INTEGRATION of OPERATIONS RESEARCH and AI CONSTRAINT-BASED TECHNIQUES for COMBINATORIAL OPTIMIZATION

Michela Milano

Università di Bologna - ITALY

mmilano@deis.unibo.it

http://www-lia.deis.unibo.it/Staff/MichelaMilano/

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - AI and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - Search

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - AI and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - Search

COMBINATORIAL OPTIMIZATION PROBLEMS

- We consider discrete NP-hard problems

- Minimizing (Maximizing) a function of many variables subject to
  - *Mathematical constraints*
  - *Non binary constraints (referred to as global constraints)*
  - *Integrality restrictions on some or all variables*

- Many application areas:
  - *resource allocation, scheduling, planning, routing, sequencing, design,configuration….*

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - AI and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - Search

CSP AND OR TECHNIQUES

- Constraint Satisfaction
  - *Declarative Modelling*
  - *Constraint Propagation (Local Consistency)*
  - *Search*

- Operations Research
  - *branch & bound*
    - *Relaxation: e.g., Linear Programming*
  - *branch & cut*
    - *Cutting planes*
  - *column generation*
  - *dynamic programming*
- We consider only *COMPLETE methods*

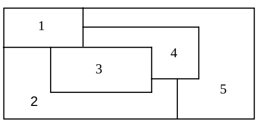
CONSTRAINT SATISFACTION

- Problem modelling
  - *Variables range on a finite domain of objects of arbitrary type*
  - *Constraints among variables*
    - *mathematical constraints*
    - *symbolic constraints*
- Problem solving
  - *Propagation algorithms embedded in constraints*
    - *Arc consistency as standard propagation*
    - *More sophisticated propagation for global constraints*
  - *Search strategies*
  - *Branch & Bound for optimization*
- CSP problems often modelled and solved through Constraint Programming (CP) languages

CSP: PROBLEM MODELLING

- A problem should be modelled in terms of
  - Variables ⇨ Problem entities
  - Domains ⇨ Possible Values
  - Constraints ⇨ Relations among variables
  - Objective function (if any) ⇨ Optimization Criteria

- Map coloring



variables: zones **X1,…,X5**
domains: colours **[red,yellow,blue,green]**
constraints: **X1≠X2,X1≠X3,X1≠X4,X1≠X5,**
X2≠X3,X2≠X4,X2≠X5,X3≠X4,X4≠X5

CSP: PROBLEM CONSTRAINTS

- Mathematical constraints: =, >, <, ≠, ≥, ≤
  - Propagation: arc-consistency, bound-consistency
- Non binary constraints
  - General methods: GAC ***[Bessiere, Regin IJCAI 99]***
  - Special purpose propagation algorithms ***[Beldiceanu, Contejean, Math.Comp.Mod. 94]***
  - More concise formulation
    - **alldifferent([X_1,...X_m])**
      all variables have different values
    - **element(N,[X_1,...X_m],Value)**
      the n-th element of the list should be equal to Value
    - **cumulative([S_1,...S_m],[D_1,...D_n],[R_1,...R_n],L)**
      used for capacity constraints
    - disjunctive constraints

CSP: PROBLEM SOLVING

- Notion of Consistency:
  - Is the set of constraint consistent ?
  - Does a solution exist?

- Constraint Propagation: inference mechanism
  - Remove from domains inconsistent values
  - Infer new constraints

- Search: branching strategies
  - Variable selection
  - Value selection
  - Others: problem dependent

CSP:OPTIMIZATION

- In some applications, we are not interested in a feasible solution but in the OPTIMAL solution according to a given criterion

- ENUMERATION ⟹ inefficient
  - find all feasible solutions
  - chose the best one

- CSP Branch & Bound
  - each time a solution is found whose cost is C*, impose a constraint on the remaining search tree, stating that further solutions (whose cost is C) should be better than the best one found so far

C < C*

OR: BRANCH & BOUND

- Two step tree search procedure:
  - solving a relaxation of the original problem
  - splitting the problem into subproblems

- Relaxation:
  - consider the original problem P at a given node
  - relax some constraints and generate P_{rel} <u>easier than</u> P
- Branching:
  - select a variable Var
  - impose additional constraints on Var

OR: BRANCH & BOUND

- The relaxation P_{rel} of a problem P provides a lower bound for P in minimization problems, an upper bound for P in maximization problems

- For minimization problems
  - Z_{rel}^* = optimal solution of P_{rel}
  - Z^* = optimal solution of P

  $\Rightarrow$ $\boxed{Z_{rel}^* \leq Z^*}$

- At a given node, if the lower bound is greater than the best solution found so far, the corresponding subtree can be pruned.

OR: BRANCH & BOUND

- Use of upper bounds in minimization problems:
  - we need a solution Z_{up}
  - solve the problem with a heuristic algorithm

- For minimization problems
  - Z_{up}^* = heuristic solution of P
  - Z^* = optimal solution of P

  $\Rightarrow$ $\boxed{Z^* \leq Z_{up}^*}$

- At a given node, if the lower bound is greater than the upper bound, the corresponding subtree can be pruned.
- Upper bound updated during search

INTEGER PROGRAMMING

- Standard form of Combinatorial Optimization Problem (IP)
  [Nemhauser Wolsey: Integer and Combinatorial Optimization 88]

  – *min* $z = \sum_{j=1}^{n} c_j x_j$
  – *subject to*

  $\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1..m$

  $x_j \geq 0 \qquad j = 1..n$

  x_j *integer* ⇐ May make the problem NP complete
- Inequality y ≥ 0 recasted in y - s = 0
- Maximization expressed by negating the objective function
- When only some variables should be integer: Mixed Integer (Linear) Problem MIP

0-1 INTEGER PROGRAMMING

- Many Combinatorial Optimization Problem can be expressed in terms of 0-1 variables (IP)

  – *min* $z = \sum_{j=1}^{n} c_j x_j$
  – *subject to*

  $\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1..m$

  x_j *:[0,1]* ⇐ May make the problem NP complete

LINEAR RELAXATION

- General form of Combinatorial Optimization Problem (IP)

– $\min \; z = \sum_{j=1}^{n} c_j x_j$

– *subject to*

$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1..m$

$x_j \geq 0 \quad j = 1..n$

x_j *integer* ⟸ Removed

⟸ Linear Relaxation

- The linear relaxation is solvable in POLYNOMIAL TIME
- The SIMPLEX ALGORITHM is the technique of choice even if it is exponential in the worst case

0-1 LINEAR PROGRAMMING

- Many Combinatorial Optimization Problem can be expressed in terms of 0-1 variables (IP)

– $\min \; z = \sum_{j=1}^{n} c_j x_j$

– *subject to*

$\sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1..m$

$0 \leq x_j \leq 1$ ⟸ Relaxed

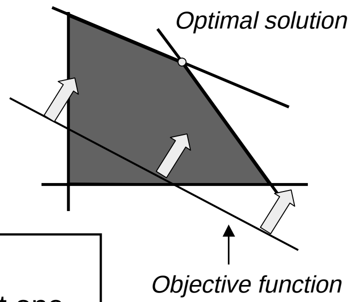
⟸ Linear Relaxation

GEOMETRIC PROPERTIES OF LP

- The set of constraints defines a polytope
- The optimal solution is located on one of its vertices

– *min* $z = \sum_{j=1}^{n} c_j x_j$
– *subject to*

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1..m$$
$$x_j \geq 0 \qquad j = 1..n$$



The simplex algorithm starts from
one vertex and moves to an adjacent one
with a better value of the objective function

PROPERTIES OF LP SOLUTION

- The optimal solution of the relaxation is an assignment of values to variables such that all linear inequalities are satisfied and the objective function is minimized
- The optimal LP solution is in general fractional: violates the integrality constraint
- Each LP (primal) has an associated dual problem where dual variables correspond to constraints and dual constraints correspond to primal variables

DUAL PROBLEM

min $z = c\,x$ s.t.			max $z' = \pi\, b$ s.t.	
$a_i\, x = b_i$	$i \in M$	$\leftrightarrow$	$\pi_i \lesseqgtr 0$	$i \in M$
$a_i\, x \geq b_i$	$i \in \overline{M}$	$\leftrightarrow$	$\pi_i \geq 0$	$i \in \overline{M}$
$x_j \geq 0$	$j \in N$	$\leftrightarrow$	$\pi\, A_j \leq c_j$	$j \in N$
$x_j \lesseqgtr 0$	$j \in \overline{N}$	$\leftrightarrow$	$\pi\, A_j = c_j$	$j \in \overline{N}$

- The optimal LP solution provides reduced costs (through dual variables) representing the cost to be paid if a given value is in the solution

EXAMPLE

- Produce and sell two kinds of products: A and B
- Both products have to be processed on two machines M1 and M2
  - Product A process lasts 12 min. on M1 and 30 min. on M2
  - Product B process lasts 24 min. on M1 and 24 min. on M2
  - The resource availability of M1 is 400 hours and that of M2 is 490 hours
  - Profit of selling product A is $12 and for product B is $20.
- Goal: produce at least 100 units of each product and maximize the profit

EXAMPLE: MODEL

- Decision variables **`prodA`** and **`prodB`** representing the quantity to be produced. Convert minutes to hours

max 12*prodA + 20*prodB ***(Profit to be maximized)***

s.t.

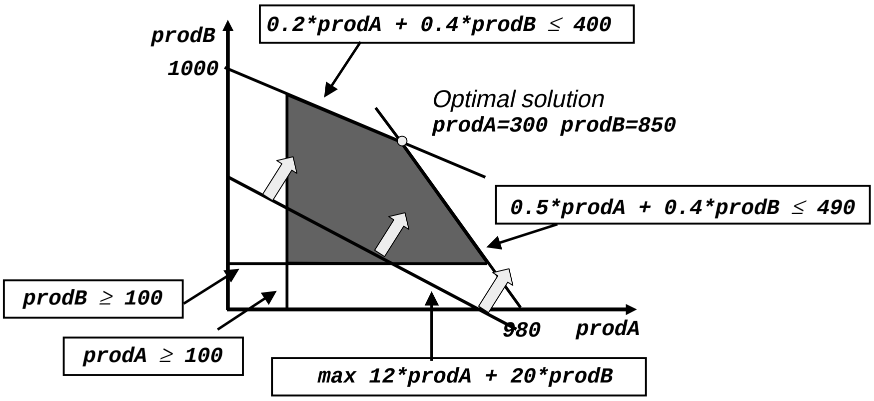
$$0.2 \cdot prodA + 0.4 \cdot prodB \leq 400 \quad \textit{(Availability M1)}$$

$$0.5 \cdot prodA + 0.4 \cdot prodB \leq 490 \quad \textit{(Availability M2)}$$

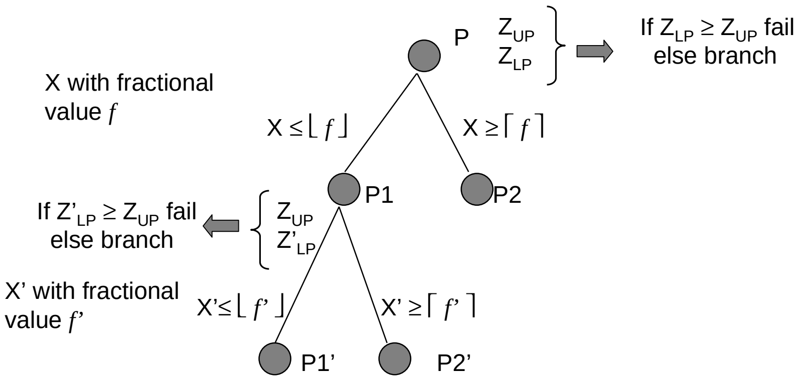
$$\left.\begin{array}{l} prodA \geq 100 \\ prodB \geq 100 \end{array}\right\} \quad \textit{(minimal quantity required)}$$

prodA, prodB integer

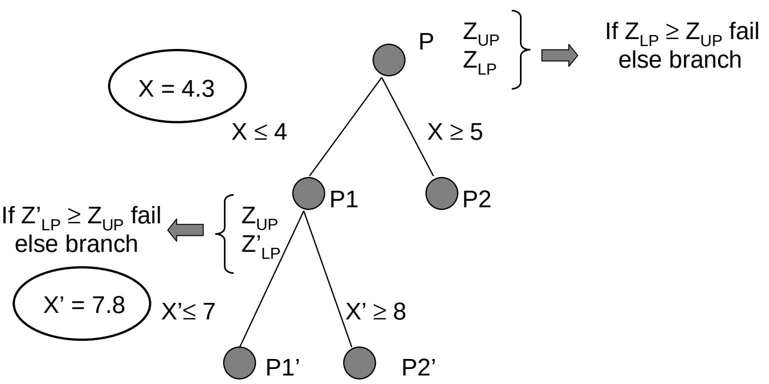
EXAMPLE: MODEL



BRANCH & BOUND based on LP



BRANCH & BOUND based on LP: EXAMPLE

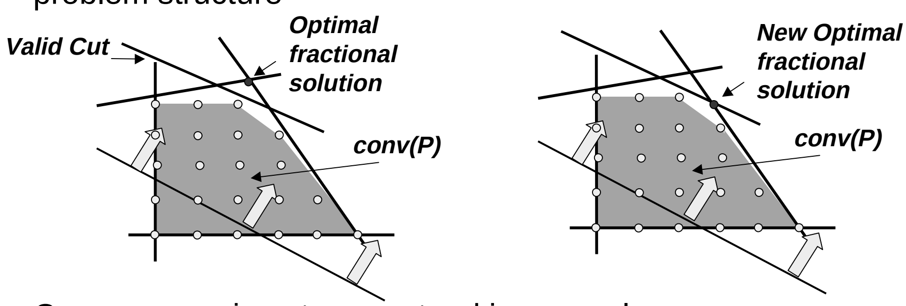


OR: CUTTING PLANES ALGORITHM

- Iterative procedure:
  - solving a linear relaxation of the problem P, x* optimal solution
  - add cutting planes when the optimal solution of the relaxation is not integral
- Cutting Planes: *[Gomory, 63]*
- linear inequalities $\alpha x \le \alpha_0$
  - should cut off the optimal solution of the Linear Relaxation
    - $\alpha x^* > \alpha_0$
  - should not remove any integer solution ⇒ valid cut
    - $\alpha x \le \alpha_0 \ \forall x \in$ conv(P) where conv(P) is the convex hull of P

