A System For Integrating Optimization Techniques

Ionut Aron
Brown University

John Hooker, Tallys Yunes
Carnegie Mellon University

APMOD, Brunel University, June 2004

SiMPL

A System For Integrating Optimization Techniques

Ionut Aron
Brown University

John Hooker, Tallys Yunes
Carnegie Mellon University

SiMPL

Outline

- **Why Integrated Methods?**
- **SIMPL Philosophy**
- **Architecture**
- **Modeling Examples**

SiMPL

Why Integrated Methods?

- Basic motivation
- Product configuration problem
- Planning and scheduling problem
- Stochastic integer programming

Basic motivation

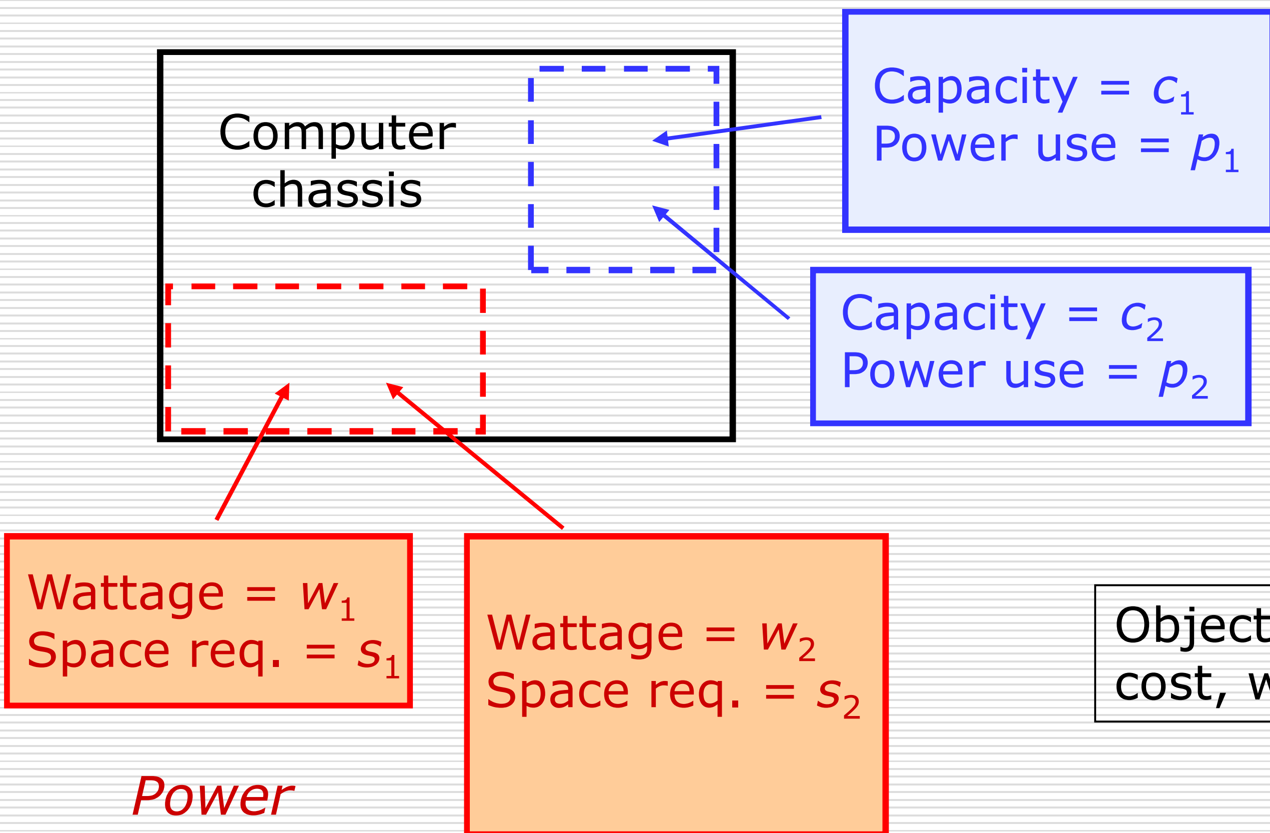
- Integrated methods can result in simpler models and faster execution.
    - Math programming + constraint programming
- Full benefits of integrated methods currently require low-level coding.
    - This discourages research and applications.
- Goal: a high-level modeling/solution system that permits micro-level integration: **SIMPL**.
    - Eventual goal: extend to local search.

Product configuration problem

Example of an integrated approach…

- Find optimal configuration of a computer or other product.
- Choose power supply, memory chips, etc., to satisfy requirements and constraints.

Product configuration problem



Memory chip options

Requirements: bounds on capacity, power usage

Power supply options

Requirements: bounds on wattage, space

Objective: minimize cost, weight, etc.

Problem formulation

$$\min_{t,q,r} \sum_k c_k r_k$$

$$\text{subj.to} \quad r_k = \sum_i q_i A_{kit_i}, \quad \text{all } k$$

$$r_k \geq R_k, \quad \text{all } k$$

$$t_i \in \{\text{component types}\}$$

Quantity of component *i* used *(e.g., number of disk drives)*

Amount of attribute *k* supplied (consumed) by type t_i of component *i*

Amount of attribute *k* supplied/consumed *(e.g., amount of wattage supplied)*

Type of component *i* chosen *(e.g., type of power supply)*

MILP model

$$
\begin{aligned}
\min_{t,q,r} \quad & \sum_k c_k r_k \\
\text{subj.to} \quad & r_k = \sum_{ij} q_{ij} A_{kij}, \quad \text{all } k \\
& \sum_j t_{ij} = 1, \quad \text{all } i \\
& q_{ij} \le M t_{ij}, \quad \text{all } i, j \\
& r_k \le R_k, \quad \text{all } k \\
& t_{ij} \in \{0,1\}
\end{aligned}
$$

disaggregated variables (q_{ij}, t_{ij})

Integrated model

Same as original formulation

Constraint programming deals routinely with variable indices using the *element* global constraint

$$
\begin{aligned}
\min_{t,q,r} \quad & \sum_k c_k r_k \\
\text{subj.to} \quad & r_k = \sum_i q_i A_{kit_i}, \quad \text{all } k \\
& r_k \geq R_k, \quad \text{all } k \\
& t_i \in \{\text{component types}\}
\end{aligned}
$$

Element global constraint

x_y is replaced by z, plus the constraint

$$\text{element}(y, (x_1, \ldots, x_n), z)$$

Sets z equal to yth variable in the list $x_1, \ldots, x_n$

To implement $q_i A_{kit_i}$, replace it with x_{kit_i} (which is implemented with *element*) and write constraints $x_{kij} = q_i A_{kij}$ for all j

Solving the problem

- From MILP and CP: Solve by *branching search*.
- From MILP: Solve *linear relaxation* at each node of search tree.
  - Provides bound for branch and bound.
  - Will generate relaxation of *element*.
- From CP: At each node, perform *domain filtering* for each constraint.
  - Remove values that a variable cannot take in any solution that satisfies the constraint.
  - Will use specialized filtering algorithm for *element.*
- From MILP: Reduced cost variable fixing.

Relaxation of *element*

If $0 \le x_j \le b$ for each *j*, a *convex hull relaxation* of

$$\text{element}(y, (x_1, \ldots, x_n), z)$$

is given by

$$\sum_{j \in D_y} x_j - (n-1)b \le z \le \sum_{j \in D_y} x_j$$

Current domain of *y*

Domain filtering for *element*

Example...

$$\text{element}(y, (x_1, x_2, x_3, x_4), z)$$

The initial domains are:

$D_z = \{20,30,60,80,90\}$

$D_y = \{1,3,4\}$

$D_{x_1} = \{10,50\}$

$D_{x_2} = \{10,20\}$

$D_{x_3} = \{40,50,80,90\}$

$D_{x_4} = \{40,50,70\}$

The reduced domains are:

