SIMPL: An Integrated Solver for Optimization Problems

Ionuț D. Aron, IBM T. J. Watson Research Center
John Hooker, Carnegie Mellon University
Tallys H. Yunes, University of Miami

March 2007
Outline

- Intro and Motivation

- **SIMPL**: Our General Purpose System

- Examples
  - Production Planning
  - Product Configuration
  - Machine Scheduling

- Future Work
Why Integrate?

- **User convenience:** several methods and their combinations available in one package.

- **Simpler models:** constraint-programming style modeling adapted to general optimization.
  - Model communicates problem structure to solver.
  - It uses *metaconstraints* = generalization of global constraints in constraint programming

- **Better performance:** combine the complementary strengths of different methods.
  - Solver chooses best mix of relaxation and propagation methods, based on metaconstraints.
SIMPL Objectives

- **High-level** modeling language
  - Concise and **easily understandable** models
  - **Natural** specification of integrated models
  - Allow the modeler to **reveal problem structure** to the solver

- **Micro-level** integration
  - Integrated methods are more effective when the underlying technologies interact at **micro level** throughout the search process.
Models vs. problems

- **Models** should be distinguished from **problem instances**.
  - MIPLIB is a collection of **models**, not problem instances.
    - Unfortunately, the problem original description is often unknown.
    - So it’s hard to write a better model.

- **A model should reveal problem structure.**
  - Much as a **scientific model**.
  - MIPLIB instances are better seen as **formulations** than models.
    - They are not **explanatory**.
Algorithmic Idea behind **SIMPL**

- **CP** and **IP** are special cases of a general method, not separate methods to be combined.

- Common solution strategy:
  
  Search – Infer – Relax

- **Search** = enumeration of problem restrictions
For example: Classical Solution Methods

- **Branch and Cut (IP)**
  - **Inference**: preprocessing at root node, cutting planes
  - **Relaxation**: dropping integrality constraints
  - **Search**: branching on integrality constraints

- **Typical CP solver**
  - **Inference**: domain reduction
  - **Relaxation**: collecting reduced domains into constraint store
  - **Search**: domain splitting

- **Benders decomposition**
  - **Inference**: Benders cuts
  - **Relaxation**: master problem
  - **Search**: creating sub-problems by fixing “hard” variables

- **(Meta) Heuristics**
**Constraints, Constraints**

**Constraint-oriented modeling and solution**

- **Infer:** constraints **drive the inference**
  - Each constraint has a filtering/inference module
  - This module provides new constraints to **tighten the relaxation**

- **Relax:** constraints **determine the relaxations**
  - Each constraint has a relaxation module
  - This module **reformulates the constraint** and sends it to the appropriate relaxation (LP, CP, MIP etc)

- **Search:** constraints **direct the search**
  - Each constraint has a branching module
  - This module **creates new problem restrictions**
Example 1: Production Planning

- Manufacture several products to maximize net income

- Each product has several production modes (small scale, medium scale, etc.)

- Only certain ranges of production quantities are possible (gaps in the domain)

- Net income function $f(x)$ is semi-continuous piecewise linear (non-convex and non-concave)
Production Planning: $f(x)$
Production Planning: MIP Model

- \( x_i \) = quantity of product \( i \)
- \( y_{ik} = 1 \) if product \( i \) is manufactured in mode \( k \)
- \( \lambda_{ik}, \mu_{ik} \) = weights for mode \( k \) (convex comb.)

\[
\begin{align*}
\text{max} & \sum_{ik} (\lambda_{ik} c_{ik} + \mu_{ik} d_{ik}) \\
\sum_i x_i & \leq C \\
x_i & = \sum_k (\lambda_{ik} L_{ik} + \mu_{ik} U_{ik}), \forall i \\
\sum_k (\lambda_{ik} + \mu_{ik}) & = 1, \forall i
\end{align*}
\]

\[
\begin{align*}
0 & \leq \lambda_{ik} \leq y_{ik}, \forall i, k \\
0 & \leq \mu_{ik} \leq y_{ik}, \forall i, k \\
\sum_k y_{ik} & = 1, \forall i \\
y_{ik} & \in \{0, 1\}, \forall i, k
\end{align*}
\]
Production Planning: Integrated

- \( x_i = \) quantity of product \( i \)
- \( y_i = \) net income from product \( i \)

\[
\begin{align*}
\max \sum_i y_i \\
\sum_i x_i &\leq C \\
piecewise(x_i, y_i, L_i, U_i, c_i, d_i), \forall i \\
x_i \in D_i
\end{align*}
\]
Production Planning: **Integrated**