OR: CUTTING PLANES ALGORITHM

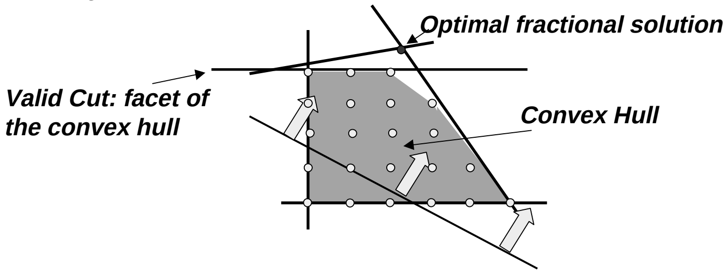
- Cutting Planes: syntactic cuts: do not exploit the problem structure



- Convergence is not guaranteed in general
- For some cases, i.e., Gomory cuts, the process converges but it can be too expensive

POLYHEDRAL CUTS

- Problem structure dependent
- Given an Integer Problem: S is the set of its solutions
  - conv(S): convex hull of S
  - if we have a constraint representation of the convex hull we can optimally solve the IP with Linear Programming
  - impossible to find the conv(S) efficiently
- Idea: generate cuts that are facets of the convex hull



OR: BRANCH & CUT

- Integrates Branch & Bound and Cutting Planes
- Two step tree search procedure: at each node
  - solving a relaxation of the original problem
  - add cuts when the optimal solution of the relaxation is not integral in order to improve the bound
- Branch when cuts are no longer effective
  - Cuts valid locally to a node: too memory expensive
- In Branch & Cut in general we have a unique pool of cuts globally valid

OR: BENDERS DECOMPOSITION

- Cutting planes generation technique for the solution of specially structured MIP.
- Given a problem P if for a subset of variables $X \subset V$ a fixing can be identified partitioning the problem P in disconnected subproblems Sp_i which are easily solvable we can solve P by a two step search procedure.
  - At each iteration a Relaxed Master Problem RMP_k is solved for assigning variables in $X = X_k$. These values are used to build subproblems SP_i^k.
  - Sp_i^k are solved and the solutions used to tighten the relaxation RMP_k by introducing Benders cuts $\beta_i^k(X)$
- Benders cuts play the role of nogoods in CSPs.

OR: COLUMN GENERATION

- Method for solving large scale problems *[Dantzig-Wolfe Econometrica 61] [Gilmore, Gomory OR61]*
- Avoid considering all variables (say X) of the problem, but consider only a subset $X' \subset X$ $\Rightarrow$ *MASTER PROBLEM*
- Once the master problem is solved, search for new variables in X\X' which can improve the solution $\Rightarrow$ *SUBPROBLEM*
- From duality theory variables with negative reduced costs improve the solution

OR: COLUMN GENERATION-EXAMPLE

- Partition a Directed Acyclic Graph into the minimal number of paths.
- *MASTER PROBLEM* ⇨ Choose paths

V: set of variables (n nodes O(2^n))
$x_j \in \{0,1\}$ x_j = 1 if Path j is choosen

min $\Sigma_{xj\in V}$ x_j
$\Sigma_{xj\in V}$ a_{ij} x_j = 1 each node belongs to one path

⇨ ***Provides dual values λ_i***

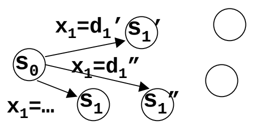
- Solve the master problem on V' ⊂ V, then add variables by solving the *SUBPROBLEM* ⇨ find a path with negative reduced costs

$y_j \in \{0,1\}$ y_j = 1 if node i is on that path
z cost of the path
$\boxed{z - \Sigma_{i\in V} \lambda_i y_i \leq 0}$

⇨ ***Provides new columns x_{ji}***

DYNAMIC PROGRAMMING

- Problems are decomposed into a nested family of subproblems.
- Consider a discrete system characterized by:
  - *a final state* s_k
  - *a decision variable* x_k
  - *a cost/profit function* $p(s_k, x_k)$
  - *a state transition function* $s_k = t(s_{k-1}, x_k)$



⇨ ***State space graph***: each node is a state, each arc a transition whose cost/profit is the corresponding p

- In a minimization problem the objective function is:
  - $z = \min\{\Sigma_{k=1}^{n} p(s_k, x_k)\}$

DYNAMIC PROGRAMMING

• DP solves a set of subproblems each corresponding to a system composed by $\mathbf{i}$ steps and a state $\mathbf{s_i}$ at the end of step $\mathbf{i}$.

• The cost function is computed as

$$f_i(s_i) = \min_{x_i} \{ \min_{s_{i-1}} \{f_{i-1}(s_{i-1})\} + p_i(s_i,x_i)\}$$

where $s_i = t_i(s_{i-1},x_i)$

• Boundary condition: if $s_1 = t_1(s_0,x_1)$

$$f_1(s_1) = \min_{x_1} \{p_1(s_1,x_1)\}$$

EXAMPLE: TSP

• DP formulation of a TSP (defined on vertex set V):
  - *each state* $s_i = (Y,v_k)$ *where* $v_k \in Y$ *represents the path of cardinality* $\mathbf{i}$ *covering all nodes in* $\mathbf{Y}$ *and ending in* v_k
  - *a transition is* $((Y,i),(Y\cup\{i\},j))$
  - *the cost of the transition* $((Y,i),(Y\cup\{i\},j))$ *is* c_{ij}

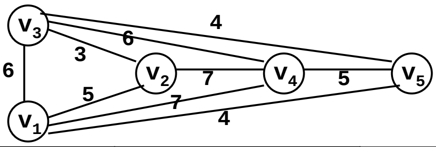
• The cost function is computed as

$$f_k(Y,i) = \min_{j\in Y\setminus\{i\}\cap E_i} \{f_{k-1}(Y\setminus\{i\}) + c_{ij}\} \quad Y\subseteq V\setminus\{0\}, |Y|\geq 2, \forall i\in Y$$

where $\mathbf{V}$ is the vertex set and E_i the set of arcs ending in $\mathbf{i}$

• Boundary condition: $f_1(\{i\},i) = c_{0i}$

EXAMPLE: TSP



|Y|=1			|Y|=2			|Y|=3			|Y|=4		
Y	v_i	f_1	Y	v_i	f_2	Y	v_i	f_3	Y	v_i	f_4
$\{v_2\}$	v_2	5	$\{v_2\ v_3\}$	v_2	9	$\{v_2\ v_3\ v_4\}$	v_2	16	$\{v_2\ v_3\ v_4\ v_5\}$	v_2	18
$\{v_3\}$	v_3	6	$\{v_2\ v_3\}$	v_3	8	$\{v_2\ v_3\ v_4\}$	v_3	17	$\{v_2\ v_3\ v_4\ v_5\}$	v_3	19
$\{v_4\}$	v_4	7	$\{v_2\ v_4\}$	v_2	14	$\{v_2\ v_3\ v_4\}$	v_4	14	$\{v_2\ v_3\ v_4\ v_5\}$	v_4	17
$\{v_5\}$	v_5	4	$\{v_2\ v_4\}$	v_4	12	$\{v_2\ v_3\ v_5\}$	v_2	11	$\{v_2\ v_3\ v_4\ v_5\}$	v_5	19
			$\{v_2\ v_5\}$	v_2	$+\infty$	$\{v_2\ v_3\ v_5\}$	v_3	$+\infty$			
			$\{v_2\ v_5\}$	v_5	$+\infty$	$\{v_2\ v_3\ v_5\}$	v_5	12			
			$\{v_3\ v_4\}$	v_3	13	$\{v_2\ v_4\ v_5\}$	v_2	16			
			$\{v_3\ v_4\}$	v_4	12	$\{v_2\ v_4\ v_5\}$	v_4	$+\infty$			
			$\{v_3\ v_5\}$	v_3	8	$\{v_2\ v_4\ v_5\}$	v_5	17			
			$\{v_3\ v_5\}$	v_5	10	$\{v_3\ v_4\ v_5\}$	v_3	15			
			$\{v_4\ v_5\}$	v_4	9	$\{v_3\ v_4\ v_5\}$	v_4	14			
			$\{v_4\ v_5\}$	v_5	12	$\{v_3\ v_4\ v_5\}$	v_5	17			

STATE SPACE RELAXATION

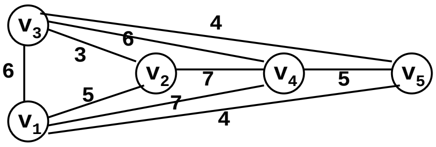
- Number of states in DP: exponential
- Relaxations can be defined to obtain bound on the problem
- Consider a function **w** mapping
  - *each state* $\mathbf{s_k}$ *in* $\mathbf{w(s_k)}$
  - *each transition between* $\mathbf{(s_i\ s_k)}$ *in* $\mathbf{(w(s_i),w(s_k))}$

State Space Relaxation

useful if the resulting number of states is polynomial

– Different functions **w** can be defined: in our example **w** is based on the cardiality of the path

STATE SPACE RELAXATION



•The cost function is computed as

$$g_k(h,i) = \min \{g_{k-1}(h-1,j) + c_{ji}\} \quad \forall i \in V \setminus \{0\}$$

Boundary condition: $g_1(1,i) = c_{0i}$

v_i	h=1	h=2	h=3	h=4
v_2	5	9	11	15
v_3	6	8	12	14
v_4	7	9	14	17
v_5	4	10	12	16

SPECIAL PURPOSE ALGORITHMS

- Beside the general methods (branch and bound, branch and cut, column generation), we have special purpose algorithms suited for solving a particular structured problem:
  - *network flow algorithms*
  - *primal-dual algorithms*
  - *edge finder*
  - *...*

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - AI and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

COMPARISONS

- Many studies on:
  - comparisons on the same practical application
    - *[Smith et al. Constraints 96], [Puget, De Backer APMOD'95],[Darby-Dowman et al. Constraints 98], [Proll, Smith INFORMS JOC98], [Darby-Dowman, Little INFORMS JOC98]*
  - comparisons for defining general similarities and differences in abstract ways
    - *[Heipcke Annals of OR99],[Darby-Dowman, Little INFORMS JOC98], [Van Hentenryck, CP95],* [*Williams Tut.EURO XVI],*
  - works aimed at defining unifying frameworks
    - *[Bockmayr, Kasper INFORMS JOC 98], [Heipcke PhD99]*
    - *Logic based methods for optimization [Hooker, 2000]*

COMPARISONS: MODELING

- Problem definition:

CSP model

variables $X_1,X_2, \ldots,X_n$

domains $D_1,D_2, \ldots,D_n$

constraints $c(X_1,\ldots,X_k)$

objective function

$f(X_1,X_2, \ldots,X_n)$

MIP model

min $z = c^T x + h^T y$

subject to

$Ax + By = b$

$x \geq 0, y \geq 0$

$x \in Z, y \in R$

COMPARISONS: MODELING

- Variables
  - CSP variables have a name and a domain. Type: real, rational, boolean, sets, integer, symbolic
  - MIP variables: binary, integer, real (semi-continuous), member of sets, non-zero

- Domains:
  - In CSP the domain contains values that can be assigned to the variable and that are not proved to be inconsistent
  - In MIP variables can have bounds

COMPARISONS: MODELING

- Constraints:
  - In CSPs a constraint is a relation (true or false) over a set of variables
    - Domain constraints X :: [1,2,5,7], Y:: [1..10]
    - Mathematical constraints X = Y, X ≤ Y, X ≠ Y,...
    - Symbolic constraints alldifferent([X,Y,Z,K])
  - In MIP constraints are equalities/inequalities between linear terms plus the integrality constraint which is relaxed in the correspondent LP

- Redundant Constraints: entailed by other constraints
  - In CSP the addition of redundant constraints can help CSP solution procedures
  - In MIP a similar concept is the addition of valid cuts

COMPARISONS: MODELING

- Objective function:
  - In the two frameworks it has the same meaning.

- Feasible solution:
  - In CSP a feasible solution is an assignment of values to variables that satisfies all the constraints. Solution may also denote the result after the application of a (local) consistency algorithm.

    If such an assignment exists, the problem is feasible

  - In MIP, the solution space is denoted by:
    - $S = \{(x,y) : Ax + By = b, x \geq 0, y \geq 0\}$

    If S is not empty, the problem is feasible

COMPARISONS: MODELING general remarks

- Model
  - CSP has a more intuitive, declarative and flexible problem formulation
  - MIP requires more expertise in order to write a good model
  - Both approaches can model the same problem in different ways. One model can be better than another.