$D_z = \{80,90\}$

$D_y = \{3\}$

$D_{x_1} = \{10,50\}$

$D_{x_2} = \{10,20\}$

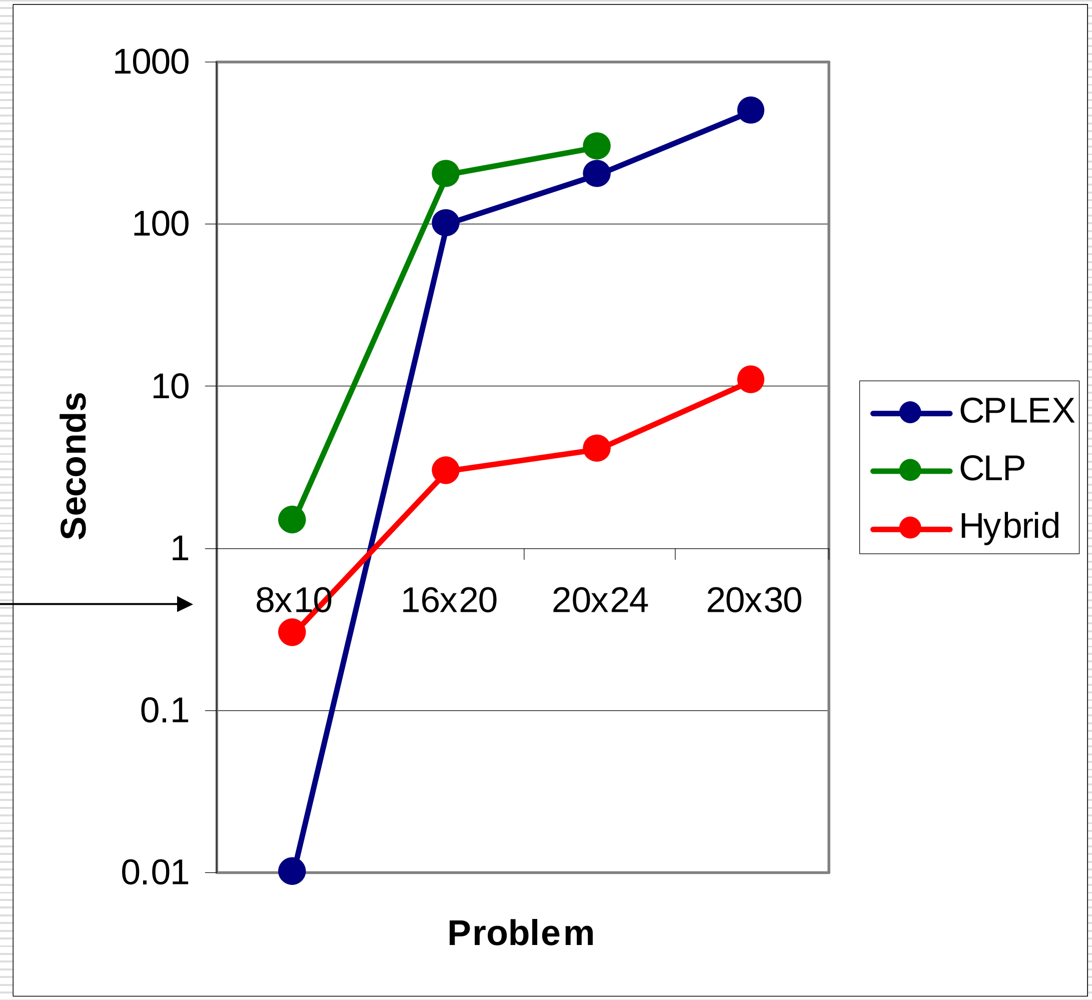
$D_{x_3} = \{80,90\}$

$D_{x_4} = \{40,50,70\}$

Computational results

From: Ottosson & Thorsteinsson, 2001

$$\begin{pmatrix} \text{number} \\ \text{of} \\ \text{components} \end{pmatrix} \times \begin{pmatrix} \text{number} \\ \text{of} \\ \text{attributes} \end{pmatrix}$$



Planning & scheduling problem

- Allocate tasks to facilities.
- Schedule tasks on each facility.
  - Subject to release times & deadlines.
  - Facilities may run at different speeds and incur different costs.

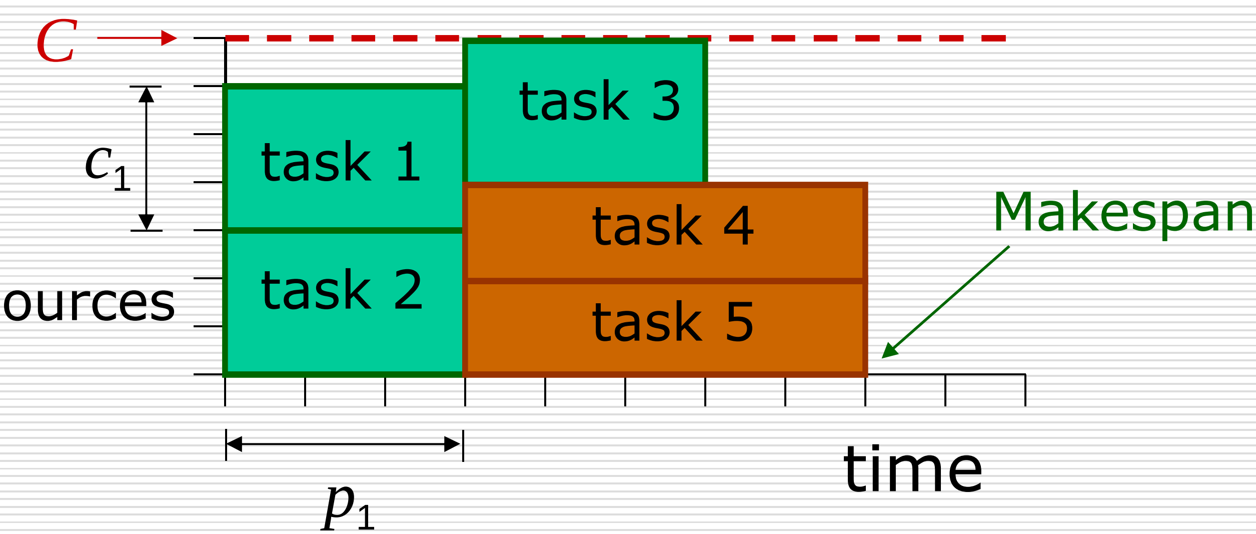
- MILP is good at allocation.
- CP is good at scheduling.
- We will combine them.

In practice, there is often an informal give-and-take between master planners and schedulers.

This process can be automated by *logic-based Benders composition.*

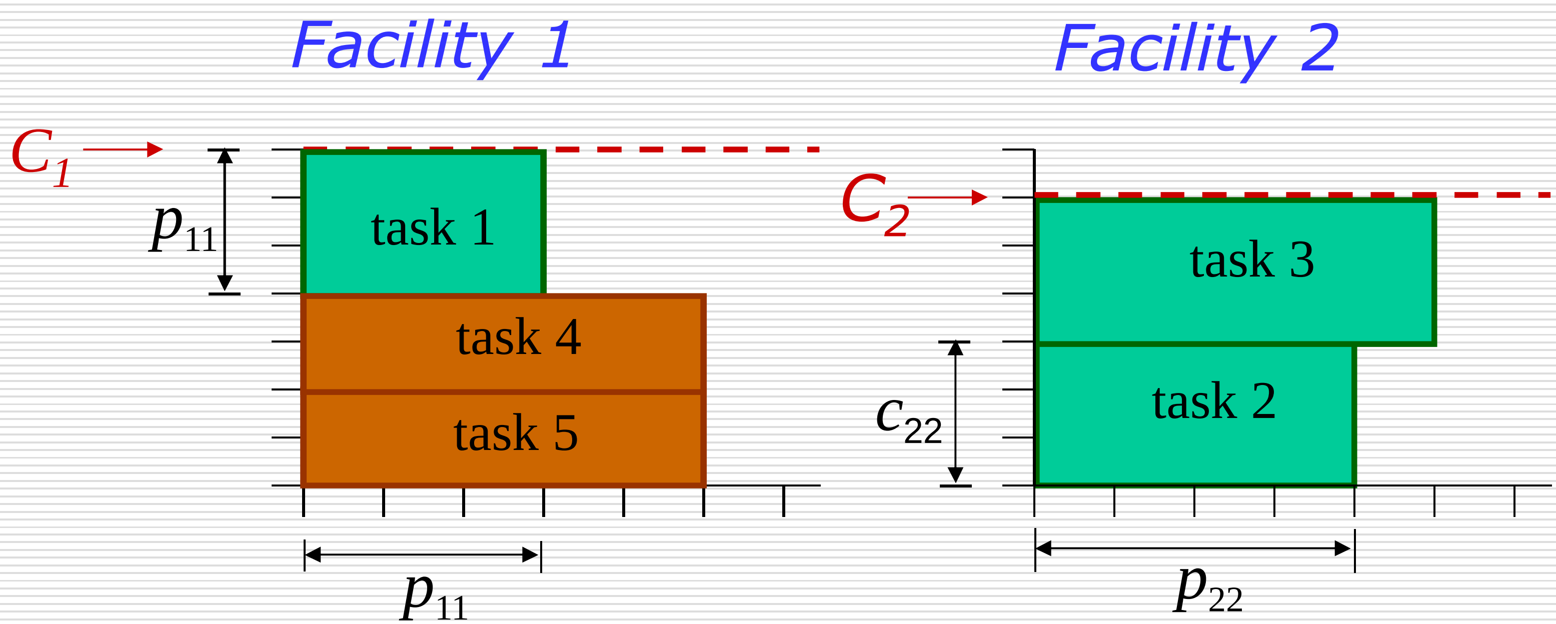
Cumulative scheduling

Several tasks may run simultaneously on a given facility.



