- For $i \in N_1$, $ub - lb \leq \overline{c}_i \Longrightarrow x_i = 1$
- For $i \in N_0$, $ub - lb \leq \overline{c}_i \Longrightarrow x_i = 0$

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.

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.:} & Ax \leq b, \\ & x \in \{0,1\}^n, \end{array}$$

# The Multi-Knapsack Problem

- Generalisation of the classical knapsack Problem.

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.:} & Ax \leq b, \\ & x \in \{0,1\}^n, \end{array}$$

- 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 !
-

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

3500

# 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 !
-

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

- Let $\overline{k}$ and $\underline{k}$ be the optimal values of the above problem.

# Application to the Multi-Knapsack Problem

- For $k \in \{\underline{k}, \ldots, \overline{k}\}$, create a subproblem :

$$
\begin{array}{ll}
\max & c^T x \\
\text{s.t.:} & Ax \leq b, \\
& \sum_{i=1}^{n} x_i = k \\
& c^T x \geq lb \\
& x \in \{0,1\}^n,
\end{array}
$$

# Application to the Multi-Knapsack Problem

- For $k \in \{\underline{k}, \ldots, \overline{k}\}$, create a subproblem :

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.:} & Ax \leq b, \\ & \sum_{i=1}^{n} x_i = k \\ & c^T x \geq lb \\ & x \in \{0,1\}^n, \end{array}$$

- 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 takcle constaints of the forme :$\sum_{i=1}^{n} x_i = k$ and $\sum_{i=1}^{n} a_{ji} x_i \leq \bar{b}_j$

# The KnapSum Constraint

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

# The KnapSum Constraint

- At each node of the search tree, the CP lawyer has to takcle constaints of the forme :$\sum_{i=1}^{n} x_i = k$ and $\sum_{i=1}^{n} a_{ji} x_i \leq \bar{b}_j$
- They can be grouped into Global Constraints !
- $$knapSum(a_j, b_j, k, x) \Leftrightarrow \underline{b_i} \leq \sum_{i=1}^{n} a_{ji} x_i \leq \bar{b}_i; \quad \sum_{j=1}^{n} x_j = k$$

# The KnapSum Constraint Filtering Algorithm

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

# The KnapSum Constraint Filtering Algorithm

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

# The KnapSum Constraint Filtering Algorithm

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

# The KnapSum Constraint Filtering Algorithm

- Let's assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coeficients $a_{ij}$ are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  - $$\text{Check} \quad \bar{S} = \sum_{j=1}^{k} a_{ij} \geq \underline{b_i} \quad \text{and} \quad \underline{S} = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \qquad (1)$$
- Phase 2: fixing variables

# The KnapSum Constraint Filtering Algorithm

- Let's assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coeficients $a_{ij}$ are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  - $$\text{Check} \quad \bar{S} = \sum_{j=1}^{k} a_{ij} \geq \underline{b_i} \quad \text{and} \quad \underline{S} = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \qquad (1)$$
- Phase 2: fixing variables
  - Set $x_1$ to 0, and check if $\bar{S} < \underline{b_i} \Rightarrow$ Fix $x_1 = 1$

# The KnapSum Constraint Filtering Algorithm

- Let's assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coeficients $a_{ij}$ are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  - $$\text{Check} \quad \bar{S} = \sum_{j=1}^{k} a_{ij} \geq \underline{b_i} \quad \text{and} \quad \underline{S} = \sum_{j=n-k}^{n} a_{ij} \leq \bar{b}_i \qquad (1)$$
- Phase 2: fixing variables
  - Set $x_1$ to 0, and check if $\bar{S} < \underline{b_i} \Rightarrow$ Fix $x_1 = 1$

# The KnapSum Constraint Filtering Algorithm

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

# The KnapSum Constraint Filtering Algorithm

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

# The KnapSum Constraint Filtering Algorithm

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

# The KnapSum Constraint Filtering Algorithm

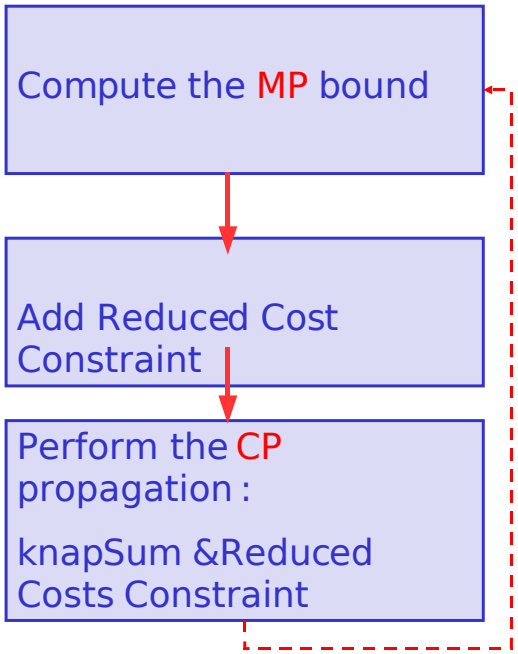
- Let's assume that for a constraint $knapSum(a_i, b_i, k, x)$ the coeficients $a_{ij}$ are ordered from the biggest to the smallest.
- Phase 1: checking the feasibility.
  - $$\text{Check} \quad \bar{S} = \sum_{j=1}^{k} a_{ij} \geq \underline{b_i} \quad \text{and} \quad \underline{S} = \sum_{j=n-k}^{n}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

# The KnapSum Constraint Filtering Algorithm

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

3518

# Summary of the Hybrid Method for the MKP



# Combining equality constraints

- Given two equality constrains with integer coefficients:

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

where $x$ is an integer variable.

# Combining equality constraints

- Given two equality constrains 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
$(\alpha > max\{\mid a_2 x - b_2 \mid\}$

# Combining equality constraints

- Given two equality constrains 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
  $(\alpha > max\{\mid a_2 x - b_2 \mid\}$
- Old knwon result.

# Combining equality constraints

- Given two equality constrains 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
  $(\alpha > max\{\mid a_2 x - b_2 \mid\}$
- Old knwon result.
- Makes linear programming models weaker

# Combining equality constraints

- Given two equality constrains 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
  $(\alpha > max\{\mid a_2 x - b_2 \mid\}$
- Old knwon 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 constrainst 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 constrainst will only be used in the CP layer of our process.
- Several aggregations were tested but the most efficient was to aggregate each of intial 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 : Average time for each instance set in seconds

| Instance Set | CPLEX | VBV | KPS | KPS + Combo |
|---|---|---|---|---|
| 5.100 | 5.16 | 0.19 | 0.06 | 0.05 |
| 5.250 | 244.02 | 99.73 | 1.5 | 1.23 |
| 5.500 | 4219.56 | 926.7 | 19.09 | 14.35 |
| 10.100 | 39.53 | 6.37 | 2.54 | 2.07 |
| 10.250 | N/A | 8536.26 | 3050.51 | 2392.99 |
| 30.100 | 3469.27 | 1376.09 | 587.5 | 380.83 |

# Conclusion

- 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 ...

# 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.

# 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 developped very efficient rules and techniques for "branching".

# 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 developped very efficient rules and techniques for "branching".
- It might be an interesting and fruitfull 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.

# 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).

# 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 ...

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