- Relaxations:
  - In CSP each constraint represents an independent subproblem
    - feasibility problem
    - adding constraints is straightforward
  - In MIP some constraints are relaxed (say the integrality).
    - Optimization problem

COMPARISONS: SOLVING

- Solution method: tree search
  - CSP: each node is generated by a labelling procedure. At each node propagation is performed until a fix point is reached. Constraint Propagation achieves a consistency property. Optimality is dealt with by imposing a cost constraint that (poorly) propagates to variables

  - MIP: each node is generated by setting variable bounds. At each node the Linear relaxation is solved to optimality. If the lower bound value is worse than the current best solution, the node is fathomed as its successors can only be worse

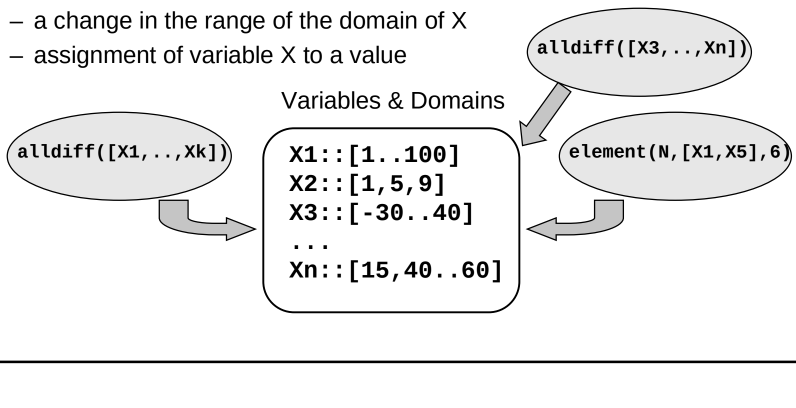
COMPARISONS: SOLVING

- Constraint Propagation: CSP
  - Performed at each node
  - Consistency algorithm remove values which cannot appear in a consistent solution. If a domain becomes empty, the corresponding problem is infeasible.
  - **NC, AC:*AC1-3*** *[Mackworth AIJ (8), 77] [Montanari Inf.Sci (7), 74]*, ***AC4*** *[Mohr, Henderson AIJ(28), 86]*, ***AC5*** *[Van Hentenryck, Deville and Teng AIJ(58), 92]*, ***AC6*** *[Bessiere AIJ(65), 94]*, ***AC7*** *[Bessiere, Freuder, Regin AIJ(107), 99]*, **PC:** *[Mackworth AIJ (8), 77]*, ***PC3*** *[Mohr, Henderson AIJ(28), 86]* ***PC4*** *[Han, Lee AIJ(36), 88]* , ***k-consistency*** *[Freuder CACM (21), 78], [Cooper AIJ (41), 89]*, GAC (for non binary constraints) *[Bessiere, Regin IJCAI 99]*, specialized procedured.
  - Trade off between time spent at a node and total number of nodes
  - Constraints can be seen as interacting agents triggering propagation each time an event is raised

INTERACTION AMONG CONSTRAINTS

- Constraints interact with each other through shared variables (and their domains) in the constraint store

- Trigger propagation each time an event is raised on one variable X
  - a change in the domain of X
  - a change in the range of the domain of X
  - assignment of variable X to a value



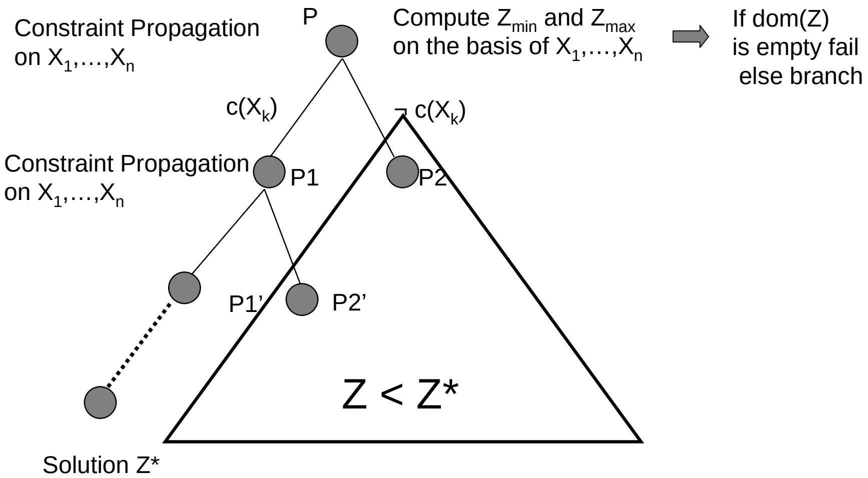
COMPARISONS: SOLVING

- Pre-processing: (MIP)
  - Performed at the root node
  - Variable fixing, variable removal, multiple, redundant or dominated row removal, variable bound tightening, matrix scaling, probing (addition of logical consequences)
  - Only sometimes applied also at other nodes
- Cut generation: (MIP)
  - Addition of valid inequalities
  - Global cuts are globally valid. Local cuts are valid in a subtree.

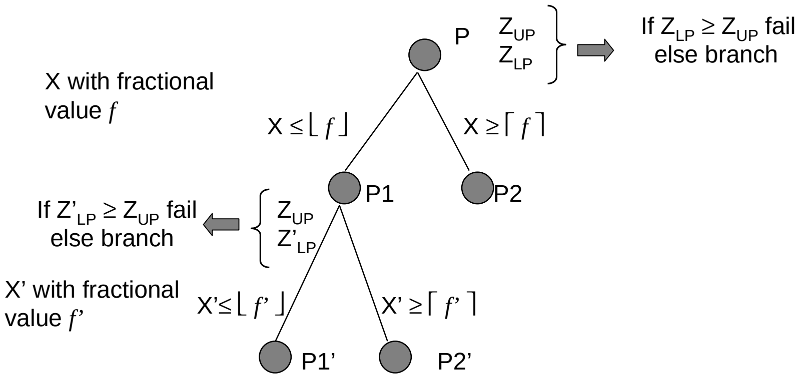
COMPARISONS: SOLVING

- Optimization:
  - In CSP each time a feasible solution is found Z*, a constraint is added on the objective function variable Z < Z*. Since Z is linked to problem variables, propagation is performed. At each node, when variables are instantiated and propagation is performed, bounds of Z are updated.
  - In MIP, at each node the LP relaxation is solved providing a lower bound on the problem. If the lower bound is worse than the current upper bound, the node is fathomed. Otherwise, a non integral variable x is selected and the branching is performed on its bounds. An initial upper bound can be in general computed.

BRANCH & BOUND in CSP



BRANCH & BOUND based on LP



UNIFYING FRAMEWORKS

- Purpose: define general concepts aimed at capturing the basic concepts of CP and OR techniques
- Define basis for understanding correspondences, similarities and differences between the two approaches
- Define the basis for a possible integration

UNIFYING FRAMEWORK

- Branch and Infer *[Bockmayr Kasper INFORMS JoC98]*
  - identifies common concepts to ILP and CP
- Primitive and non primitive constraints
  - CP primitive constraints: $\{X \leq u, X \geq b, X \neq v, X = Y, \text{integer}(X)\}$
  - ILP primitive constraints: linear equalities and inequalities
- Branch and Infer based on transition rules
  - *name_rule* : $\dfrac{\langle P,S \rangle}{\langle P',S' \rangle}$ *if Cond*

P: disjunctive subproblems

S: feasible solution set

UNIFYING FRAMEWORK

- Goal: derive primitive from non primitive constraints
  - *bi_infer :* $\dfrac{\langle (c \cup C) \cup P,S \rangle}{\langle (p \cup (c \cup C)) \cup P,S \rangle}$ p: primitive, c: non primitive; $Prim(C) \not\rightarrow p$ $Prim(C) \wedge c \rightarrow p$
- CP: global constraints
  - declarative abstractions embedding powerful filtering algorithms: derive primitive constraints $X \leq u$, $X \geq b$, $X \neq v$
- MIP: addition of cuts
  - cuts are primitive constraints

UNIFYING FRAMEWORK

- Branching: splitting the problem into subproblems
  - *bi_branch :* $\dfrac{\langle C \cup P,S \rangle}{\langle \{c_1 \cup C, \ldots, c_k \cup C\} \cup P,S \rangle}$ $C \equiv C \wedge (\vee\, c_i)$; if c_i primitive $Prim(C) \not\rightarrow c_i$
- Branching can be avoided if a subproblem is infeasible
  - *bi_clash:* $\dfrac{\langle C \cup P,S \rangle}{\langle P,S \rangle}$ $Prim(C) \rightarrow \perp$
  - *The infeasibility test is performed only on the relaxation generated by primitive constraints*

UNIFYING FRAMEWORK

- Solving Combinatorial Problems:

  – *bi_sol :*

$$\frac{\langle C \cup P, S\rangle}{\langle P, S \cup S^*\rangle} \qquad \begin{array}{l} S^* = \text{extract}(\text{Prim}(C)) \\ S^* \rightarrow C \end{array}$$

- Solving Combinatorial Optimization Problems: B&B *a-la* CP

  – *bi_climb:*

$$\frac{\langle \{C_1,\ldots,C_n\}, \{s\}\rangle}{\langle \{c\cup C, c\cup C_1,\ldots, c\cup C_n\}, \{s^*\}\rangle} \qquad \begin{array}{l} s^* = \text{extract}(\text{Prim}(C)) \\ f(s^*)>f(s),\ c\equiv (f(x) > f(s^*)) \end{array}$$

  – *extract: responsible of extracting a feasible solution of the relaxation*

UNIFYING FRAMEWORK

- Solving Combinatorial Optimization Problems: B&B *a-la* IP

  – *bi_bound :*

$$\frac{\langle C \cup P, \{s\}\rangle}{\langle P, \{s\}\rangle} \qquad \begin{array}{l} \text{if } \max\{f(x): x\in \text{sol}(C)\} \leq \text{lub} \leq f(s) \\ C \text{ is fathomed} \end{array}$$

  – *bi_opt:*

$$\frac{\langle C \cup P, \{s\}\rangle}{\langle P, \{s^*\}\rangle} \qquad \begin{array}{l} f(s^*) = \max\{f(x): x\in \text{sol}(\text{Prim}(C))\} \\ s\in \text{sol}(C) \qquad f(s^*) > f(s) \end{array}$$

  – *the lub should be computed efficiently*

UNIFYING FRAMEWORK

- Combinatorial Problems in CP(FD)
  - *bi_infer, bi_branch, bi_clash, bi_sol*
- Combinatorial Optimization Problems in CP(FD)
  - *bi_infer, bi_branch, bi_clash, bi_climb*
- Combinatorial Optimization Problems in IP B&B
  - *bi_branch, bi_clash, bi_bound, bi_opt*
- Combinatorial Optimization Problems in IP B&C
  - *bi_infer, bi_branch, bi_clash, bi_bound, bi_opt*

BASIS FOR INTEGRATION

- Branch and Infer allows to identify possible directions for the integration *[Kasper PhD98]*
  - First direction
    - Primitive constraints: Prim(ILP)
    - Non primitive constraints: N_Prim(ILP) and N_Prim(FD): restrict the inference of FD constraints to bound reductions
  - Second direction
    - Primitive constraints: Prim(FD)
    - Non primitive constraints: N_Prim(FD) and a **`linear`** constraint providing bounds and variable fixing
  - Third direction
    - Primitive constraints: Prim(ILP) and Prim(FD) excluding **`integer`**
    - Non primitive constraints: N_Prim(ILP) and N_Prim(FD) including the **`integer`** constraint

FINAL REMARKS on COMPARISONS

- There are no general guidelines to know in advance which technique is the most appropriate
- Unifying frameworks and problem class features can help in deciding the best technique
- The integration can lead to exploit advantages of both sides.

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - CSP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

INTEGRATION: MOTIVATIONS

- The main motivations for integrating modeling and solving techniques from AI and MIP are:
  - Combine the advantages of the two approaches
    - CSP: modelling capabilities, interaction among constraints
    - MIP: global reasoning on optimality, solution methods
  - Overcome the limitations of both
    - CSP: poor reasoning on the objective function
    - MIP: not flexible models, no symbolic constraints

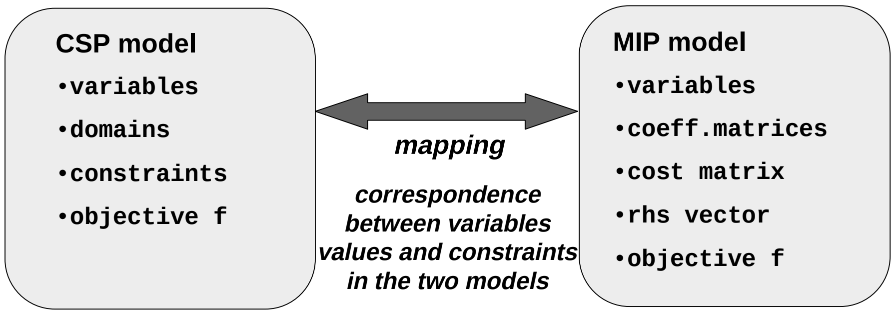
- Integration directions:
  - Problem modelling
  - Problem solving

INTEGRATION: MODELLING

- CSP have flexible, declarative, easily understandable models often described through CP languages
  - Different models for the same problem lead to different performances in finding solutions (expertise required)
  - Global constraints: powerful modelling abstractions
  - The addition of new problem constraints straightforward

- MIP provides less flexible mathematical models suitable for sophisticated problem solving techniques
  - Different models for the same problem lead to different performances in finding solutions (expertise required)
  - The addition of new problem constraints requires the re-definition of solution strategies

INTEGRATION: MODELLING



INTEGRATION: MODELLING

- CSP and MIP models should coexist, cooperate or even merge in a single language.

- Different levels of integration can be achieved:
  - approaches which provide the user only the CP language and hide the OR model and a mapping between the CP and OR models which is transparent to the user
  - approaches which provide the user both models and allow to state both constraints on the CP part and the OR part
  - approaches which merge the two models in order to provide a unique model embedding both the OR and CP side

TRANSPARENT MODEL INTEGRATION

- The user works with a Constraint Programming language
- The mapping and the MIP model are hidden
- Need of a translation of the CP model in terms of MIP model *[Rodosek, Wallace,Hajian AnnalsOR98, Refalo CPAIOR00]*
  - automatic transformation of CLP programs in non-disjunctive form through auxiliary binary variables
    - optimization on the number of binary variables
  - mapping of global constraints
- After the translation
  - Model = (CP store, MIP store)

Possible duplication: same constraint to both solvers

handled by the CP solver

handled by the MIP solver

TRANSPARENT MODEL INTEGRATION

- *[Rodosek, Wallace,Hajian 98]* automatic transformation of CLP programs in non-disjunctive form through auxiliary binary variables

  $\mathtt{p(Args_1)\text{:-}\ q_{11}(Args_{11}),\ldots.,q_{1k}(Args_{1k}).}$

  …

  $\mathtt{p(Args_r)\text{:-}\ q_{r1}(Args_{r1}),\ldots.,q_{rk}(Args_{rk}).}$

  - translated in

    $\mathtt{p(Args,B)\text{:-}\ p_1(Args,B_1),\ldots,p_r(Args,B_r),B_1+\ldots+B_r=B.}$

    $\mathtt{p_1(Args_1,B_1)\text{:-}\ q_{11}(Args_{11}),\ldots,q_{1k}(Args_{1k}),binary(B_1).}$

    ...