Total resource consumption $\leq C$ at all times.

Allocation + cumulative scheduling



Total resource consumption ≤ C_i at all times.

Problem formulation

Fixed cost of assigning task *j* to facility x_j

$$\min \quad \sum_j F_{x_j j}$$

$$\text{subj.to} \quad \sum_{\substack{j \\ t_j \le t \le t_j + p_{x_j j}}} c_{x_j j} \le C_i, \quad \text{all } i$$

Resource requirement of task *j* on facility x_j

$$r_j \le t_j \le d_j - p_{x_j j}, \quad \text{all } j$$

$$x_j \in \{\text{facilities}\}$$

Time windows

Discrete-time MILP model

$$\min \sum_{ijt} F_{ij} x_{ijt}$$

= 1 if task j starts at discrete time t on facility i ($t = 1,\ldots,N$)

subject to

$$\sum_{it} x_{ijt} = 1, \quad \text{all } j$$

Task j starts at one time on one facility

$$\sum_{j} \sum_{\substack{t' \\ t - p_{ij} < t' \le t}} c_{ij} x_{ijt'} \le C_i, \quad \text{all } i,t$$

Resources consumed at time t on facility i

$$x_{ijt} = 0, \quad \text{all } j,t \text{ with } d_j - p_{ij} < t$$

$$x_{ijt} = 0, \quad \text{all } j,t \text{ with } t > N - p_{ij} + 1$$

Time windows

$$x_{ijt} \in \{0,1\}$$

Discrete-event MILP model (continuous time)

Idea: Türkay & Grossmann

$$\min \sum_{ijk} F_{ij} x_{ijk}$$

subject to

$$\sum_{ik} x_{ijk} = 1, \quad \sum_{ik} y_{ijk} = 1, \quad \text{all } j$$

$$\sum_{ij} x_{ijk} + y_{ijk} = 1, \quad \text{all } k$$

$$\sum_{k} x_{ijk} = \sum_{k} y_{ijk}, \quad \text{all } i, j$$

$$t_{i,k-1} \le t_{ik}$$

continued…

x_{ijk} = 1 if event *k* is start of task *j* on facility *i* (*k* = 1,…,2*N*)

y_{ijk} = 1 if event *k* is end of task

t_{ik}: Start time of event *k* (disaggregated by facility)

Finish time of task *j* (disaggregated by facility)

$$0 \le t_{ik}, \quad f_{ij} \le d_j, \quad \text{all } i, j, k$$

$$t_{ik} + p_{ij} x_{ik} - M(1 - x_{ijk}) \le f_{ij} \le t_{ik} + p_{ik} x_{ijk} + M(1 - x_{ijk}), \quad \text{all } i, j, k$$

$$t_{ik} - M(1 - y_{ijk}) \le f_{ij} \le t_{ik} + M(1 - y_{ijk}), \quad \text{all } i, j, k$$

$$R_{ik} \le C_i, \quad \text{all } i, k$$

Resource limit

$$R_{i1} = R^s_{i1}, \quad R^s_{ik} = \sum_j c_{ij} x_{ijk}, \quad R^f_{ik} = \sum_j c_{ij} y_{ijk}, \quad \text{all } i, k$$

$$R^s_{ik} + R_{i,k-1} - R^f_{ik} = R_{ik}, \quad \text{all } i, k$$

$$x_{ijk}, y_{ijk} \in \{0,1\}$$

Calculation of resource consumption on facility *i* at time of each event

Cumulative scheduling in CP

$$\text{cumulative}\begin{pmatrix} (t_1,\ldots,t_n) \\ (p_1,\ldots,p_n) \\ (c_1,\ldots,c_n) \\ C \end{pmatrix}$$

is equivalent to

$$\sum_{\substack{j \\ t_j \le t < t_j + p_{ij}}} c_j \le C \;, \quad \text{all } t$$

Schedules tasks at times t_1, …, t_n so as to observe resource constraint.

Edge-finding algorithms, etc., reduce domains of t_j.

Integrated model

Must recognize that the resource limit is an instance of the *cumulative* constraint

$$\begin{aligned}
\min \quad & \sum_j F_{x_j j} \\
\text{subj.to} \quad & \text{cumulative}\begin{pmatrix} (t_j \mid x_j = i) \\ (p_{ij} \mid x_j = i) \\ (c_{ij} \mid x_j = i) \\ C_i \end{pmatrix}, \quad \text{all } i \\
& r_j \le t_j \le d_j - p_{x_j j}, \quad \text{all } j \\
& x_j \in \{\text{facilities}\}
\end{aligned}$$

Logic-based Benders: Basic idea

Decompose problem into

allocation	+	resource-constrained scheduling
Assign tasks to facilities		*Schedule tasks on each facility*
Master problem		Subproblem
Solve with MILP		Solve with CP

Generate logic-based *Benders cuts* from subproblem solutions, and add them to master problem.

Master problem: Allocate tasks

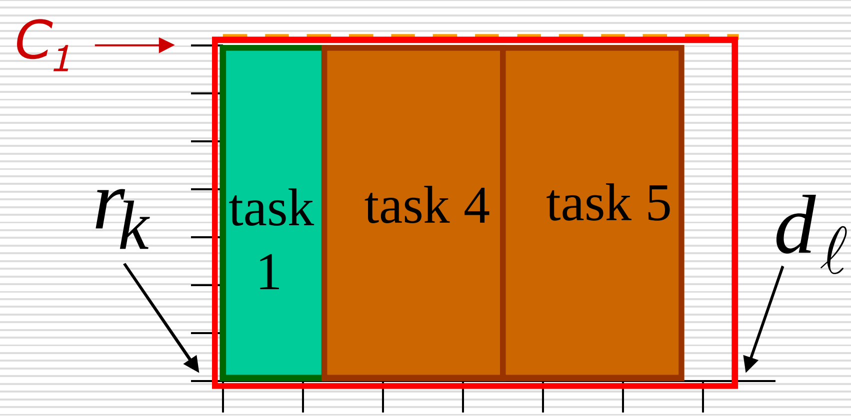
$$\min \sum_{ij} g_{ij} x_{ij}$$

subject to

$$\sum_{i} x_{ij} = 1, \quad \text{all } j$$

Benders cuts

$$\sum_{\substack{j \\ r_j \geq r_k \\ d_j \leq d_\ell}} p_{ij} c_{ij} x_{ij} \leq C_i (d_\ell - r_k), \quad \text{all } i, \text{all distinct } r_k, d_\ell$$



Relaxation of subproblem:
"Area" of tasks in time window $[r_k, d_\ell]$ must fit.

Subproblem: Schedule tasks

solution of master problem

$$\text{cumulative}\begin{pmatrix} (t_j \mid \bar{x}_{ij} = 1) \\ (p_{ij} \mid \bar{x}_{ij} = 1) \\ (c_{ij} \mid \bar{x}_{ij} = 1) \\ C_i \end{pmatrix}, \quad \text{all } i$$

$$r_j \le t_j \le d_j$$

Let J = set of tasks assigned to facility i.

If subproblem i is infeasible, solution of subproblem "dual" is a proof that not all tasks in J can be assigned to facility i.

Obvious Benders cut prevents this in future iterations.

