A System For **Integrating** Optimization Techniques

Ionut Aron  
Brown University

John Hooker, Tallys Yunes  
Carnegie Mellon University

APMOD, Brunel University, June 2004
A System For Integrating Optimization Techniques
Outline

Why Integrated Methods?
SIMPL Philosophy
Architecture
Modeling Examples
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

Objective: minimize cost, weight, etc.

Computer chassis

Wattage = \( w_1 \)
Space req. = \( s_1 \)

Wattage = \( w_2 \)
Space req. = \( s_2 \)

Memory chip options

Capacity = \( c_1 \)
Power use = \( p_1 \)

Capacity = \( c_2 \)
Power use = \( p_2 \)

Requirements: bounds on capacity, power usage

Power supply options

Requirements: bounds on wattage, space
Problem formulation

\[
\min_{t, q, r} \sum_{k} c_k \cdot r_k \\
\text{subj. to } \sum_{i} q_i \cdot A_{kit_i} = r_k, \quad \forall k \\
q_i \geq 0, \quad \forall i \\
r_k \geq R_k, \quad \forall k \\
t_i \in \{\text{component types}\}
\]

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

\[
\begin{align*}
\text{min} & \quad \sum_{k} c_k r_k \\
\text{subj.to} & \quad r_k = \sum_{ij} q_{ij} A_{kij}, \quad \text{all } k \\
& \quad \sum_{ij} t_{ij} = 1, \quad \text{all } i \\
& \quad \sum_{j} q_{ij} \leq M t_{ij}, \quad \text{all } i, j \\
& \quad r_k \leq R_k, \quad \text{all } k \\
& \quad t_{ij} \in \{0,1\}
\end{align*}
\]
Integrated model

Same as original formulation

\[
\begin{align*}
\text{min} & \quad \sum_{k} c_k r_k \\
\text{subj.to} & \quad r_k = \sum_{i} q_i A_k i_t_i, \quad \text{all } k \\
& \quad r_k \geq R_k, \quad \text{all } k \\
& \quad t_i \in \{\text{component types}\}
\end{align*}
\]

Constraint programming deals routinely with variable indices using the element global constraint.
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 $y$th 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 \textit{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 \( 0 \leq x_j \leq b \) for each \( j \), a convex hull relaxation of

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

is given by

\[
\sum_{j \in D_y} x_j - (n-1)b \leq z \leq \sum_{j \in D_y} x_j
\]

Current domain of \( y \)
Domain filtering for \textit{element}

Example...

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

The initial domains are:  \quad The reduced domains are:

\begin{align*}
D_z &= \{20,30,60,80,90\} & D_z &= \{80,90\} \\
D_y &= \{1,3,4\} & D_y &= \{3\} \\
D_{x_1} &= \{10,50\} & D_{x_1} &= \{10,50\} \\
D_{x_2} &= \{10,20\} & D_{x_2} &= \{10,20\} \\
D_{x_3} &= \{40,50,80,90\} & D_{x_3} &= \{80,90\} \\
D_{x_4} &= \{40,50,70\} & D_{x_4} &= \{40,50,70\}
\end{align*}
Computational results

From: Ottosson & Thorsteinsson, 2001

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

![Graph showing computational results for CPLEX, CLP, and Hybrid methods. The x-axis represents different problem sizes (8x10, 16x20, 20x24, 20x30), and the y-axis represents time in seconds. The graph shows increasing time as problem size increases.]
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.

Task 1

Task 2

Task 3

Task 4

Task 5

Resources

Time

$p_1$

$c_1$

$C$

Total resource consumption $\leq C$ at all times.
Allocation + cumulative scheduling

**Facility 1**
- \( C_1 \)
- \( p_{11} \)
- task 1
- task 4
- task 5

**Facility 2**
- \( C_2 \)
- \( c_{22} \)
- task 3
- task 2

Total resource consumption \( \leq C_i \) at all times.
Problem formulation

\[
\begin{align*}
\text{min} & \quad \sum_{j} F_{x_j j} \\
\text{subj.to} & \quad \sum_{j} c_{x_j j} \leq C_i, \quad \text{all } i \\
& \quad t_j \leq t \leq t_j + p_{x_j j} \\
& \quad r_j \leq t_j \leq d_j - p_{x_j j}, \quad \text{all } j \\
& \quad x_j \in \{\text{facilities}\}
\end{align*}
\]
Discrete-time MILP model

\[
\begin{align*}
\text{min} & \quad \sum_{ijt} F_{ij} x_{ijt} \\
\text{subject to} & \quad \sum_{it} x_{ijt} = 1, \quad \text{all } j \\
& \quad \sum_{j} \sum_{t'} c_{ij} x_{ijt'} \leq C_i, \quad \text{all } i, t
\end{align*}
\]

- \( t - p_{ij} < t' \leq t \)
- \( x_{ijt} = 0, \quad \text{all } j, t \) with \( d_j - p_{ij} < t \)
- \( x_{ijt} = 0, \quad \text{all } j, t \) with \( t > N - p_{ij} + 1 \)
- \( x_{ijt} \in \{0,1\} \)