    $\mathtt{p_r(Args_r,B_r)\text{:-}\ q_{r1}(Args_{r1}),\ldots,q_{rk}(Args_{rk}),binary(B_r).}$

  - If one $\mathtt{q_{r1}(Args_{r1})}$ is a linear constraint $\mathtt{X} \le \mathtt{Y}$ is translated in $\mathtt{X + M*B} \le \mathtt{Y + M}$ with $\mathtt{M}$ large enough

TRANSPARENT MODEL INTEGRATION

- Automatic translating constraints

$$Var::[minV..maxV] \Leftrightarrow binary(B_{minV}),\ldots,binary(B_{maxV})$$
$$B_{minV} + \ldots + B_{maxV} = 1$$

if $Var_i = j$ the corresponding $B_{ij} = 1$

$$alldifferent([V1,\ldots,Vn]) \Leftrightarrow B_{11} + \ldots + B_{n1} \leq 1$$
$$\ldots\ldots$$
$$B_{1k} + \ldots + B_{nk} \leq 1$$

TRANSPARENT MODEL INTEGRATION

- Disjunction and inequality constraints

$X \neq Y$
$X::[min_x..max_x]$
$Y::[min_Y..max_Y]$

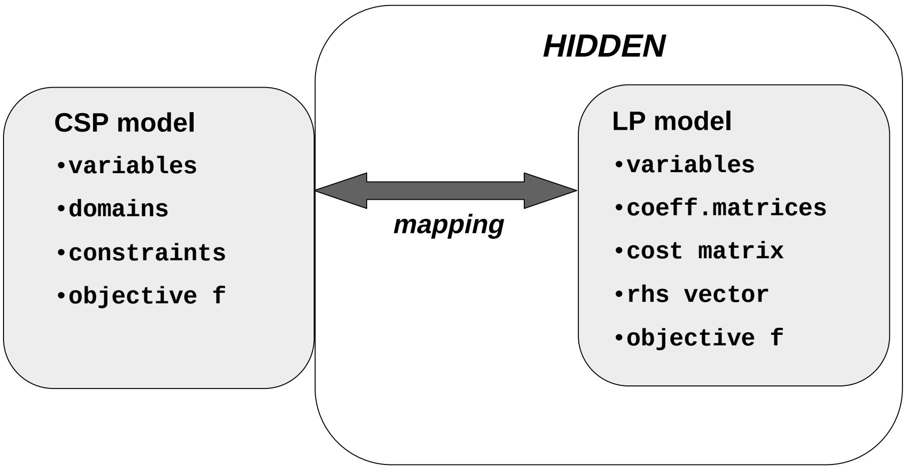
- or equivalently

$X < Y \vee X > Y$
$X::[min_x..max_x]$
$Y::[min_Y..max_Y]$

$\Leftrightarrow$

$$X + B_1M - Y \leq M - 1$$
$$Y + B_2M - X \leq M - 1$$
$$B_1 + B_2 = 1$$
$$min_Y \leq Y \leq max_Y$$
$$min_X \leq X \leq max_X$$

INTEGRATION: MODELLING



TRANSPARENT TRANSLATION OF GLOBAL CONSTRAINTS

- The user works with a CP language
  - global constraints embed a linear model *[Focacci,Lodi,Milano CP99, Regin CP99]* which allows to perform a propagation based on costs
  - same translation as before, inside the constraint not for the whole problem

Var::[minV..maxV] $\quad B_{minV} + \ldots + B_{maxV} = 1$

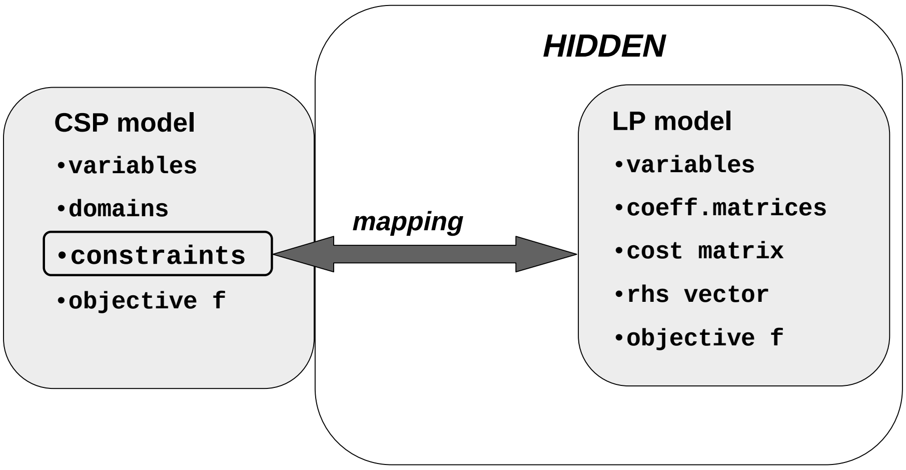
if $Var_i = j$ **the corresponding** $B_{ij} = 1$

alldifferent([V1,…,Vk]) $\quad B_{11} + \ldots + B_{n1} \leq 1$

……..

$B_{1k} + \ldots + B_{nk} \leq 1$

INTEGRATION: MODELLING



EXPLICIT MODEL COOPERATION

- The user explicitly states constraints to the two solvers. More complex programs. *[Beringer, De Backer 95] [Heipcke PhD99]*
- The user does not control the mapping between the two models which is in general achieved through shared variables
- In the same program:

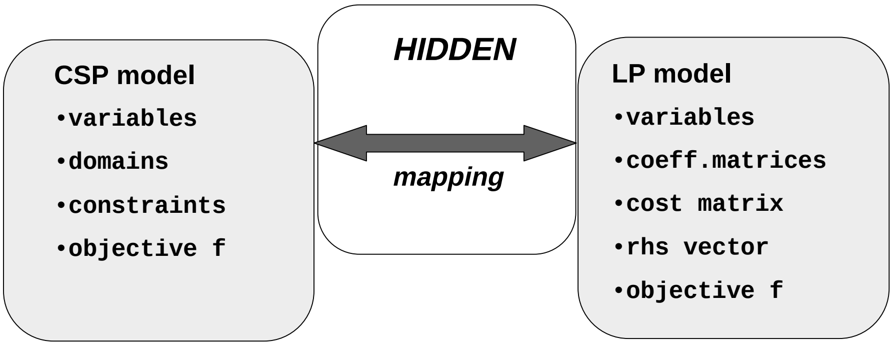
```
Vars::Domains,
constraints_CP(Vars),
Objective :: DomObj,
link(Objective, Vars)
```

```
Vars ≥ 0,
linear_inequal(Vars),

link(Objective, Vars)
```

Same variables involved in different constraints

INTEGRATION: MODELLING



UNIQUE MODEL

- The user designs a unique model suitable for the hybrid solver: MLLP: Mixed Logical/Linear Programming *[Hooker et al. AAAI99]*
- More complex model but closer to the hybrid architecture

min $z = c\,x$

– *subject to*

$h_i(y) \rightarrow A_i\,x \leq b_i$

$x_i \in R^n$ $\quad y \in FD$

Linear constraints added to the linear solver when the finite domain constraint is entailed by the FD store

Constraints handled by the CP finite domain solver

Constraints handled by the linear solver

UNIQUE MODEL: GLOBAL CONSTRAINTS

- Global constraints can be introduced in MLLP: example on Variable Subscripts in Linear Constraints *[Ottoson,Thorsteinsson CPAIOR00]*

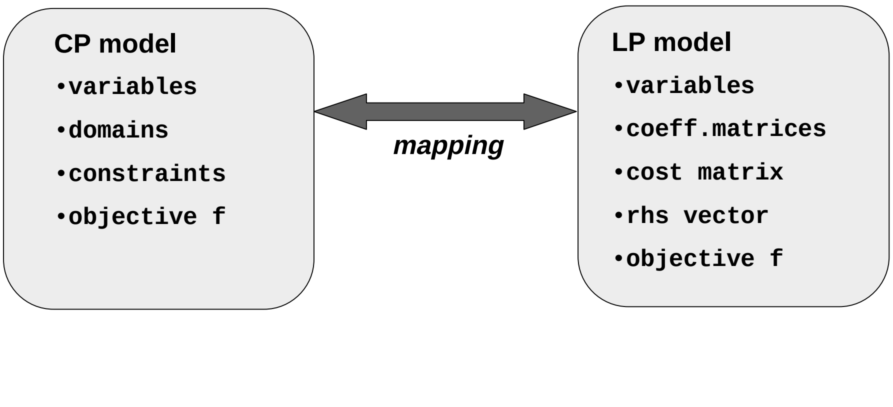
- $z \geq \Sigma_j \; c_{j\,y_j}$ sum of costs of assigning worker y_j to job j
- $c_{j\,y_j}$ can be substituted by z_j $\quad z \geq \Sigma_j z_j$

  $y_j = k \;\rightarrow\; z_j = c_{j\,k} \quad \forall$ j $\in$ {1…n}, k $\in$ {1…m}

or alternatively $\quad z \geq \Sigma_j z_j$

element(y,[c_{11},…,c_{1n}],z1), element(y,[c_{21},…,c_{2n}],z2),
…………   , element(y,[c_{m1},…,c_{mn}),z_m).

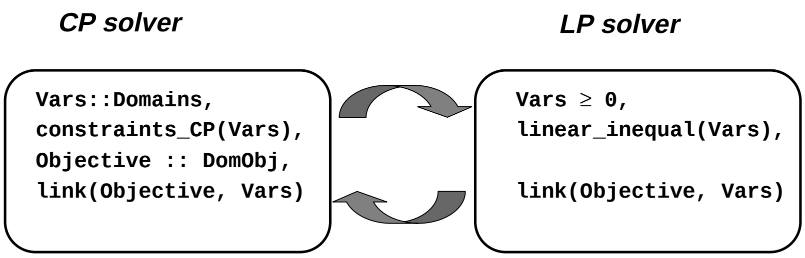
INTEGRATION: MODELLING



WHICH PARTS OF THE PROBLEM ?

- The mapping defines a correspondence between the CP model and the MIP (or LP) model.
- Which parts of the problem are involved ?
  - the whole problem is represented in both models
    - transparent model integration *[Rodosek, Wallace,Hajian AnnalsOR98]*
    - explicit model integration of the whole problem
  - only some parts of the problem are represented in both models:
    - translation of global constraints *[Refalo CP2000]*
    - LP/IP model within global costraints *[Focacci,Lodi,Milano CP99]*
    - explicit model integration of some parts of the problem
  - some parts of the problem are represented in the CP model and other parts in the IP/LP model
    - problem decomposition *[El Sakkout, Wallace Constraints 2000]*

TRANSLATION OF THE WHOLE PROBLEM
AUTOMATIC OR EXPLICIT TRANSLATION



Exchange information on the same entities (variables and objective function)

TRANSLATION OF PART OF THE PROBLEM
AUTOMATIC OR EXPLICIT TRANSLATION

For example: non linear parts are not translated

CP solver

```
Vars::Domains,
constraints_CP(Vars),
Objective :: DomObj,
link(Objective, Vars)
```

LP solver

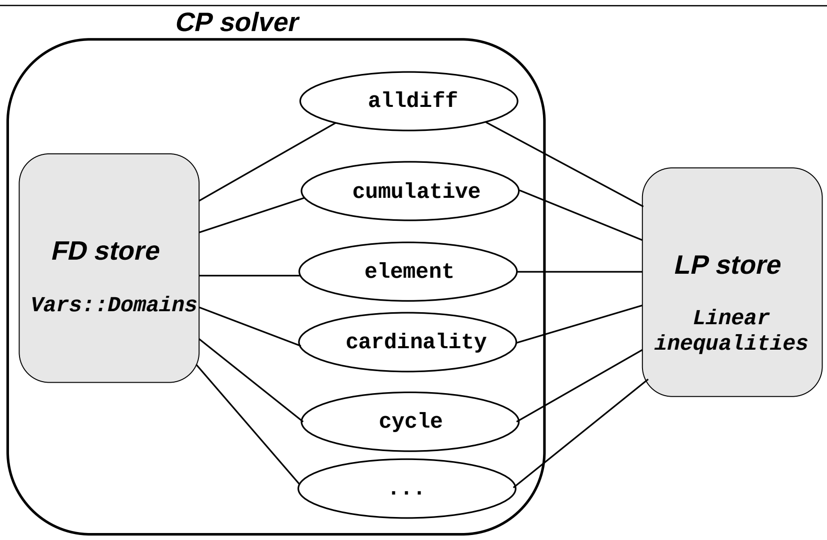
```
Vars’ ≥ 0,
linear_inequal(Vars’),

link(Objective’, Vars’)
```

Vars’ ⊂ Vars

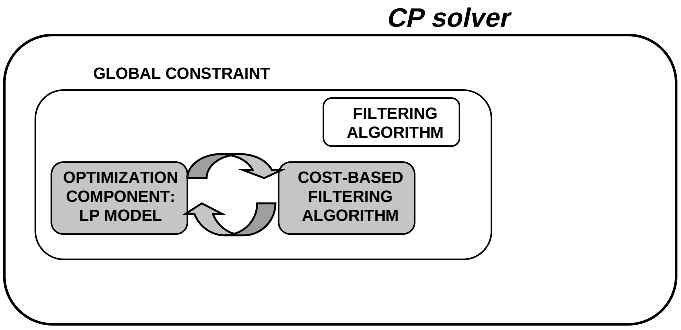
Exchange information on the common entities (variables and objective function)

TRANSLATION OF PART OF THE PROBLEM
TRANSLATION OF GLOBAL CONSTRAINTS



TRANSLATION OF PART OF THE PROBLEM
LP/IP MODEL WITHIN GLOBAL CONSTRAINTS

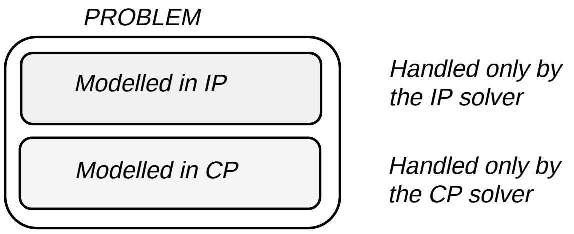
- The user works with a CP language
  - global constraints embed a linear model *[Focacci,Lodi,Milano CP99, Regin CP99]* which allows to perform a propagation based on costs



PROBLEM DECOMPOSITION

- If a problem can be decomposed in subproblems choose the best technique/solver to solve it. Interaction among subproblems should be handled
  - CP: subproblems are single constraints

- Application to Dynamic Scheduling *[El Sakkout, Wallace Constraints 2000]*



OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - CP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

INTEGRATION: SOLVING

- CP solving is based on interleaved search and inference (constraint propagation)
  - constraint propagation rules in global constraints exploit global problem-dependent knowledge to perform pruning
  - different constraints interact through shared variables
  - problem dependent branching strategies
- MIP solving is based on interleaved search and the computation of the optimal solution of a relaxation
  - pruning is performed on nodes for which the relaxation is infeasible or proven sub-optimal
  - branching is guided by relaxation information (relaxation provides a point in space where search should be centered)

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - CP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
    - *Search*

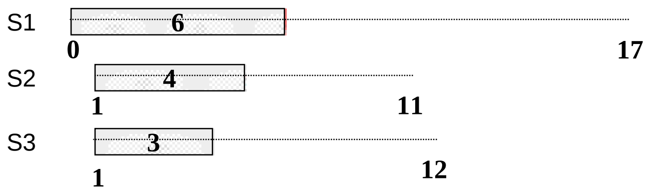
INTEGRATION: SOLVING

- Integration on solving: two aspects
  - feasibility
  - optimality
- Concerning feasibility, global constraints embed powerful filtering algorithms that prune the search space
- Many of them coming from OR
  - Edge Finder
  - Network flow based algorithms

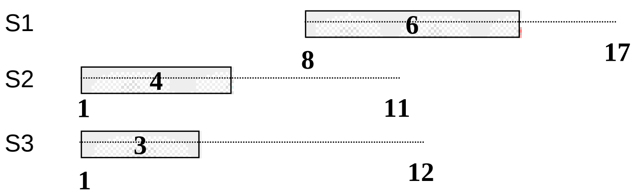
EDGE FINDER

- Useful for scheduling application *[Carlier Pinson 90] [Baptiste, Le Pape, Nuijten, IJCAI95]*

Consider a unary resource and three activities.