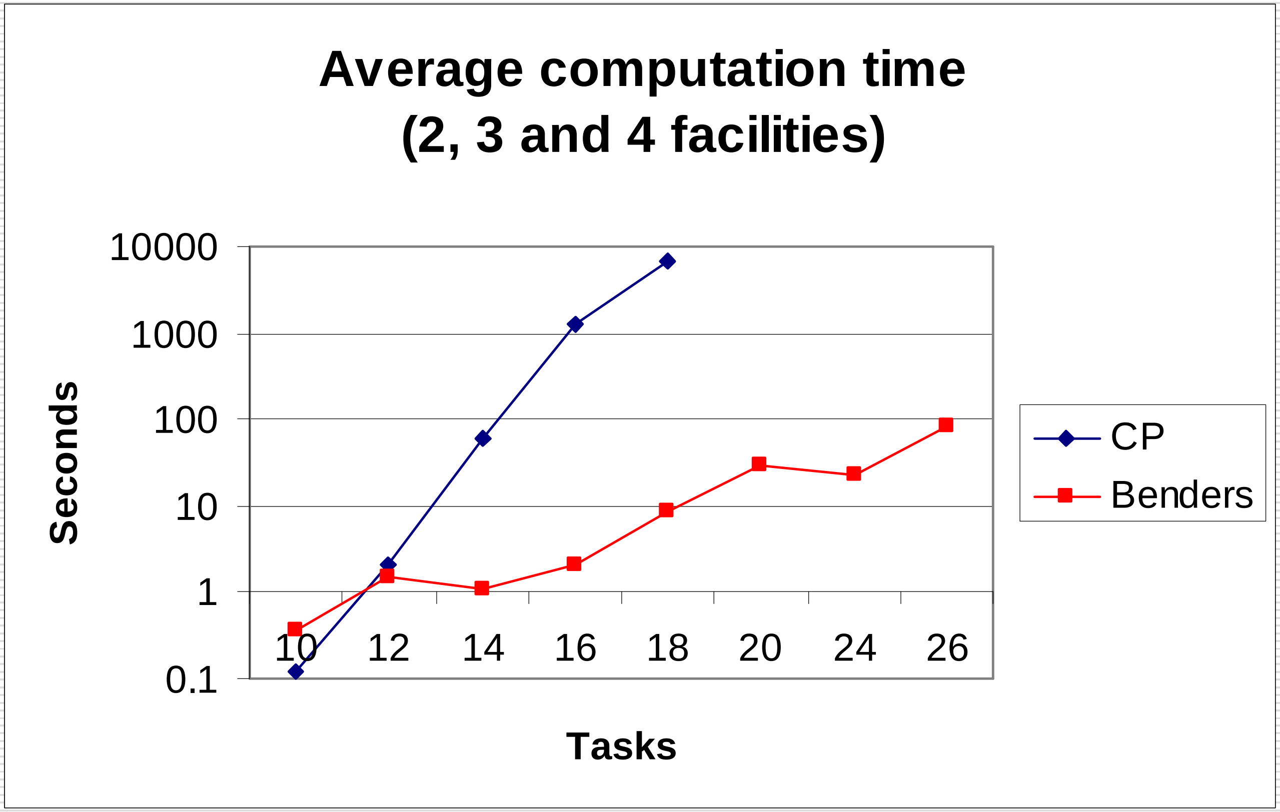
Master problem with Benders cuts

$$\begin{aligned}
\min \quad & \sum_{ij} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_i x_{ij} = 1, \quad \text{all } j \\
& \sum_{j \in J_{ih}} (1 - x_{ij}) \geq 1, \quad \text{all } i, h \\
& \sum_{\substack{j \\ r_j \leq r_k \\ d_j \leq d_\ell}} p_{ij} r_{ij} x_{ij} \leq C_i (d_\ell - r_k), \quad \text{all } i, \text{all distinct } r_k, d_\ell \\
& x_{ij} \in \{0,1\}
\end{aligned}$$

Logic-based Benders cuts

Computational results: Min cost

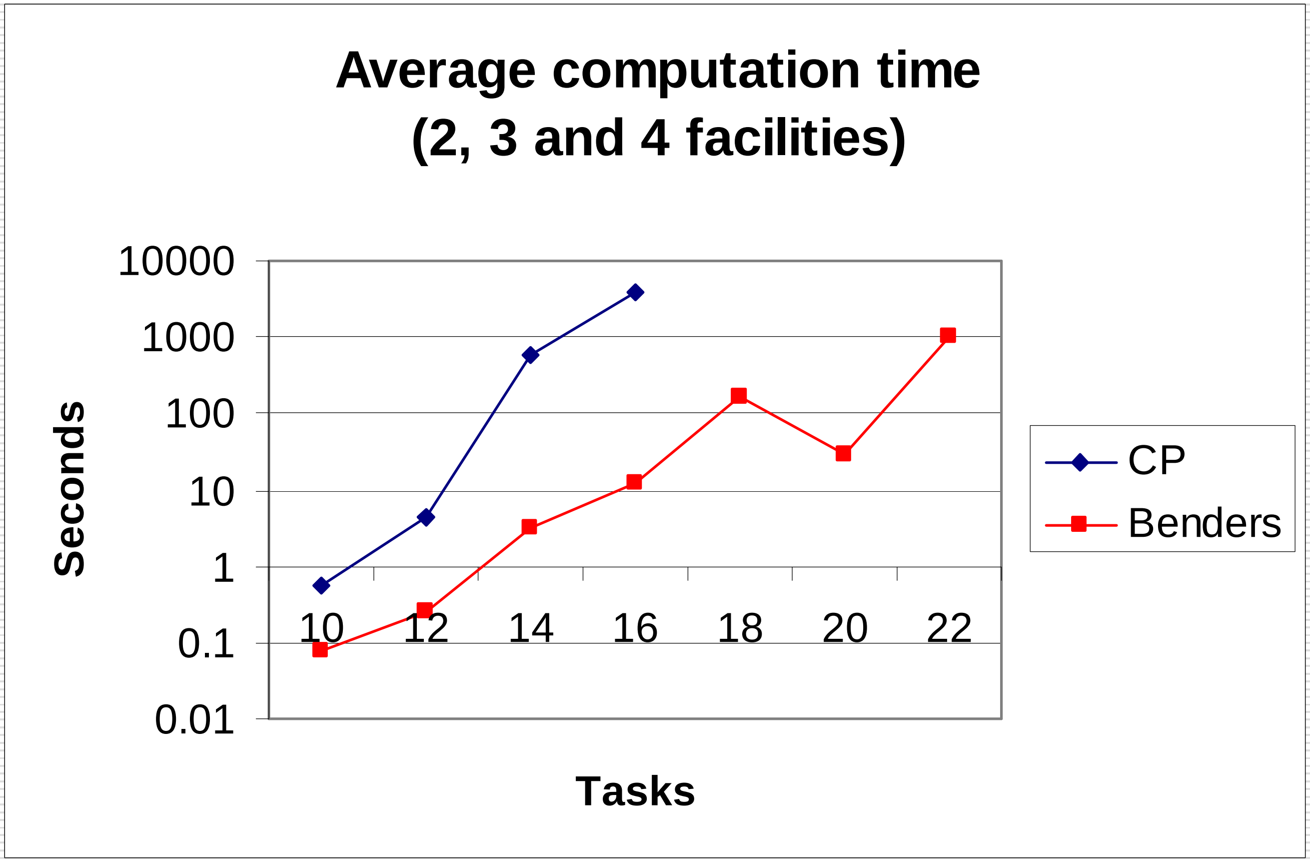
From:
JH, 2004



Each point is average over 15 problems. Computation terminated after 7200 seconds.

MILP (CLEX) was slower than CP (ILOG Scheduler) and started running out of memory at 16 tasks.

Computational results: Min makespan



Logic-based Benders cuts are less obvious for these problems.