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

Task \( j \) starts at one time on one facility

Resources consumed at time \( t \) on facility \( i \)

Time windows
Discrete-event MILP model (continuous time)

Idea: Türkay & Grossmann

\[
\begin{align*}
\min & \quad \sum_{ijk} F_{ijk} x_{ijk} \\
\text{subject to} & \quad \sum_{ik} x_{ijk} = 1, \quad \sum_{ik} y_{ijk} = 1, \quad \text{all } j \\
& \quad \sum_{ij} x_{ijk} + y_{ijk} = 1, \quad \text{all } k \\
& \quad \sum_{ik} x_{ijk} = \sum_{ik} y_{ijk}, \quad \text{all } i, j \\
& \quad t_{i,k-1} \leq t_{ik} \\
\end{align*}
\]

= 1 if event \( k \) is start of task \( j \) on facility \( i \) \((k = 1, \ldots, 2N)\)

Start time of event \( k \)
(disaggregated by facility)

continued…
\begin{align*}
0 \leq t_{ik}, \quad \left( f_{ij} \right) \leq d_j, & \quad \text{all } i, j, k \\
t_{ik} + p_{ij}x_{ik} - M (1 - x_{ijk}) & \leq f_{ij} \leq t_{ik} + p_{ik}x_{ijk} + M (1 - x_{ijk}), \quad \text{all } i, j, k \\
t_{ik} - M (1 - y_{ijk}) & \leq f_{ij} \leq t_{ik} + M (1 - y_{ijk}), \quad \text{all } i, j, k \\
R_{ik} & \leq C_i, \quad \text{all } i, k \\
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\}
\end{align*}

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

Resource limit

Calculation of resource consumption on facility \( i \) at time of each event
Cumulative scheduling in CP

\[
\text{cumulative}{\begin{pmatrix}
(t_1, \ldots, t_n) \\
(p_1, \ldots, p_n) \\
(c_1, \ldots, c_n) \\
C
\end{pmatrix}} \text{ is equivalent to}
\sum_j c_j \leq C \quad \text{all } t
\]

\[t_j \leq t < t_j + p_{ij}\]

Schedules tasks at times \(t_1, \ldots, t_n\) 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

\[
\begin{align*}
\min & \quad \sum_{j} F_{x_{i,j}} \\
\text{subj.to} & \quad \text{cumulative} \begin{cases} 
(t_j \mid x_j = i) \\
(p_{ij} \mid x_j = i) \\
(c_{ij} \mid x_j = i) \\
C_i
\end{cases}, \quad \text{all } i \\
r_j \leq t_j \leq d_j - p_{x_{i,j}}, \quad \text{all } j \\
x_j \in \{\text{facilities}\}
\end{align*}
\]
Logic-based Benders: Basic idea

**Decompose** problem into

- allocation
- resource-constrained scheduling

Assign tasks to facilities

**Master problem**
Solve with MILP

Schedule tasks on each facility

**Subproblem**
Solve with CP

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

\[ \begin{align*} 
\text{min} & \quad \sum_{ij} g_{ij} x_{ij} \\
\text{subject to} & \quad \sum_{i} x_{ij} = 1, \quad \text{all } j \\
& \quad \text{Benders cuts} \\
& \quad \sum_{j} p_{ij} c_{ij} x_{ij} \leq C_i (d_{\ell} - r_k), \quad \text{all } i, \text{all distinct } r_k, d_{\ell} \\
& \quad r_j \geq r_k \\
& \quad d_j \leq d_{\ell} 
\end{align*} \]

Relaxation of subproblem: “Area” of tasks in time window \([r_k, d_{\ell}]\) must fit.
Subproblem: Schedule tasks

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.

\[
\begin{align*}
\text{cumulative} & \left\{ \begin{array}{l}
(t_j | \overline{x}_{ij} = 1) \\
(p_{ij} | \overline{x}_{ij} = 1) \\
(c_{ij} | \overline{x}_{ij} = 1) \\
C_i
\end{array} \right. \\
r_j \leq t_j \leq d_j
\end{align*}
\]
Master problem with Benders cuts

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

Average computation time
(2, 3 and 4 facilities)

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.
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
- Relax
- Infer
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.
The architecture of SIMPL

SIMPL maintains its own domains for all the variables in the original model.

Domain changes are propagated to and from relaxations in the form of in-domain constraints.

- **LP Relaxation**
- **MIP Relaxation**
- **CP Relaxation**
- **NLP Relaxation**

**Simulated**

**SIMPL Domain Store**

**LP Solver**

**MIP Solver**

**CP Solver**

**NLP Solver**

**LPSolve**

**ECLiPSe**

**CPLEX**

**N/A**
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.

- NLP Relaxation
- MIP Relaxation
- CP Relaxation
- LP Relaxation

The SIMPL Domain Store

N/A

CPLEX

ECLiPSe

Xpress
Examples

Knapsack with side constraint (in detail)

Quadratic assignment
An integer knapsack problem with side constraint

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

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

\[
\begin{align*}
x_i &= \sum_j jy_{ij}, \quad \text{all } i \\
\sum_j y_{ij} &= 1, \quad \text{all } i
\end{align*}
\]
## Constraints...