EDGE FINDER



We can deduce that earliest start time of S1 is 8.

This is based on the fact that S1 must be scheduled after S2 and S3.

Global reasoning: suppose either S2 or S3 is scheduled after S1. Then the maximum of the completion times of S2 and S3 is at least 13 (out of the domain of S2 and S3).

EDGE FINDER

Basic Theorem: [*Carlier, Pinson, Man.Sci.95*]

Let o be an activity and S a set of activities all to be scheduled on the same unary resource (o not in S). The earliest start time is e, the sum of durations is D and the latest completion time C. If

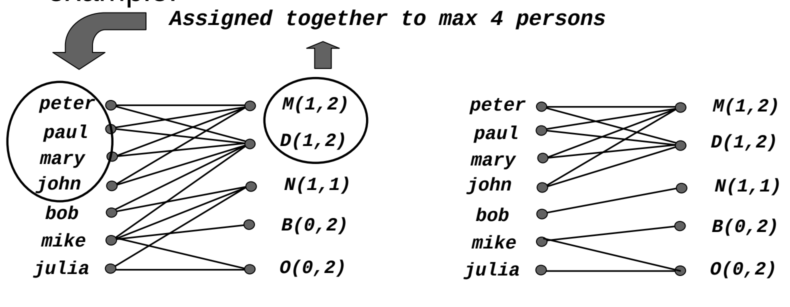
$$e(S+\{o\}) + D(S+\{o\}) > C(S)$$

then no schedule exists in which o precedes any of the operations in S. This implies that the earliest start time of o can be set to

$$\max_{(S' \subseteq S)} \{e(S') + D(S')\}.$$

GLOBAL CARDINALITY CONSTRAINT

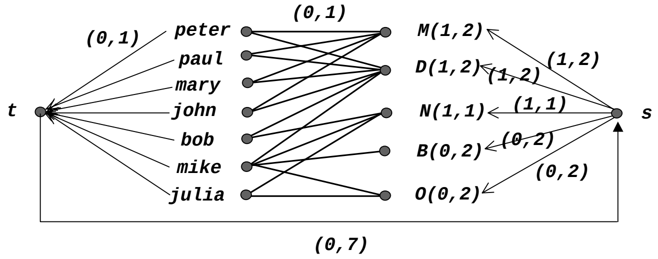
- **`gcc(Var,Val,LB,UB)`** *[Regin AAAI96]* **`Var`** are variables, **`Val`** are values and **`LB`** and **`UB`** are the minimum and the maximum number of occurrences of each value in **`Val`** assigned to **`Var`**
- example:



GLOBAL CARDINALITY CONSTRAINT

- Notion of consistency based on *network maximum flow* algorithm on the value network N(C)

> - **gcc on k variables is consistent**
> - **there is a max flow from s to t of value k**



GLOBAL CARDINALITY CONSTRAINT

- Filtering algorithm based on *network maximum flow* algorithm on the residual graph R w.r.t. flow f: R(f)
  - if **f(u,v) < ub(u,v)** then **res(u,v) = ub(u,v) - f(u,v)**
  - if **f(u,v) > lb(u,v)** then **res(v,u) = f(u,v) - lb(u,v)**
  - the residual capacity of a path**: min res(u,v) in path**

- Let C a consistent gcc and f a maximum flow on N(C) from s to t. A value a for x is not consistent with C iff f(a,x)=0 and a and x do not belong to the same strongly connected component in R(f) - {(s,t)}

GLOBAL CONSTRAINTS

- References:
  - Global constraints in CHIP ***[Beldiceanu Contejean, Math.Comp.Mod.94]***
  - Task Intervals ***[Caseau Laburthe ICLP94] [Caseau Laburthe LNCS1120] [Caseau Laburthe JICSLP96] [Caseau Laburthe LNCS1120]***
  - alldifferent ***[Regin AAAI94]*** Symmetric alldifferent ***[Regin IJCAI99]***
  - Edge Finder ***[Baptiste Le Pape Nuijten, IJCAI95], [Nuijten Le Pape JoH98] [Nuijten Aarts Eur.J.OR96]***
  - Sequencing ***[Regin Puget CP97]***
  - Cardinality ***[Regin AAAI96]***
  - Sortedness ***[Bleuzen Colmerauer Constraints 2000]***

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - CP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

INTEGRATION: SOLVING

- Optimization Problems

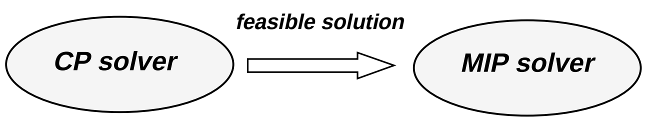
- CP and MIP/LP solvers should interact and cooperate by exchanging information

- Different levels of integration can be achieved:
  - approaches which keep the two solvers separate and independent exchanging information
    - sequential computations
    - interleaved computations
  - approaches which integrate an optimization component (LP solver) in global constraints

LOOSE INTEGRATION
SEQUENTIAL COMPUTATION

- CP and MIP solvers are used in sequence:
  - the CP solver computes the first feasible solution (rather quickly)
  - the MIP solver uses this solution as warm start for a Simplex based Branch & Bound

- Approach used in the British Airways Fleet Assignment *[Hajiaan et al. TR95/09-01]*



TIGHTER INTEGRATION
INTERLEAVED COMPUTATION

- CP and LP solvers are interleaved:
  - the CP solver performs propagation
  - LP solver optimally solves the linear relaxation

- CP and LP solvers exchange information:
  - variable bounds
  - variable fixing
  - optimal solution of LP/reduced costs

- Branching performed on the CP or on the MIP side
  - CP branching based on problem structure or on var. domains
  - MIP branching based on optimal solution of the relaxation

TIGHTER INTEGRATION
INTERLEAVED COMPUTATION

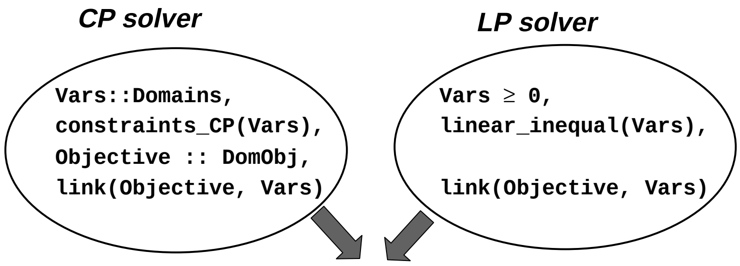
- Solving technique for modelling approaches where the CP and the LP model are kept separate

  - CASE A: the user can explicitly impose which constraints are handled by the CP solver and which ones are handled by the LP solver *[Beringer-De Backer95] [Heipcke PhD99]*

  - CASE B: the automatic linearization of global constraints is sent to the LP solver while the propagation of global constraints is performed as usual by the CP solver *[Refalo CP2000, Focacci Lodi Milano CP99]*

TIGHTER INTEGRATION

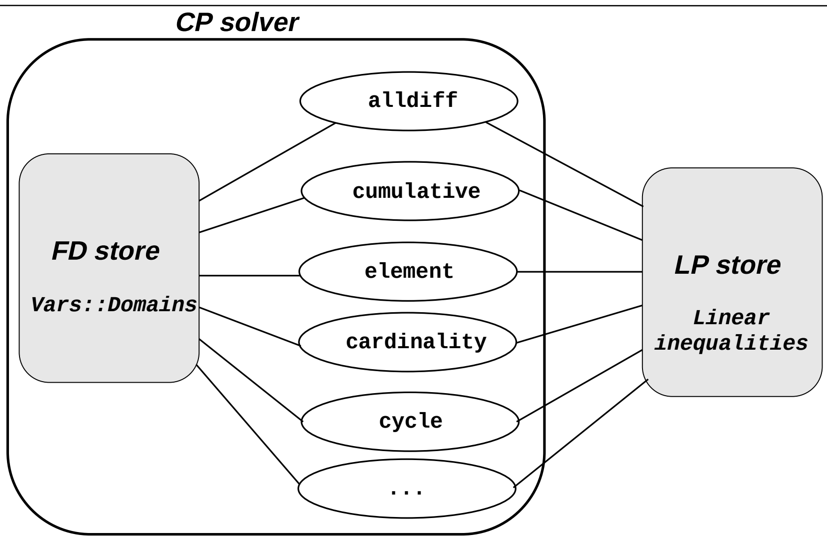
INTERLEAVED COMPUTATION: CASE A



Same variables involved in
different constraints
Events trigger propagation/recomputation

TIGHTER INTEGRATION

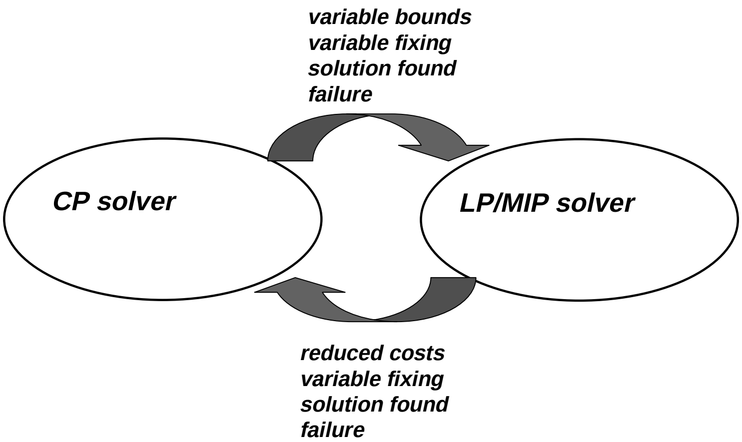
INTERLEAVED COMPUTATION: CASE B



TIGHTER INTEGRATION
INFORMATION EXCHANGES

- Results of the CP solver inference
  - domain bound reduction
  - domain value removal
  - variable instantiation
  - solution found
- Results of the LP solver inference
  - variable fixing
  - solution found (relaxed problem)
  - reduced costs

TIGHTER INTEGRATION
INFORMATION EXCHANGES



TIGHTER INTEGRATION
EXPLOITATION OF RESULTS

- Where results of the CP solver can be used
  - domain bound reduction ⟹ sent to the LP solver which adds the new bounds to the problem formulation (cut)
  - domain value removal ⟹ the corresponding binary variable fixed to 0
  - variable instantiation ⟹ the corresponding binary variable fixed to 1
  - solution found ⟹ upper bound available

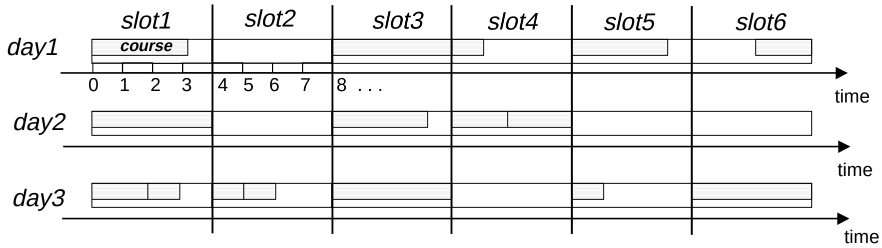
TIGHTER INTEGRATION
EXPLOITATION OF RESULTS

- Where results of the LP solver can be used
  - variable fixing
    - to 0 ⟹ corresponding value removed
    - to 1 ⟹ corresponding variable instantiated
  - reduced costs ⟹ domain filtering
  - solution found ⟹ lower bound on the objective function variable

Suggestions for guiding the search more on this later...