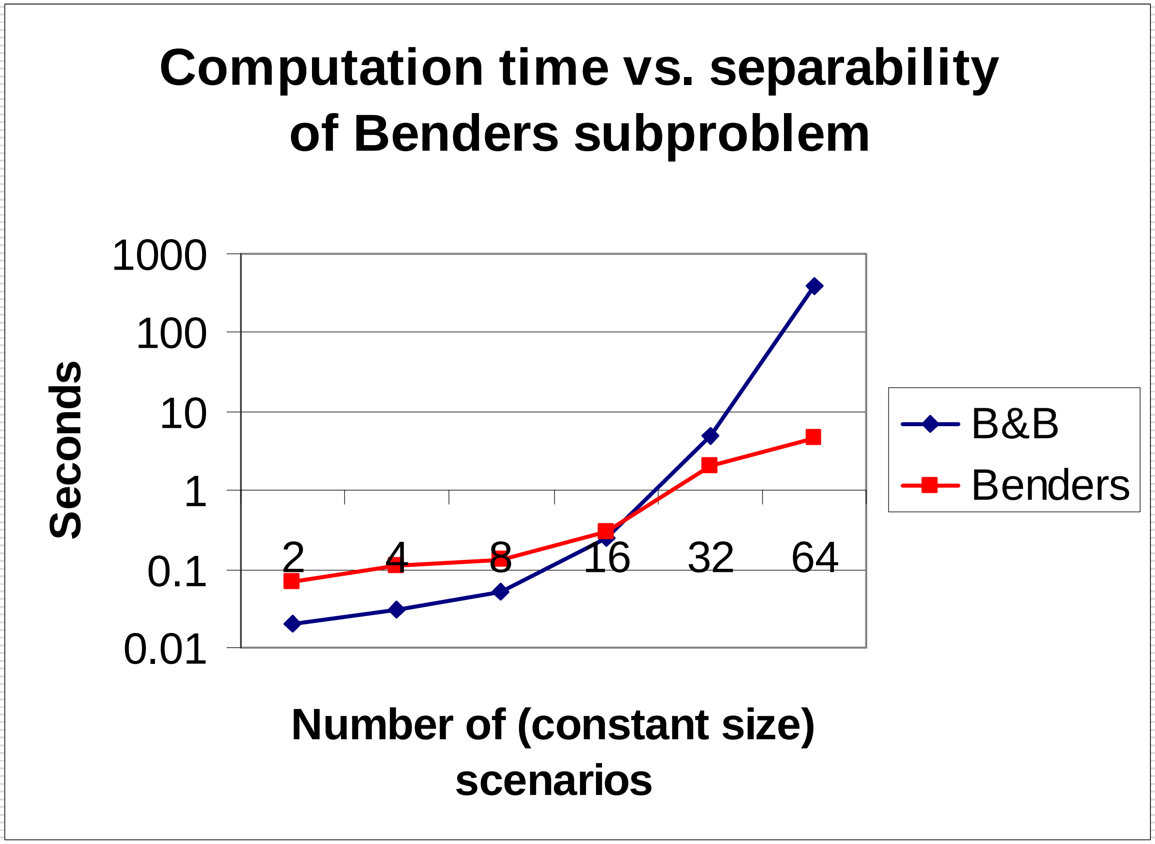
Stochastic integer programming

- Assumption: when some integer variables are fixed, remaining subproblem separates into smaller IPs.
  - Application: 2-stage stochastic IP.
  - Subproblem components correspond to scenarios.

- Use logic-based Benders.
  - Derive Benders cuts from B&B tree used to solve IP subproblems.
  - Benders cuts are disjunctions of linear inequalities.

Computational results: IP

From:
JH &
Ottosson,
2003



Benders is superior for >20 scenarios.

Advantage increases rapidly.

SiMPL

Philosophy

Objectives

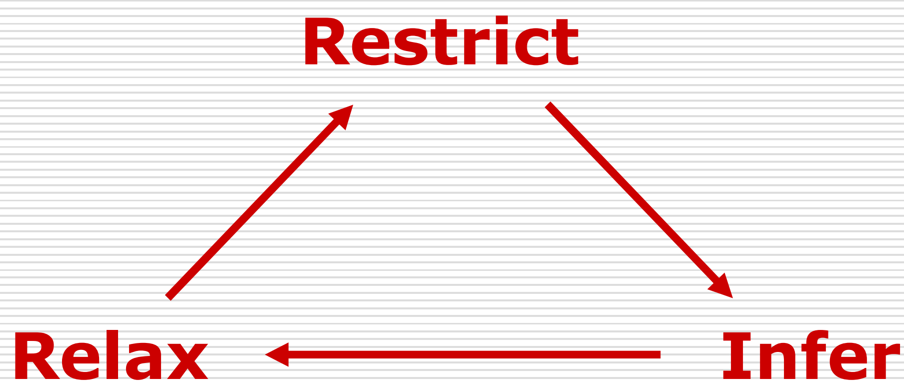
- **High-level modeling language**
  - The model reveals problem structure to the solver.
  - The constraint types dictate how integration occurs
- **Micro-level integration**
  - Integration more effective at the micro level.
  - A framework for both complete and local search methods.
- **Modularity, flexibility, extensibility**
  - Easy to add new constraint types, relaxation types, solvers and search strategies

Exploit common solution strategy

View CP, MILP and LS as special cases of a general method

…not separate methods **to be combined**

Common strategy:



Restrict – infer – relax

- **Restrict:** Enumerate restrictions of problem.
  - Formed by branching or adding constraints.
- **Infer:** Deduce additional constraints from current restriction.
  - Helps to rule out bad solutions.
- **Relax:** Solve relaxation of current restriction.
  - May be easier to solve.
  - May provide bounds that accelerate search.
  - Provides guidance for choosing next restriction.

Special case: MILP

- **Restrict:** Enumerate nodes of a branch-and-bound tree.
  - Restrictions created by branching on variables.
- **Infer:** Add cutting planes.
  - Makes linear implications explicit, and so tightens linear relaxation.
  - Pre-solve also an instance of inference (limited form of constraint propagation).
- **Relax:** Solve linear relaxation of current restriction.
  - Provides bounds for branch and bound.
  - Branch on fractional variables in solution of relaxation.

Special case: CP

- **Restrict:** Enumerate nodes of a search tree.
  - Restrictions may be created by branching on variable domains.
- **Infer:** Domain reduction (filtering) and constraint propagation.
  - Specialized filters for global constraints.
  - Results propagated from one constraint to another through the domain store, which contains variable domains.
- **Relax:** Domain store is a relaxation.
  - Branch on a domain in current domain store.

Special case: Classical Benders

- **Restrict:** Enumerate subproblems.
  - Restriction (subproblem) is created by fixing the master problem variables to their solution values.
- **Infer:** Generate Benders cuts.
  - Obtained from dual of subproblem.
- **Relax:** Solve master problem.
  - It is an incomplete description of projection of feasible set onto master problem variables.
  - Solution of master problem indicates which restriction (subproblem) to enumerate next.

Special case: Local search

- **Restrict:** Enumerate sequence of neighborhoods.
  - Neighborhood is feasible set of a restriction of the problem.
- **Infer:** Neighborhood reduction.
  - Eliminate infeasible solutions from current neighborhood.
- **Relax:** Relaxation generally identical with current restriction (but not necessarily).
  - Solve relaxation by (possibly incomplete) search of neighborhood.
  - Create next restriction by defining neighborhood of current solution.

How solver integrates methods

- At each stage of the **restrict – infer – relax** cycle, solver select techniques from **MILP**, **CP**, **Benders**, **local search**, etc.

For example:

- *Product configuration problem:* Use branching from **CP** and **MILP** to **restrict**, domain filtering from **CP** to **infer**, and linear relaxations from **MILP** to **relax**.

- *Planning and scheduling problem:* Use **Benders** to **relax** and **restrict**, **MILP** to solve resulting master problem, and **CP** to **infer** Benders cut.

- *Large neighborhood search:* Use **local search** to **restrict** and **CP** to search current neighborhood.

Constraint-oriented

- **Infer:** constraints drive the inference
  - Each constraint has **filtering/inference** algorithms
- **Relax:** constraints know how to relax
  - Each constraint has a **relaxation generator** that sends constraints to the appropriate relaxation (CS,LP,MIP etc)
- **Restrict:** constraints direct the search
  - Each constraint has a **branching module** that creates new restrictions based on solution of relaxation
  - Branch on a **violated constraint.**

Exploit structure at the constraint level

- **Infer:** Domain filters, cutting planes tailored to constraints.
  - Library of filters for **global constraints**
  - Library of cutting planes for specially-structured sets of **MILP constraints**
- **Relax:** Each constraint generates relaxations appropriate for it.
- **Restrict:** Constraints determine how nodes of the search tree are created.
  - To branch on variables, branch on in-domain constraints.
  - **Node selection** is determined by overall search procedure.