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Branching</th>
<th>Inference</th>
<th>Relaxation</th>
</tr>
</thead>
</table>
| \(3x_1 + 5x_2 + 2x_3 \geq 30\) | N/A | Domain filtering (bounds consistency) | - LP: send inequality to the LP solver  
- CS: send inequality to the CP solver |
| \(\text{alldiff}(x_1,x_2,x_3)\) | \(x_i = x_k \Rightarrow x_i < x_k \lor x_k < x_i\) | Filtering (hyperarc consistency) | - LP: add linearization to the LP (not very useful)  
- CS: send constraint to the CP solver |
| \(x_j \in \{1,2,3,4\}\) | split domain | none | - LP: add linearization to the LP solver  
- CS: send constraint to the CP solver |
An integrated model in **SIMPL**

**DECLARATIONS**

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

**OBJECTIVE**

\[
\text{min } \sum \text{ item of cost[item] } \times x[\text{item}]
\]

**CONSTRAINTS**

\[
\text{capacity means } \\
\quad \sum \text{ item of weight[item] } \times x[\text{item}] \geq \text{ cap}; \\
\quad \text{relaxation} = \{ \text{ LP, CS } \}
\]

\[
\text{distinct means } \\
\quad \text{alldiff}(x); \\
\quad \text{relaxation} = \{ \text{ CS } \}
\]

**SEARCH**

\[
\text{type } = \{ \text{ BB : DFS } \} \\
\text{branching } = \{ x : \text{ first } \}
\]
An integrated model in SIMPL

**CONSTRAINTS**

- **capacity** means
  - \( \text{sum item of weight[item] * x[item] >= cap} \)
  - relaxation = \{ LP, CS \}

- **distinct** means
  - \( \text{alldiff(x)} \)
  - relaxation = \{ CS \}

**SEARCH**

- type = \{ BB : DFS \}
- branching = \{ x : first \}

Branching on the in-domain constraint of x
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
minimize \( 5x_1 + 8x_2 + 4x_3 \)
subject to

\[
\begin{align*}
3x_1 + 5x_2 + 2x_3 & \geq 30 \\
\text{alldiff}(x_1, x_2, x_3) \\
x_j & \in \{1, 2, 3, 4\}, \quad \forall j \\
x_1 & \in \{1, 2, 3, 4\} \\
x_2 & \in \{2, 3, 4\} \\
x_3 & \in \{1, 2, 3, 4\} \\
z & = (2.666, 4, 1)
\end{align*}
\]

**Search tree**

- \( x_1 \in \{1, 2, 3, 4\} \)
- \( x_2 \in \{2, 3, 4\} \)
- \( x_3 \in \{1, 2, 3, 4\} \)
- \( z = 49.333 \)

**Nodes**

- \( x_1 \in \{1, 2, 3, 4\} \)
- \( x_2 \in \{2, 3, 4\} \)
- \( x_3 \in \{1, 2, 3, 4\} \)
- \( z = 54 \)

**Branches**

- \( x_1 \leq 2 \)
- \( x_1 \geq 3 \)
- \( x_1 \leq 3 \)
- \( x_1 \geq 4 \)

**Infeasible**

- \( x_1 \in \{3, 4\} \)
- \( x_2 \in \{2, 3, 4\} \)
- \( x_3 \in \{1, 2, 3, 4\} \)
- \( z = 52 \)
Exploring node 1 (root node):
   Improved dual bound to 49.3333.
   Branching on $x_1$.

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
$x[1] = 3$
$x[2] = 4$
$x[3] = 1$
Example – Quadratic Assignment
Quadratic assignment

Assign \( n \) facilities to \( n \) locations to minimize total travel:

\[
\min \sum_{ij} f_{ij} d_{x_i x_j}
\]

subj.to \( \text{alldiff}(x_1, \ldots, x_n) \)

\( x_j \in \{1, \ldots, n\} \)

Location assigned to facility \( j \)

Flow between facilities \( i \) and \( j \)

Distance between locations \( x_i \) and \( x_j \)
IP model

Requires 0-1 variables with 4 subscripts:

\[
\begin{align*}
\text{min} & \quad \sum_{ijkl} f_{ijkl} d_{kl} y_{ijkl} \\
\text{subj.to} & \quad \sum_{kl} y_{ijkl} = 1, \quad \text{all } i, j \\
& \quad y_{ijkl} \in \{0,1\}
\end{align*}
\]
**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:

\[
\min \sum_{ij} f_{ij} d_{x_i x_j}
\]

subj.to \(\text{alldiff}(x_1, \ldots, x_n)\)
\(x_j \in \{1, \ldots, n\}\)

Not much more we can do here.

2-dimensional element constraint. Some potential here: distance matrix may have special structure element\(((x_i, x_j), d, z_{ij})\)
Constraints

- Linear inequalities
- SOS1
- Global constraints

Relaxations/solvers

- LP (CPLEX, XpressMP, LPSolve)
- MIP (CPLEX, XpressMP)
- CS (ECLiPSe)

Implementation status

Conditional (\(\iff\))

Global constraints

- SOS1
- Linear inequalities

Implementation status
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**