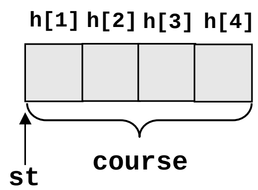
EXAMPLE

- Timetabling problem from *[Caseau Laburthe CP97]*
  - 4-Hours Slots - 1 to 4 Hours Courses
  - Two courses cannot overlap
  - A course must be contained in a single slot
  - Preferences are associated with: Course-Slot assignments
  - Maximize Sum of Preferences



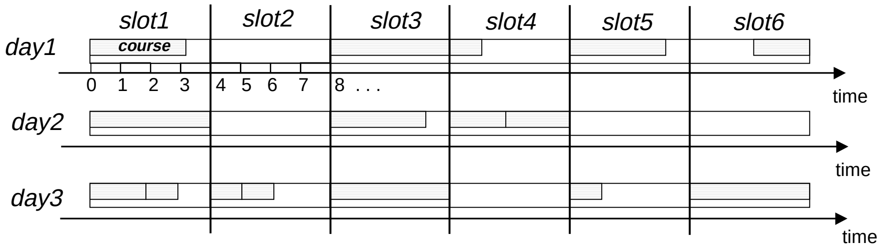
EXAMPLE

- Variables associated to each course lasting **`Nh`** hours
  - Start time **`st`**: domain contains possible starting time for the course
  - Single hours **`h[i] i=1..Nh`**: domain contains hours along the time line
  - Course **`course`**: domain contains slots
- Constraints link the different variables



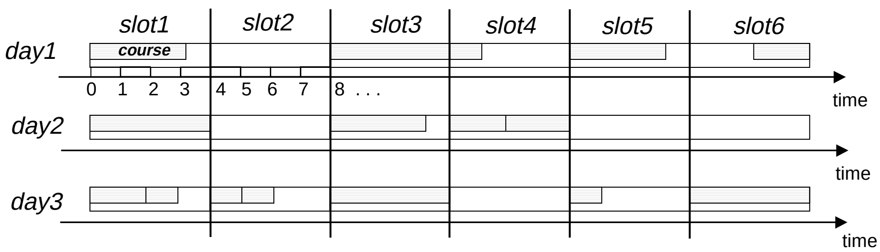
EXAMPLE

- Course lasting **`Nh=3`** hours
  - **`st::[0,1,4,5,8,9,…]`**
  - **`h[1]::[0,1,4,5,8,9,…]  h[2]::[1,2,3,4,9,10,…]`**
  - **`h[3]::[2,3,6,7,10,11,…]`**
  - **`course::[day1slot1,day1slot2,…,day2slot1,…,daynslotm]`**



EXAMPLE

- Constraints
  - **`TimeTable(startVars, durations);`**
  - **`// relaxing the contiguity constraint`**
  - **`AllDiffCost(singleHours,objective,costsMtx);`**
  - **`// subproblem defined by 3 and 4 hour courses`**
  - **`AllDiffCost(courses34Hours,objective1,costsMtx1);`**



MODEL MAPPING

`AllDiffCost(singleHours,objective,costsMtx);`

- `singleHours`: array of `SumOfDurations` variables whose domain contains single hours

`singleHours[i]=j` $\longleftrightarrow$ $x_{ij}=1$

`singleHours[i]`$\neq$`j` $\longleftrightarrow$ $x_{ij}=0$

`costMtx[i][j]` $\longleftrightarrow$ c_{ij}

`costMtx[i][j]` = ∞ if $x_{ij}=0$

`costMtx[i][j] = pref[i][j]/dur`

ILP-Model

$$\min z = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x_{ij}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1..n \quad (A)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1..n \quad (B)$$

x_{ij} integer

↓ ↓

LB ***reduced costs***

INTERACTIONS AMONG SOLVERS

- LB-based Propagation

  from LB towards objective function $Z::[Z_{min}..Z_{max}]$

  $\boxed{LB \leq Z}$

- Reduced Cost-based Propagation

  from reduced costs towards decision variables

  $X_i::[i_1,i_2,\ldots,i_m]$ each i_j has a gradient function $grad(X_i,i_j)$ measuring the cost to pay if $X_i = i_j$

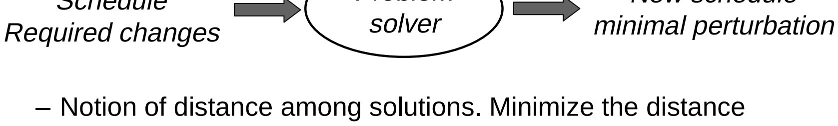
  $\boxed{\text{if } LB + grad(X_i,i_j) \geq Z_{max} \text{ then } X_i \neq i_j}$

TRIGGERING EVENTS

- In CP constraint propagation is triggered each time the domain of one variable appearing in it is modified
  - Variable domains are changed both due to
    - other constraint propagation
    - variable fixing from the linear solver
    - reduced cost based propagation
  - In particular reduced cost based propagation triggered when LP computes a new solution or when the upper bound of the objective function changes
- LP solver triggered each time a value in the solution of the LP is deleted from the variable domain

TIGHTER INTEGRATION
BASED ON PROBLEM DECOMPOSITION

- If a problem can be decomposed in subproblems choose the best technique/solver to solve it. Interaction among subproblems should be handled
  - CP: subproblems are single constraints
- Application to Dynamic Scheduling *[El Sakkout, Wallace Constraints 2000]*



  - Notion of distance among solutions. Minimize the distance

TIGHTER INTEGRATION
BASED ON PROBLEM DECOMPOSITION

- Example on Dynamic Scheduling *[El Sakkout, Wallace Constraints 2000], [El Sakkout,PhD99]:* possible changes
  - Temporal constraints (e.g., distance between activities)
  - Activity constraints (e.g., changing the set of activities, duration, required resources)
  - Resource constraints (e.g. reductions in resource availability)
  - Piecewise constraints (considered in *[Ajili, El Sakkout CPAIOR2001])*

Temporal subproblem: totally unimodular ⇒ LP provides optimal integer solutions satisfying temporal constraints and minimizing the differences to the given (temporally inconsistent solution)

Activity subproblem: same feature after a transformation

TIGHTER INTEGRATION
BASED ON PROBLEM DECOMPOSITION

- Temporal and Activity subrpoblem are solved via LP

- Remaining violations: resource utilization
  - aim: reduce contention in the constraints
  - contention exists when the resource used at a given time point exceeds the limited (modified) capacity
  - degree of contention: the difference

- Search interleaved with LP re-optimization
  - select a set of constraints subject to contention
  - select a decision with minimal impact
  - *…more on this later*

TIGHTER INTEGRATION
BRANCH AND CHECK

- Recent framework *[Thorsteinson CP2001]* generalizing problem decomposition
- Branch and check heavily relies on Benders Decomposition
- *Idea: a part of the problem is considered BASIC*
  the remaining part is considered DELAYED

- *Branch and Check is based on a branching search on the basic part. The delayed part is checked as late or as seldom as possible*
  - *min* $cx + f(x)$
  - *s.t* $A\,x \leq b$
  - $H(x,y)$ → *non linear part + mapping*

TIGHTER INTEGRATION
BRANCH AND CHECK

- Completely ignoring the delayed part does not work: a relaxation of the delayed part is added to the basic model.
  - If the non linear part is a CSP model with alldifferent or piecewise linear constraints, the linearization of these constraines should be added to the basic part.

- When a delayed part is solved, bounding cuts are added to the model

- *Benders decomposition has been recently introduced in a Constraint Programming language ECLiPSe [Eremin and Wallace CP2001]*

ADVANTAGES OF THE INTEGRATION

- Relaxation and inference combined
  - Constraint propagation meets Linear Programming
- Relaxation of overlapping constraint: yet another channel of communication among subproblems
  - example of timetabling two alldifferent constraint providing two Linear Relaxations in the same problem
- Integration of different optimization constraints through Lagrangian Relaxation *[Sellman, Fahle CP-AI-OR01]*
- LP store is a separate means of interaction

SPECIAL PURPOSE ALGORITHMS

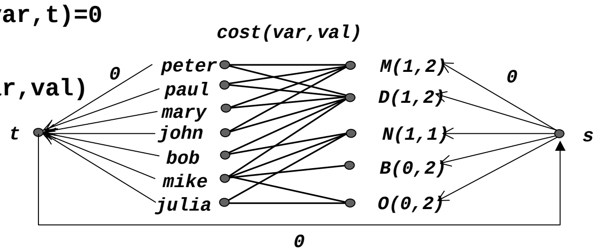
- Instead of using an LP solver, we can embed in global constraints special purpose algorithms when the linear relaxation of the constraint represents a well structured problem
- Characteristics of the algorithm
  - polynomial time complexity
  - incremental behaviour
  - should provide the optimal solution of the relaxed problem
  - should (possibly) provide reduced costs for enhancing propagation

HUNGARIAN ALGORITHM FOR THE AP

- Primal-dual algorithm
  - starts with an initial solution where all variables are set to 0 which is optimal and not feasible for the primal
  - at each iteration, it looks for an alternating (possibly augmenting) path of zero cost. If it is augmenting a new assignment is performed, otherwise the value of dual variables is changed
- Incrementality
  - each time an arc in the current AP solution is removed, the algorithms looks for a single augmenting path.
- Complexity n nodes: $O(n^3)$ the first time $O(n^2)$ incrementally
- Reduced costs provided with no extra cost

FLOW ALGORITHMS FOR COST BASED FILTERING

- Consider the flow algorithm for the global cardinality constraint. Extension with costs *[Regin CP99]*
- **`gcc(Var,Val,LB,UB,Costs,H)`** same semantics of the gcc but with the sum of costs of assignments less or equal than **`H`**
- In the value network:
  - **`c(s,val)=0 and c(var,t)=0`**
  - **`c(t.s)=0`**
  - **`c(val,var)=cost(var,val)`**



FLOW ALGORITHMS FOR COST BASED FILTERING

- Notion of consistency based on *network minimum cost flow* algorithm on the value network N(C)

> - **gccCost on k variables is consistent**
> - **there is a min cost flow in N(C) <= H**

FLOW ALGORITHMS FOR COST BASED FILTERING

- Filtering algorithm based on the redisual graph R w.r.t. flow f: R(f) plus costs
  - if **f(u,v)<ub(u,v)** then **res(u,v)=ub(u,v)-f(u,v);c(u,v)=cost(u,v)**
  - if **f(u,v)>lb(u,v)** then **res(v,u)=f(u,v)-lb(u,v);c(v,u)=cost(u,v)**
  - x^0 optimal solution of minimum flow in N(C)
  - potential of each node $\pi(i)$
  - reduced cost $c_{ij}^{\pi}=c(i,j)-\pi(i)+\pi(j)$
  - $d_{i,j}(k)$ shortest path distance from i to k in R(x^0)-{(i,j)}
    - Let C a consistent gcc + costs. A value a for y is not consistent with C iff
      - $x^0(a,y) = 0$ **OR**
      - $d_{y,a}(a) > H - cost(x^0) - c_{ay}^{\pi}$

GLOBAL CONSTRAINTS for OPTIMIZATION

- References:
  - Global constraint for TSP ***[Caseau Laburthe ICLP97] [Focacci Lodi Milano Elect.Notes on DM 99], TSPTW [Focacci Lodi Milano ICLP99]***
  - Matching problems ***[Caseau Laburthe CP97] [Focacci Lodi Milano CP-AI-OR 99]***
  - Reduced cost fixing in global constraints ***[Focacci Lodi Milano CP99]***
  - Cardinality constraints + costs ***[Regin CP99]***

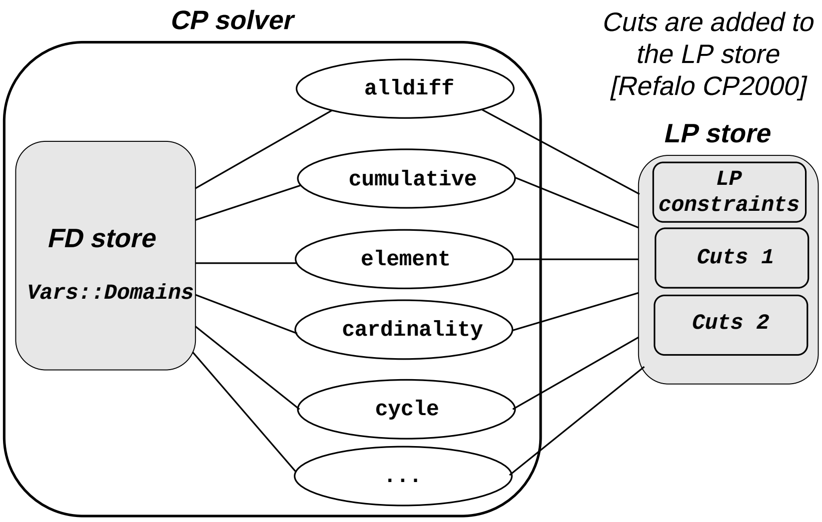
OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - CP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

CUTTING PLANES IN CP

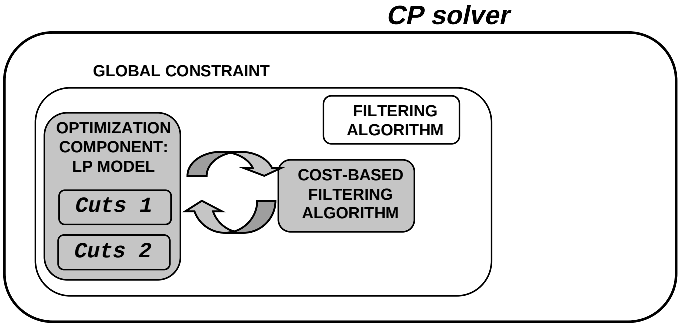
- When the LP is used, the linear relaxation of constraints can be enhanced with the addition of cutting planes
- New LP, called LP^{cut} which provides:
  - more precise bound: $Sol(LP^{cut}) \geq Sol(LP)$
  - reduced costs
- Different ways of adding cuts to LP formulation:
  - at the root node only in order to restrict the initial LP formulation
  - at the root node are relaxed in a lagrangian way in order to obtain a structured problem
  - at each node (global or local cuts)

CUTTING PLANES IN CP



CUTTING PLANES IN CP

Cuts are added within each constraint [Focacci, Lodi, Milano CP2000]



EXAMPLE: `cycle` CONSTRAINT

- Relaxation of the constraint: Assignment Problem
  - the AP finds a set of (possibly) disjoint subtours that minimizes the sum of costs
  - the CP optimal solution satisfies the integrality constraints
- Sub-tour constraints are relaxed
- Sub-tour elimination cuts (SECs) can be separated in polynomial time *[Padberg, Rinaldi 90]*
  - find the subtour polytope: in general provides a fractional solution where no subtour is present

EXAMPLE: **cycle** CONSTRAINT

CP-Model:

$X_i::[i_1, i_2, \ldots, i_m]$ $i=1..n$

$\text{cycle}([X_1, X_2, \ldots, X_n])$

$C_i::[c_{i1}, c_{i2}, \ldots, c_{im}]$ $i=1..n$

$C_1+\ldots+C_n = Z$

$\text{minimize}(Z)$

Mapping

ILP-Model

$$\min z = \sum_{i\in V}\sum_{j\in V} c_{ij}\, x_{ij}$$

$$\sum_{i\in V} x_{ij} = 1 \quad j\in V \text{ (A)}$$

$$\sum_{j\in V} x_{ij} = 1 \quad i\in V \text{ (B)}$$

$$\sum_{i\in S}\ \sum_{j\in V\setminus S} x_{ij} \geq 1 \quad S\subset V \ \ S\neq\emptyset$$

x_{ij} integer

$X_i = i_j \longleftrightarrow x_{ij}=1$

$\text{cycle}([X_1, X_2, \ldots, X_n]) \xleftrightarrow{\textit{relaxed in}} \text{(A) + (B)}$

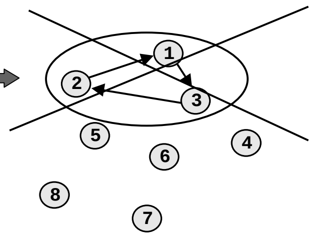
$C_i=c_{ij} \longleftrightarrow c_{ij}$ in z

EXAMPLE: **cycle** CONSTRAINT

- Addition of cuts at the root node
  - computed cuts are added to the LP and the new problem is solved through the linear solver along the whole search tree
  - computation of cuts is efficient
  - drawback: cuts can be no longer effective during the search

  - Example: subtour elimination cut

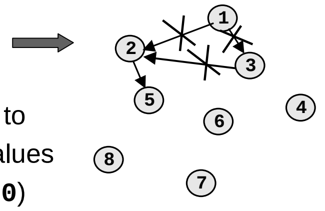
$$x_{21}+x_{12}+x_{31}+x_{13}+x_{32}+x_{23} \leq 2$$



EXAMPLE: **`cycle`** CONSTRAINT

– Example: subtour elimination cut

$$x_{21}+x_{12}+x_{31}+x_{13}+x_{32}+x_{23} \leq 2$$



when during search **X2** is assigned to **5** ($x_{25}=1, x_{21}=0, x_{23}=0$), from **X1** values **2** and **3** are removed ($x_{12}=0$, $x_{13}=0$) from **X3** values **2** is removed ($x_{32}=0$) the cut is satisfied (with <) and no longer needed

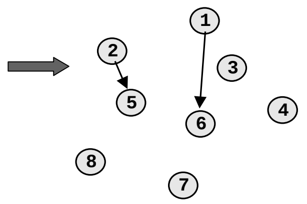
The new problem is LP^{cut}
- – $Sol(LP^{cut}) \leq Sol(LP)$
- – when all cuts are satisfied $Sol(LP^{cut}) = Sol(LP)$

EXAMPLE: **`cycle`** CONSTRAINT

- Alternative: generate cuts during search
  - – at each node compute valid cuts and add to the LP
  - – subtour elimination cuts are globally valid
  - – can be removed upon backtracking

– Example: subtour elimination cut

$$x_{21}+x_{12}+x_{31}+x_{13}+x_{32}+x_{23} \leq 2$$



when during search **X2** is assigned to **5** and **X1** is assigned to **6** the cut is trivially satisfied but another can be computed

$$x_{25}+x_{26}+x_{21}+x_{52}+x_{56}+x_{51}+x_{62}+x_{65}+x_{61}+x_{16}+x_{15}+x_{12} \leq 3$$

EXAMPLE: **`cycle`** CONSTRAINT

- Addition of cuts at the root node in the lagrangian relaxation
  - motivation: if we have a special purpose algorithm to solve the LP, say the Hungarian Algorithm, we cannot simply add cuts to the LP since the problem structure would be lost
  - computed cuts are added to the LP which is optimally solved
  - dual values associated to cuts: optimal Lagrangian multipliers λ
  - relax cuts in a lagrangian way

$\boxed{\text{LP}^{cut} \text{ has the same structure than the original LP}}$

EXAMPLE: **`cycle`** CONSTRAINT

- Generated cut $\alpha\mathbf{x} \leq \alpha_0$
- Initial Linear Problem LP (special structure)
  - **min** $z = \sum c_j x_j$
  - **subject to**

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad i = 1..m$$

$$x_j \geq 0 \qquad j = 1..n$$

$$\sum_{j=1}^{n} \alpha_{mj} x_j \leq \alpha_{0m} \qquad m = 1..k$$

Changes the problem structure

Dual values correspond to optimal lagrangian multipliers

EXAMPLE: **`cycle`** CONSTRAINT

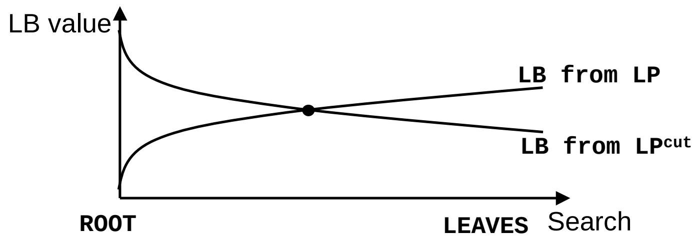
- Initial Linear Problem LP (special structure)
  - $\textbf{min} \quad z = \sum_{j=1}^{n} c_j\, x_j + \lambda \left(\sum_{j=1}^{n} \alpha_{mj}\, x_j - \alpha_{0m} \right)$ ⟸ Cost Matrix changes
  - **subject to** ⟸ Same problem structure

$$\sum a_{ij}\, x_j = b_i \qquad i = 1..m$$
$$x_j \geq 0 \qquad j = 1..n$$

- Drawback: cuts that are no longer effective during the search introduce a penalty that produces worst lower bounds $\sum_{j=1}^{n} \alpha_{mj}\, x_j - \alpha_{0m} < 0$

EXAMPLE: **`cycle`** CONSTRAINT

- LB behaviour for LP and LP^{cut}



- Purging techniques: remove non needed cuts

LOCALLY VALID CUTS

- Cuts added during search can exploit information on variable domains.
- Tighter integration of cutting planes in CP *[RefaloCP99]*
  - Definition of the convex hull of the LP + variable domain bounds.

Cartesian product of domains

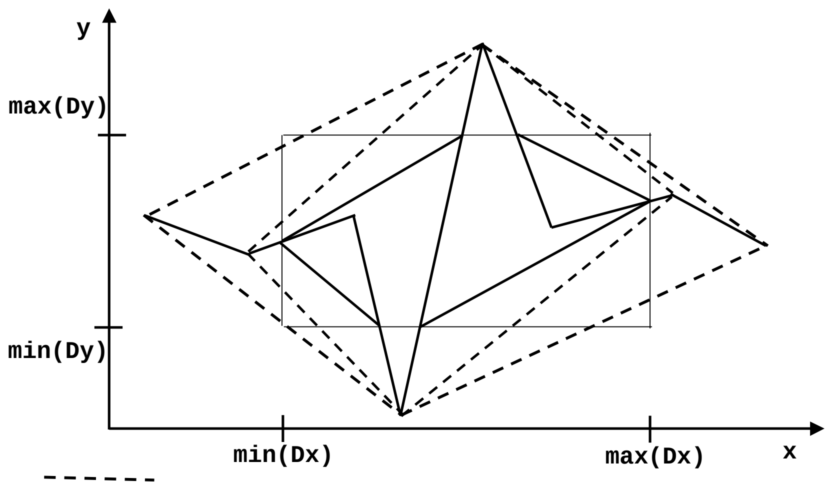
Solution set for c

$$T = \text{convex\_hull}(D_X) \cap \{ \bigcap_{c \in P} \text{convex\_hull}(S(c)) \}$$

$$T' = \bigcap_{c \in P} \text{convex\_hull}(S(c) \cap D_X) \Rightarrow$$

When added during search depend on constraint propagation *LOCALLY VALID*

EXAMPLE: **`piecewise`** linear function



EXAMPLE: **`piecewise`** linear function

- Generated Cuts are facets of the convex hull
  - from **`(min(Dx),f(min(Dx))`** to **`(u,f(u))`** **u**∈**Dx, f(u)**∈**Dy** and the slope **s1** is *maximal*
  - from **`(min(Dx),f(min(Dx))`** to **`(u,f(u))`** **u**∈**Dx, f(u)**∈**Dy** and the slope **s2** is *minimal*
  - from **`(max(Dx),f(max(Dx))`** to **`(u,f(u))`** **u**∈**Dx, f(u)**∈**Dy** and the slope **s3** is *maximal*
  - from **`(max(Dx),f(max(Dx))`** to **`(u,f(u))`** **u**∈**Dx, f(u)**∈**Dy** and the slope **s4** is *minimal*

- Other 4 cuts for **Dy**

- Locally valid cuts must be removed in backtracking

LOGICAL CUTS

- Hooker recognized the equivalence of cutting plane techniques with resolution for propositional logic
  - linear inequalities can be seen as clauses
  - valid cuts can be derived by logical inference: a cut is nothing other than a logical implication of the constraint set.

$4x_1+2x_2+x_3 \geq 3$ ***equivalent*** ⟺ $x_1 \vee x_3$, $x_1 \vee x_2$

x_1	x_2	x_3	Value
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

In fact the linear inequality is satisfied iff x_1 is set to 1 or x_2 and x_3 are both 1

LOGICAL CUTS

- Resolution provides logical cuts:

$x_1 + x_2 + x_3 \geq 1$
$(1-x_1) + x_2 + (1-x_4) \geq 1$

$x_1 \vee x_2 \vee x_3$
$\neg x_1 \vee x_2 \vee \neg x_4$

$x_2 + x_3 + (1-x_4) \geq 1$ **equivalent** $x_2 \vee x_3 \vee \neg x_4$

Rank 1 cut *Resolvent*

- Connections between k-resolution (k-rank cuts) and k-consistency
  - k-resolution generates all resolvent of less than k literals not already absorbed by other clauses

LOGICAL CUTS IN CLP(PB)

- Integration of logical cuts in CLP(PB) *[Barth Bockmayr ICLP95]* where variables are finite domains and range on [0,1]
  - interpret a CLP(PB) program as an Integer Problem in the form $Ax \leq b \quad x \in \{0,1\}^n$. Then consider the linear relaxation LP
  - add valid inequalities

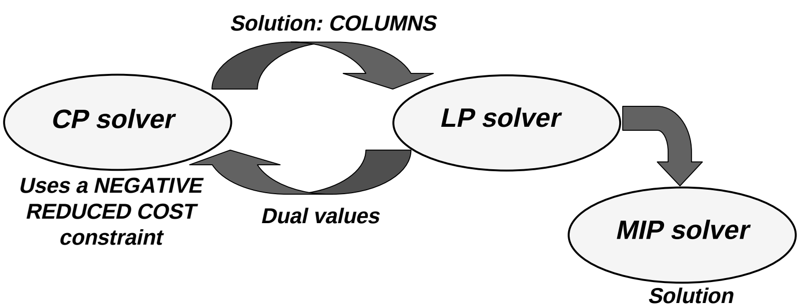
- Advantages:
  - constraints are translated in a solved form
  - entailment decided earlier
  - unfeasibility can be detected before enumeration
  - use of lower/upper bounds

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - CP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

COLUMN GENERATION and CP

- General Framework *[Junker et al. CP99]*
- Master Problem: MIP
- Subproblem: CSP



COLUMN GENERATION and CP

```
V':=getInitialColumns()
repeat
   λ:=solveLP(V')
  {xj_1,…,xj_k}:=solveSubProblem(λ)
  V':= V'∪ {xj_1,…,xj_k}
until {xj_1,…,xj_k} = ∅
```

- Applied to crew rostering application *[Junker et al.CP99]*
  - Path constraint based on set variables uses dynamic programming techniques for propagation
  - uses shortest path algorithm for Acyclic graphs
- Advantage of using CP: the subproblem can be formulated by considering many problem dependent constraints

OVERVIEW

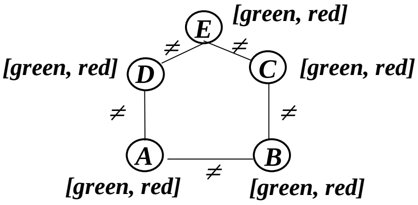
- Preliminaries:
  - Combinatorial Optimization Problems
  - CP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

BUCKET ELIMINATION

- *[Detcher AIJ 99]*: More general scope than optimization
- Bucket elimination: algorithmic framework that generalizes dynamic programming
  - The algorithm inputs are variables and functions on these variables
  - Functions are partitioned into buckets each associated with a single variable
  - Given a variable ordering, the bucket on variable X contains all functions on X but those involving variables higher that X
  - Buckets are processed from last to first
    - When bucket on X is processed, an elimination procedure produces a new function that “does not mention X”. The new function is placed in the lower bucket.
  - Complexity limited by the induced width

BUCKET ELIMINATION

- Adaptive consistency:



Bucket(E): E ≠ D, E ≠ C
Bucket(C): C ≠ B
Bucket(D): D ≠ A
Bucket(B): B ≠ A
Bucket(A):

Bucket(E): E ≠ D, E ≠ C
Bucket(C): C ≠ B || C = D
Bucket(D): D ≠ A || D ≠ B
Bucket(B): B ≠ A || B = A
Bucket(A): ||

COST NETWORKS

- Minimize the cost:

`C(a,b,c,d,e) = C(a)+C(b,a)+C(c,a)+C(e,b,c)+ C(d,a,b)`

$$\min_{B} \Sigma \; \overbrace{C(a,b,e),C(b,c),C(b,e)}$$

Bucket(B): `C(a,b,e),C(b,c),C(b,e)`

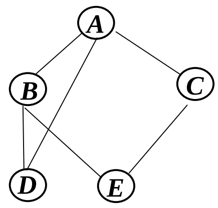
Bucket(C): `C(a,c),C(c,e),` $h^B(a,d,c,e)$