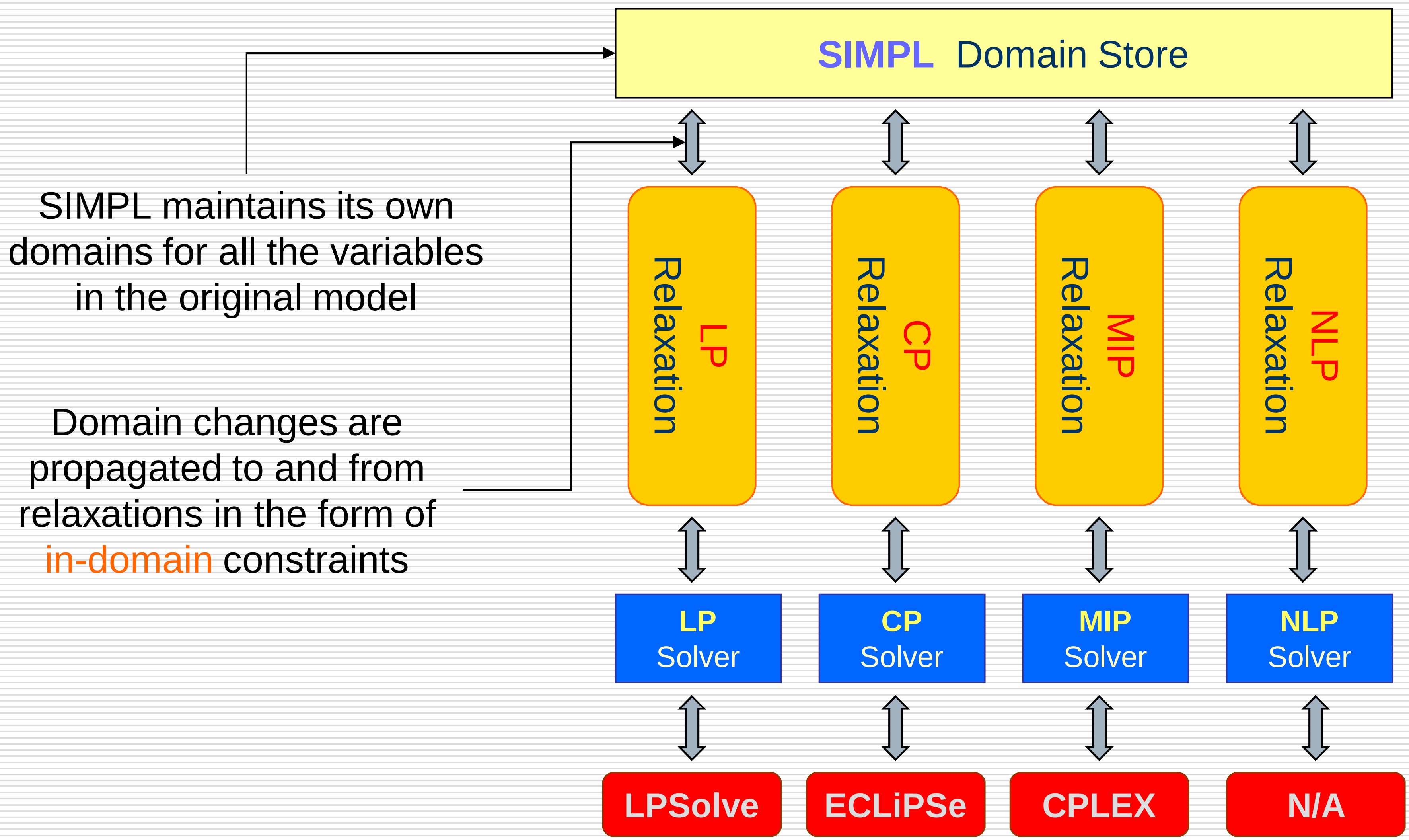
Exploit structure at the constraint level

- The modeler communicates problem structure to the solver by choice of high-level constraints.
  - Global constraints, structured subsets of inequalities, etc.
  - Traditional OR practice: convert everything to elementary constraints and hope that the solver rediscovers the structure.

SiMPL

Architecture

The architecture of SIMPL

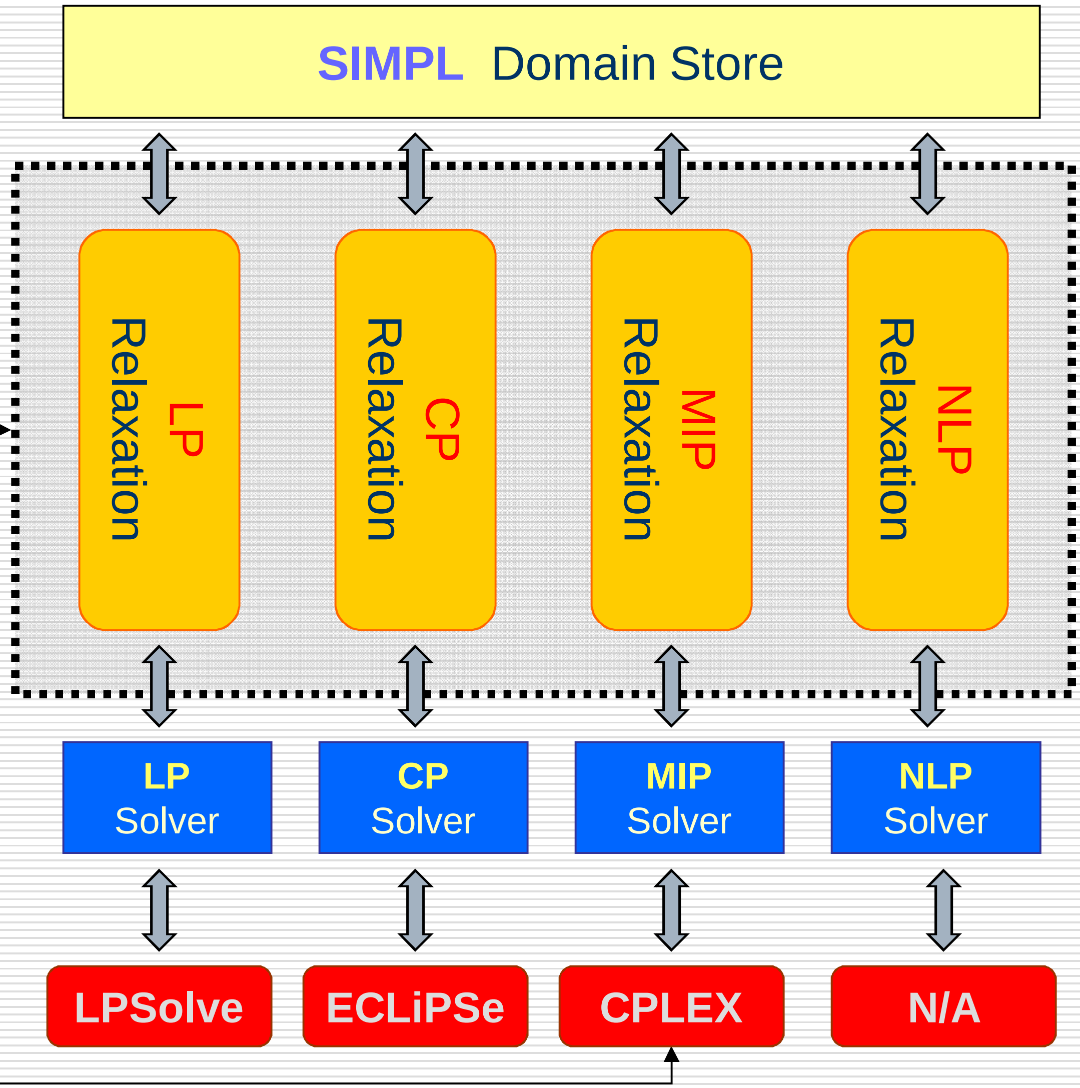


The architecture of SIMPL

The model determines the type of relaxations SIMPL will use

If the model contains a mix of linear constraints and global constraints, LP and CP relaxations are typically used

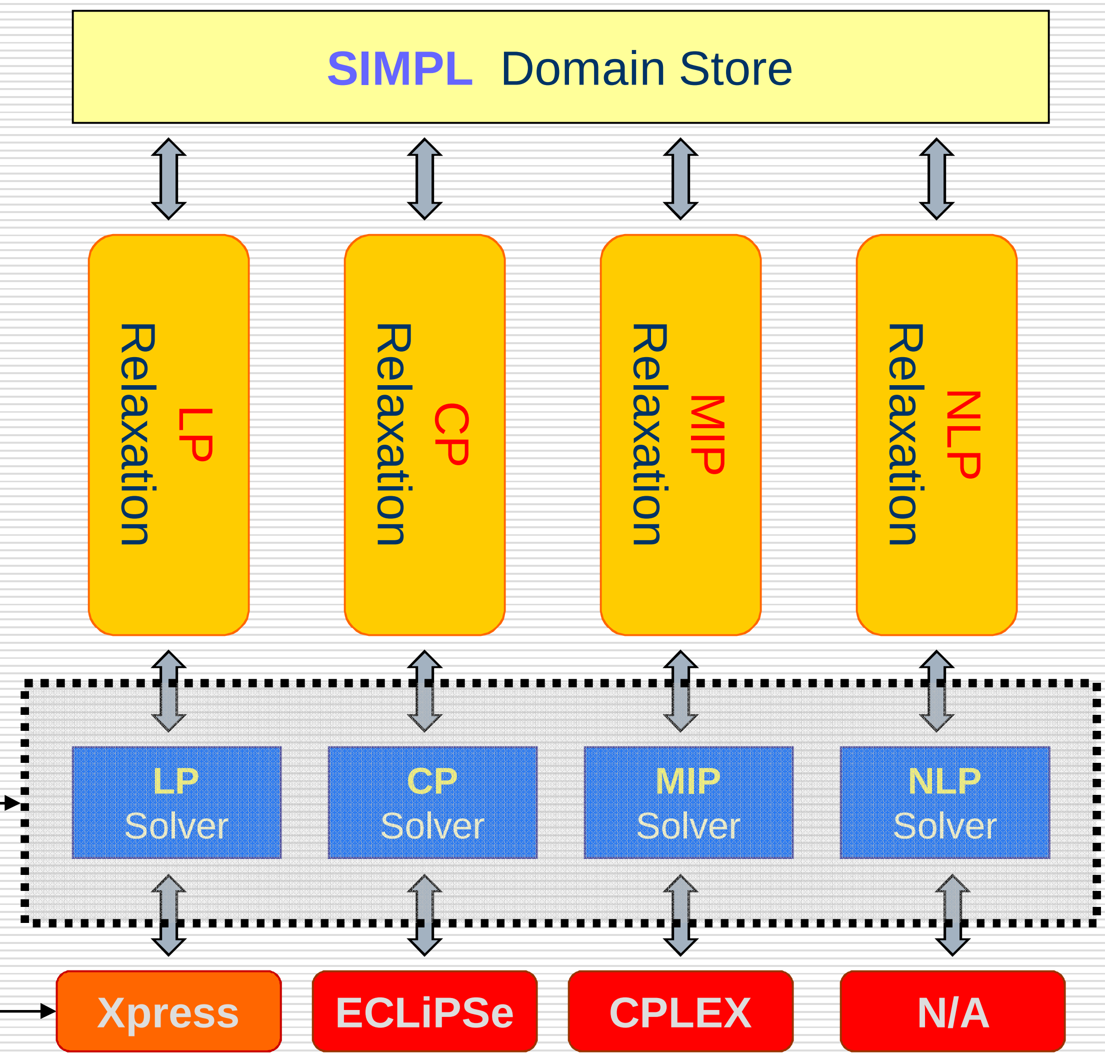
For Benders decomposition, a MIP relaxation might be needed as well (to handle the master problem)



The architecture of SIMPL

Each relaxation talks to its corresponding low-level solver through an **abstract solver interface**

Adding a new solver is easy – it amounts to implementing a standard API



SiMPL

Examples

Knapsack with side constraint (in detail)

Quadratic assignment

SiMPL

Example – Knapsack

Knapsack

An integer knapsack problem with side constraint

$$
\begin{aligned}
\text{min} \quad & 5x_1 + 8x_2 + 4x_3 \\
\text{subj.to} \quad & 3x_1 + 5x_2 + 2x_3 \geq 30 \\
& \text{alldiff}(x_1, x_2, x_3) \\
& x_j \in \{1,2,3,4\}
\end{aligned}
$$

MILP needs additional constraints and 0-1 variables to express the alldiff:

$$
\begin{aligned}
& x_i = \sum_j j y_{ij}, \quad \text{all } i \\
& \sum_j y_{ij} = 1, \quad \text{all } i
\end{aligned}
$$

Constraints…

$3x_1 + 5x_2 + 2x_3 \geq 30$	**Branching:** *N/A* **Inference:** **Domain filtering (bounds consistency)** **Relaxation:** • LP (send inequality to the LP solver) • CS (send inequality to the CP solver)
$\text{alldiff}(x_1,x_2,x_3)$	**Branching:** $x_i=x_k \Rightarrow x_i<x_k \vee x_k<x_i$ **Inference:** **Filtering (hyperarc consistency)** **Relaxation:** • LP: add linearization to the LP (not very useful) • CS: send constraint to the CP solver
$x_j \in \{1,2,3,4\}$	**Branching:** **split domain** **Inference:** **none** **Relaxation:** • LP: add linearization to the LP solver • CS: send constraint to the CP solver

An integrated model in SIMPL

```
DECLARATIONS
    nObjects = 3; nValues = 4;
    discrete range objects = 1 to nObjects;
    discrete range values  = 1 to nValues;
    cost[objects] = [5,8,4]; weight[objects] = [3,5,2]; cap = 30;
    var x[objects] in values;
```

```
OBJECTIVE
    min sum item of cost[item] * x[item]
```

```
CONSTRAINTS

    capacity means {
          sum item of weight[item] * x[item] >= cap;
          relaxation = { LP, CS }
    }

    distinct means {
          alldiff(x);
          relaxation = { CS }
    }
```

```
SEARCH
   type      = { BB : DFS }
   branching = { x : first }
```

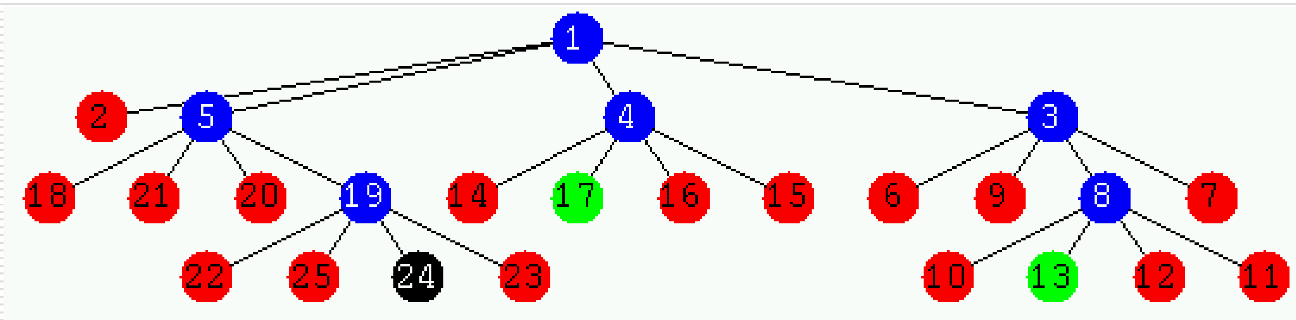
An integrated model in SIMPL

```
CONSTRAINTS

    capacity means {
       sum item of weight[item] * x[item] >= cap;
       relaxation = {LP, CS}
    }

    distinct means {
       alldiff(x);
       relaxation = {CS}
    }
```

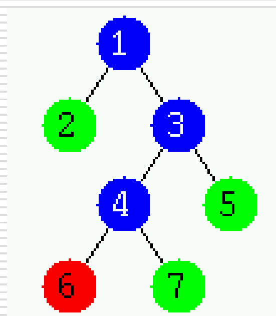
Branching on the **in-domain** constraint of **x**

```
SEARCH
   type      = { BB : DFS }
   branching = { x_: first }
```

Performance: MILP vs. Hybrid



Search tree for pure MIP model: 25 nodes



• **green**	**feasible**
• **black**	**pruned by bound**
• **red**	**Pruned by infeasibility**
• **blue**	**branched on**