Bucket(D): $h^C(a,d,e)$

Bucket(E): `E=0,` $h^d(a,e)$

Bucket(A): $h^e(a)$

OPT



STATE SPACE RELAXATION IN CP

- Example: SSR of the path constraint *[Focacci Milano CP-AI-OR01]*
  - for each node: n+3 variables
    - $\mathbf{prev_i} \in \mathbf{[0..n]}$ models the node directly preceding **i**
    - $\mathbf{pos_i} \in \mathbf{[0..n]}$ models the position of node **i** in the path
    - $\mathbf{cumul_i} \in \mathbf{[0..M]}$ models the cost from the start to node **i**
    - $\mathbf{cumul2_{ip}} \in \mathbf{[0..2M]}$ modeling the cost from the start to node **i** if node **i** is in the position **p** ⇒ meaningless if node **i** is not in the position **p** in which case it is set to values greater than **M**
  - Constraints
    - $\mathbf{cumul2_{i,pos_i} = cumul_i}$ ⇒ element constraint
    - $\mathbf{cumul2_{ip} = cumul2_{prev_i,p-1} + c_{prev_i,i}}$
    - $\mathbf{pos_i \neq p \Rightarrow cumul2_{ip} > M}$

STATE SPACE RELAXATION IN CP

- Drawback of the previous formulation:
  - $cumul2_{ip} \in [0..2M]$ modeling the cost from the start to node i if node i is in the position p ⟹ *meaningless* if node i is not in the position p in which case it is set to values greater than M

- Limited propagation of the element constraint
  - $inf(cumul_i) \geq \min_{p \in pos_i} \{inf(cumul2_{ip})\}$
  - $sup(cumul_i) \leq \max_{p \in pos_i} \{sup(cumul2_{ip})\}$ *
  - $sup(cumul2_{ip}) < inf(cumul_i) \Rightarrow pos_i \neq p$ *
  - $inf(cumul2_{ip}) > sup(cumul_i) \Rightarrow pos_i \neq p$
  - $pos_i = p \Rightarrow cumul2_{ip} = cumul_i$

CONDITIONAL VARIABLES

- A CP variable

$$X::Dom \Leftrightarrow Dom \neq \emptyset$$

- A conditional variable extends a CP variable with a constraint

$$X^{cst}::Dom \Leftrightarrow (X::Dom \Leftrightarrow cst)$$

  - A conditional is defined true if its definition constraint holds, false otherwise
  - The associated constraint can be referred to as **Cst(X)**
- Constraints on conditional variables *[Focacci Milano CP-AI-OR01]* simply extend classical constraints

$$X^{cst} = Y \Leftrightarrow (X = Y \Leftrightarrow cst)$$

$$X^{cst1} = Y^{cst2} \Leftrightarrow (X = Y \Leftrightarrow (cst1 \wedge cst2))$$

CONDITIONAL VARIABLES

- Arithmetical operator can also be defined on conditional variables

$$Z^{cst3}=X^{cst1}+Y^{cst2} \Leftrightarrow (Z=X+Y \Leftrightarrow (cst3=cst1 \wedge cst2))$$

- Conditional variables can be used to define constructive disjunction:

xunion(Cvars,Y,X)

  - where **X** and **Y** are regular variables and **Cvars** is an array of k conditional variables

xunion(Cvars,Y,X) holds iff one out of k conditional variables is true (**Cvars[i*]**), **X = i*** and **Cvars[X]=Y**

CONSTRUCTIVE DISJUNCTION

- Propagation of **xunion**

$$inf(Y) \geq \min_{i \in Dom(X)}\{inf(Cvars[i])\}$$

$$sup(Y) \leq \max_{i \in Dom(X)}\{sup(Cvars[i])\}$$

$$inf(Cvars[i]) \geq inf(Y)$$

$$sup(Cvars[i]) \leq sup(Y)$$

$$inf(Y) > sup(Cvars[i]) \Rightarrow X \neq i$$

$$sup(Y) < inf(Cvars[i]) \Rightarrow X \neq i$$

$$i \notin Dom(X) \Leftrightarrow \neg\, cst(Cvars[i])$$

$$X = i \Leftrightarrow cst(Cvars[i])$$

STATE SPACE RELAXATION IN CP

- The constraint previously used $\mathbf{cumul2}_{\mathbf{i,posi}} = \mathbf{cumul}_{\mathbf{i}}$ can be replaced by

$$\textit{xunion}(\textit{cumul2}_{ip},\textit{cumul}_i,\textit{pos}_i)$$

  - Bounds on $\mathbf{cumul2}_{\mathbf{ip}}$ can now be safely modified while in the previous model the domain should be left open to consider the case where $\mathbf{pos}_{\mathbf{i}} \neq \mathbf{p}$

EXTENSION TO CONSTRUCTIVE DISJUNCTION

- We can extend constructive disjunction when also both **X** and **Y** are conditional variables:

$$\textit{xunion}(\textit{Cvars},Y^{cst1},X^{cst2})$$

holds iff $\textit{xunion}(\textit{Cvars},Y,X) \Leftrightarrow cst1 \wedge cst2$

  - Interesting case: **Y** and **X** are conditioned by the same constraint. Propagation in this case enforces also

$$cst(Y) \Rightarrow (X = i \Leftrightarrow cst(Cvars[i]))$$

$$cst(Y) \Rightarrow (i \notin Dom(X) \Leftrightarrow \neg\, cst(Cvars[i]))$$

$$cst(Y) \Rightarrow inf(Y) \geq \min_{i \in Dom(X)}\{inf(Cvars[i])\}$$

$$cst(Y) \Rightarrow sup(Y) \leq \max_{i \in Dom(X)}\{sup(Cvars[i])\}$$

STATE SPACE RELAXATION IN CP

- With the extended **xunion** constraint we can write

$$xunion((cumul2_{j,p-1} + c_j)_j, cumul2_{i,p}, prev_i)$$

  - $\mathbf{cumul2_{ip}}$ and $\mathbf{prev_{ip}}$ share the same conditioning constraint $\mathbf{pos_i = p}$
  - in CP we have full exploitation of DP recursion since bounds on conditional variables can be updated by other constraint

SSR MODEL OF TSP

- The path constraint can be modeled by using SSR
  - for each node: 2n+3 variables
    - $prev_i \in [0..n]$ models the node directly preceding i
    - $pos_i \in [0..n]$ models the position of node i in the path
    - $cumul_i \in [0..M]$ models the cost from the start to node i
    - $cumul2_{ip}^{pos_i=p} \in [0..M]$ conditional variables for costs
    - $prev2_{ip}^{pos_i=p} \in [0..N]$ conditional variables for prev

$xunion(cumul2_{i,p}, cumul_i, pos_i)$
$xunion(prev2_{i,p}, prev_i, pos_i)$
$xunion((cumul2_{j,p-1} + c_j)_j, cumul2_{i,p}, prev2_{i,p})$

OVERVIEW

- Preliminaries:
  - Combinatorial Optimization Problems
  - CP and OR techniques
- Comparisons
- Integration:
  - *problem modelling*
  - *problem solving*
    - *Feasibility: Global Constraint Filtering Algorithms*
    - *Optimality*
      - *branch & bound*
      - *branch & cut*
      - *column generation*
      - *dynamic programming*
    - *Search*

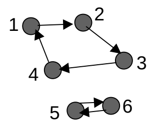
GUIDING SEARCH

- Exploiting results from the optimal solution of the relaxation in order to guide the search
  - the search is focussed on more “promising” values

- Example of search strategies based on relaxation:
  - violated constraints
    - subtour elimination strategy
    - branching on variables with fractional values
  - notion of regret
  - probe backtracking

SUBTOUR ELIMINATION STRATEGY

- Example on TSP: *[Focacci Lodi MilanoCP98 Wrk. on LSCO]*
  - – relaxation: Assignment Problem
  - – optimal solution of AP is integer, but possibly presents subtour
  - – violation of subtour constraints in TSP



- Start from the optimal AP solution
- Select a subtour and break it
  - – Example: subtour 1 2 3 4
    - $Next_1 \neq 2 \vee (Next_1=2, Next_2 \neq 3) \vee (Next_1=2, Next_2=3, Next_3 \neq 4) \vee (Next_1=2, Next_2=3, Next_3=4, Next_4 \neq 1)$
    - $1>2 \vee (1<2, 2>3) \vee (1<2, 2<3, 3>1) \vee (1<2, 2<3, 3<4, 4>1)$ more suitable for CP

BRANCHING ON FRACTIONAL VARs

- LP provides a solution with fractional values
- Select variables that are assigned to fractional values
  - – Example: optimal LP solution Xi = 4.2
  - – Branching
    - $Xi \leq 4 \vee Xi \geq 5$
- Used in OR Branch and Bound

NOTION OF REGRET

- Regret: difference between the best value and the second best
- Select the variable with maximal regret
  - should be assigned first to its best values otherwise the solution would be worsened too much
- Computation of regret on the cost matrix *[Caseau Laburthe CP97]*
- Computation of regret on reduced costs *[Focacci Lodi MilanoCP99]*
  - select the variable with higher minimal reduced cost
  - assign the value suggested by the relaxation

NOTION OF REGRET

- Computation of regret on the cost matrix *[CaseauLaburthe CP97]*

$$\mathbf{Regret_i = c_{i,best} - c_{i,second}}$$

- Computation of regret on reduced costs *[Focacci Lodi MilanoCP99]*
  - select the variable with higher minimal reduced cost
  - assign the value suggested by the relaxation

$$\mathbf{Regret_i = \min_{\substack{k=1..n \\ k \neq opt(i)}} \{\bar{c}_{ik}\}}$$

PROBE BACKTRACKING

- Backtrack search supported by lookahead procedures (*probe generators*) which dynamically generate potentially good assignments (*probes*). *[El Sakkout, Wallace Constraints 2000] [Purdom Haven SIAM J. on Computing97]*
  - the probe generator assigns each variable a tentative value
  - focus the backtrack search on regions where the probe violates constraints
  - probe generator should provide good solutions (super-optimal)

- Unimodular Probing Algorithm: the LP finds optimal integer solutions on unimodular subproblems which are considered as probes

INCOMPLETE APPROACHES

- Integration of CP and Local Search
  - CP global search is used to find an initial (feasible) solution and LS is applied to improve this solution;
  - within a CP framework, the exhaustive exploration is stopped at a chosen level of the search tree and the leafs are reached through LS;
  - within a LS (metaheuristic) framework, CP global search is used to exhaustively explore the neighborhood, or to complete in optimal way a partial solution.

- References: *[Li et al.“Modern Heuristic Search Methods”,John Wiley96], [Shaw CP98],[Pothos, Richards, Cp98 Wks.on really hard problems], [Michel, Van Hentenryck CP97],[Psarras et al. European JOR97],[Pesant, Gendrau CP96],[Nuijten LePape JOH98],[Caseau et al. CP99] [DeBacker, Furnon,Shaw CPAIOR99], [Caseau, Laburthe CPAIOR99]*

SYSTEMS

- Prolog III ***[Colmerauer CACM(33) 90]***,
- CHIP ***[Dincbas et al., JICSLP88],***
- 2LP ***[McAloon, Tretkoff PPCP93]***
- CLP(PB) and *COUPE* ***[Barth,Bockmayr ICLP95] [Kasper PhD98]***,
- OOPDB ***[Barth PhD 96]***
- Ecl^{i}ps^{e} ***[Wallace et al.97]***
- ILOG ***[Puget SPICIS94], [Puget, Leconte ILPS95]***
  - ***Solver Planner Dispatcher Scheduler***
- OPL ***[Van Hentenryck MIT Press99]***
- SCHEDEns ***[Colombani PhD97]***
- COME ***[Heipcke PACT96]***
- CLAIRE ***[Caseau Laburthe JICSLP Wks on Multi Paradigm Logic98],***
- SALSA ***[Caseau Laburthe CP98],***
- LOCALIZER ***[Michel, Van Hentenryck CP97]***
- *many others……….*

171

TO KNOW MORE...

- Conferences and Journals
  - Conferences on Constraints: CP, PACLP, Constraints...
  - Conferences on AI: ECAI, IJCAI, AAAI, AIJ...
  - Conferences on OR: IFORS, INFORMS, Annals of OR
- INFORMS Journal on Computing (Special issue vol.10, No.3 1998)
- Annals of Mathematics and AI (Special issue LSCO, to appear)
- Journal of Heuristics (Special issue on CP-AI-OR99, to appear)
- CHIC2 Deliveries
- Workshops on the subject:
  - *CP98 and CP99 Workshop on Large Scale Combinatorial Optimisation and Constraints*
  - *CP-AI-OR99, CP-AI-OR2000 CP-AI-OR2001 Workshop on integration of AI and OR techniques in CP for Combinatorial Optimization*
  - *Forthcoming CP-AI-OR'02*
  - *AAAI2000 Workshop on integration of AI and OR techniques for Combinatorial Optimization*

172

CONCLUSION & FUTURE DIRECTIONS

- CP advantages:
  - easy modeling (less bugs)
  - constraint propagation
  - guide search
  - flexibility
- CP drawbacks:
  - no global reasoning
  - no optimality reasoning

- OR advantages:
  - global problem view (LP)
  - exploitation of structure
  - optimization problems
- OR drawbacks:
  - models are less flexible
  - software engineering poor

Integration benefits from the advantages of both

CONCLUSION & FUTURE DIRECTIONS

- CP Problem modelling
  - composition of basic components (constraints)
- CP Problem solving
  - interaction of constraint propagation algorithms

- OR provides new components/algorithms
  - relaxation (LP, specific algorithms)
  - cut generation
  - dynamic programming
  - filtering algorithms (e.g. Edge finder)

- Collaboration through the constraint store

CONCLUSION & FUTURE DIRECTIONS

- Two extremes

Pure CP — *No bound, nodes explored quickly*

What between?

Composition of techniques by putting together basic blocks

Branch & Cut — *Close the problem in few nodes*

- Move away from the black box approach
  - let the user compose the best technique:
    - composition of different solvers
    - composition of constraint propagation algorithms
    - composition of relaxations

CONCLUSION & FUTURE DIRECTIONS

- Integration can exploit
  - problem decomposition
  - problem abstraction
  - problem special (sub)structure
  - problem *perspective* (cooperative solvers)
  - different solvers capabilities
  - exchange results
- Guidelines
  - each solver manages the sub-problem it is more suitable for
  - cheap solvers first, lower solver later
  - define events triggering solver computations

CONCLUSION & FUTURE DIRECTIONS

- Integration: the other way round
  - Can OR benefit from the CP paradigm ?
- Software engineering: modelling tools
- Constraint propagation during search
- Global constraints in IP *[Bockmayr Kasper INFORMS J. Comp.98]*
  - TSP structure
  - Assignment structure