Search tree for hybrid model: 7 nodes

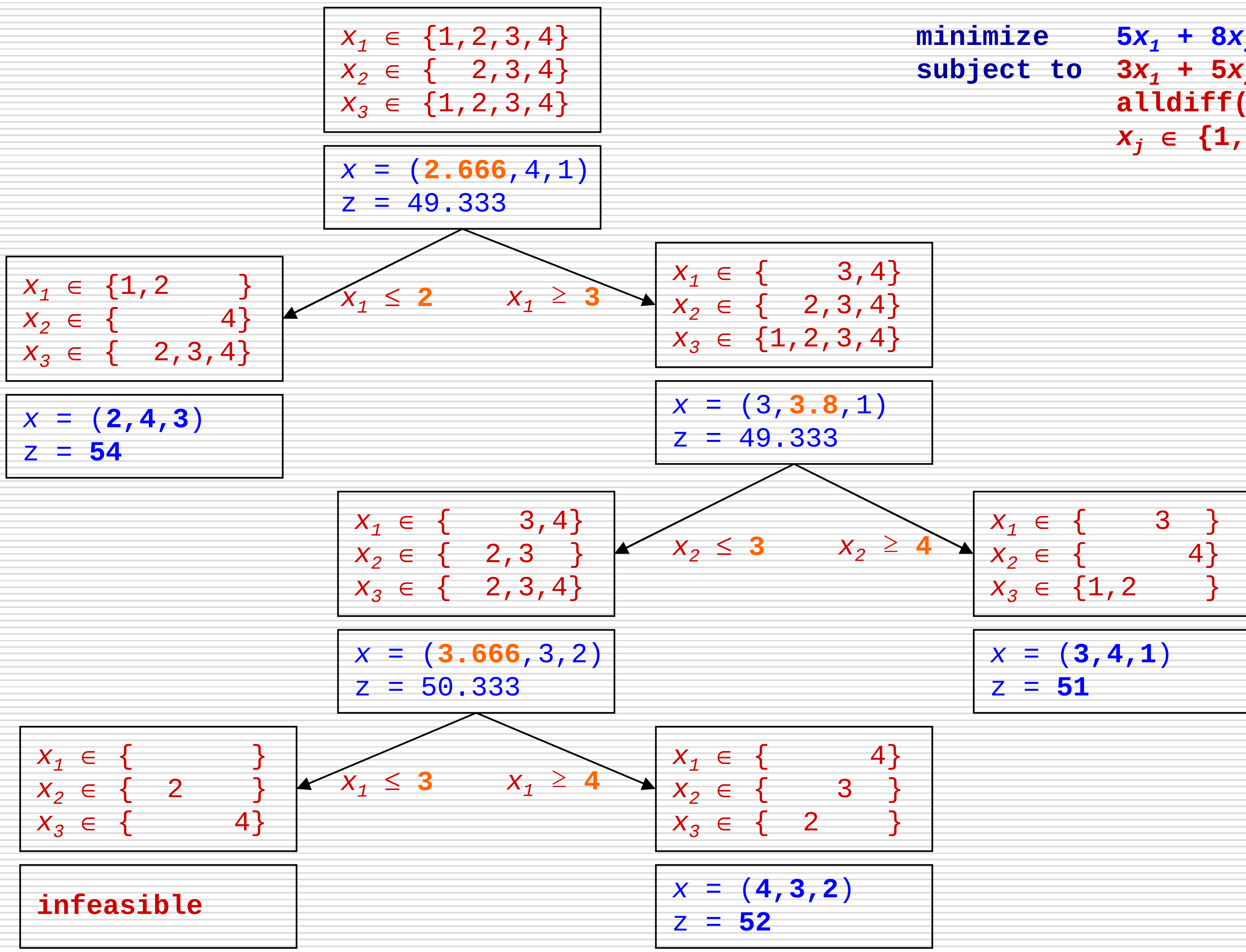
Search tree

minimize $5x_1 + 8x_2 + 4x_3$

subject to $3x_1 + 5x_2 + 2x_3 \geq 30$

$\text{alldiff}(x_1, x_2, x_3)$

$x_j \in \{1,2,3,4\}, \ \forall j$



Execution trace

```
SIMPL, Version 0.3 (beta), October 20, 2003
Copyright (c) GSIA, Carnegie Mellon University

Exploring node 1 (root node):
        Improved dual bound to 49.3333.
        Branching on x1.

Exploring node 2 (child of 1) [open: 1, done: 1]:
        Improved dual bound to 54.
        Found better feasible solution with value 54. Updating incumbent.
        Pruned by local optimality.
...
Exploring node 6 (child of 4) [open: 2, done: 4]:
        Pruned by infeasibility (pre-relaxation).
...
Explored nodes: 7.
Elapsed CPU time: 0 seconds.
Solution value = 51
x[1] = 3
x[2] = 4
x[3] = 1
```

SiMPL

Example – Quadratic Assignment

Quadratic assignment

Assign n facilities to n locations to minimize total travel:

Flow between facilities i and j

Distance between locations x_i and x_j

$$
\begin{aligned}
\min \quad & \sum_{ij} f_{ij} d_{x_i x_j} \\
\text{subj.to} \quad & \text{alldiff}(x_1, \ldots, x_n) \\
& x_j \in \{1, \ldots, n\}
\end{aligned}
$$

Location assigned to facility j

IP model

Requires 0-1 variables with 4 subscripts:

$$\begin{aligned} \min \quad & \sum_{ijkl} f_{ij} d_{kl} y_{ijkl} \\ \text{subj.to} \quad & \sum_{kl} y_{ijkl} = 1, \quad \text{all } i, j \\ & y_{ijkl} \in \{0,1\} \end{aligned}$$

Quadratic assignment in SIMPL

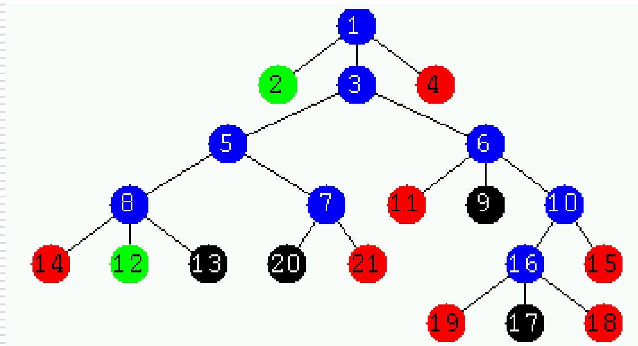
```
DECLARATIONS
    nFacilities = ...; nLocations = nFacilities; maxDistance = ...;
    discrete range facilities = 1 to nFacilities;
    discrete range locations  = 1 to nLocations;
    discrete range meters     = 0 to maxDistance;
    distance[locations,locations] in meters = ...;
    flow[facilities,facilities] in integer = ...;
    var location[facilities] in locations;
    var travel[facilities,facilities] in meters;
```

```
OBJECTIVE
    min sum f1,f2 of flow[f1,f2] * travel[f1,f2]
```

```
CONSTRAINTS

    assign means {
          travel[f1,f2] = distance[location[f1],location[f2]] forall f1,f2;
          relaxation = {LP,CS}
    }

    distinct means {
          alldiff(location);
          relaxation = {CS}
    }
```

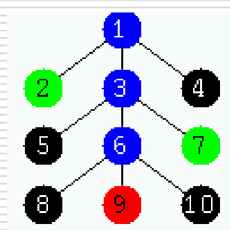
```
SEARCH
   type      = { BB : BBS }
   branching = { location : first, assign : first, distinct : first }
```

Branching options



Branching on the first violated **assign** constraint: 21 nodes

```
SEARCH
   type      = { BB : BBS }
   branching = { location : first, assign : first, distinct : first }
```



Branching on the most violated **assign** constraint: 10 nodes

```
SEARCH
   type      = { BB : BBS }
   branching = { location : first, assign : most, distinct : first }
```

Exploiting structure

Where to exploit structure in the QAP?

There are basically two constraints:

$$
\begin{aligned}
\min \quad & \sum_{ij} f_{ij} d_{x_i x_j} \\
\text{subj.to} \quad & \text{alldiff}(x_1, \ldots, x_n) \\
& x_j \in \{1, \ldots, n\}
\end{aligned}
$$

2-dimensional element constraint. Some potential here: distance matrix may have special structure

$$\text{element}\big((x_i, x_j), d, z_{ij}\big)$$

Not much more we can do here.

Implementation status

- Constraints
  - **Linear inequalities**
  - **SOS1**
  - **Global constraints**
    (element, alldiff, cumulative, sum)
  - **Conditionals** ($\Rightarrow$, $\Leftrightarrow$, $\rightarrow$)

- Relaxations/solvers
  - **LP** (CPLEX,XpressMP,LPSolve)
  - **MIP** (CPLEX, XpressMP)
  - **CS** (ECLiPSe)

Implementation status

- Search
  - **Tree search** (Branch and bound)
  - **Node selection**
    (depth-first, breadth-first, best-bound)
- User interface
  - **High-level modeling language** and/or C++ library API
  - On-the-fly **execution statistics**
  - **Search tree visualization**
  - Currently working on **modeling GUI**