**SIMPL Model**

**OBJECTIVE**

maximize sum i of u[i]

**CONSTRAINS**

capacity means {
  sum i of x[i] <= C
  relaxation = { lp, cp } }

piecewisectr means {
  piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
  relaxation = { lp, cp } }

**SEARCH**

type = { bb:bestdive }
branching = { piecewisectr:most }
Relaxation of piecewise constraint
Branching on piecewise constraint

f(X)

x value OK, y value OK: no problem
Branching on piecewise constraint

x value not OK:
Branching on piecewise constraint

f(X)  x value not OK: split domain

child 1  child 2
Branching on piecewise constraint

\[ f(X) \]

x OK, y not OK:
Branching on piecewise constraint

f(X)

x OK, y not OK: 3-way branch on x

child 1

child 2

child 3
Computational Results

- **SIMPL**
  - LP solver = CPLEX 9.0.
  - CP solver = Eclipse 5.8.

- **MILP**
  - CPLEX 9.0

- **Hardware**
  - Pentium 4, 3.7 GHz, 4GB RAM.
  - Gentoo Linux kernel 2.6.12.5
The aim is **not** to show that integrated methods can be faster.  
- This is shown in a growing literature.

The aim is to show that SIMPL can achieve **same or better speedups** than hand-crafted integrated methods in the literature.  
- With **no coding** and a **simple model** (simpler than MILP).
Computational Results: Search Nodes

All products have the same cost structure
Symmetry breaking: $x_i \leq x_{i+1}$
Computational Results: Time (s)

- Time (s)
- Number of Products

Log(Time (s))

CPLEX
SIMPL
Computational results

- These speedups are better than those reported in the literature.
  - Refalo 1999
Example 2: Product Configuration

- A product (e.g. computer) is made up of several components (e.g. memory, cpu, etc.)
- Components come in different types
- Type $j$ of component $i$ consumes/produces $A_{ijk}$ units of resource $k$
- There are lower and upper bounds on resource consumption/production
- $c_k =$ unit cost of resource $k$
- Minimize total cost
Product Configuration: **MIP Model**

- \( v_k = \text{total consumption/production of resource } k \)
- \( x_{ij} = \text{whether or not type } j \text{ is chosen for component } i \)
- \( q_{ij} = \text{units of type } j \text{ of component } i \)

\[
\begin{align*}
\min & \quad \sum_k c_k v_k \\
\text{s.t.} & \quad v_k = \sum_{i,j} A_{ijk} q_{ij}, \forall k \\
& \quad q_{ij} \leq M_i x_{ij}, \forall i, j \\
& \quad \sum_j x_{ij} = 1, \forall i \\
& \quad L_k \leq v_k \leq U_k, \forall k \text{ and } x_{ij} \in \{0, 1\}, \forall i, j
\end{align*}
\]
Product Configuration: **Integrated**

- $v_k = \text{total consumption/production of resource } k$
- $q_i = \text{quantity of component } i$
- $t_i = \text{type of component } i$

\[
\min \sum_k c_k v_k \\
\]

\[
v_k = \sum_i q_i A_{ijt_i}, \ \forall \ k \\
v_k \in D_k
\]

Modeled internally with an **element** constraint
OBJECTIVE
   minimize sum j of c[j]*v[j]
CONSTRAINTS
   usage means {
      v[j] = sum i of q[i]*a[i][j][t[i]] forall j
   relaxation = { lp, cp }
   inference = { knapsack } } 
quantities means {
   q[1] >= 1 => q[2] = 0
   relaxation = { lp, cp } }
types means {
   t[1] = 1 => t[2] in {1,2}
   relaxation = { lp, cp } }
SEARCH
   type = { bb:bestdive }
   branching = { quantities, t:most, q:least:triple, types:most } 
   inference = { q:redcost }
## Computational Results

<table>
<thead>
<tr>
<th>Instance</th>
<th>CPLEX</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Time (s)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>3</td>
<td>184</td>
<td>0.79</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.04</td>
</tr>
<tr>
<td>5</td>
<td>723</td>
<td>4.21</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.05</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>0.59</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
<td>0.19</td>
</tr>
<tr>
<td>9</td>
<td>20</td>
<td>0.17</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0.03</td>
</tr>
</tbody>
</table>
First application of integrated method was much faster than CPLEX at that time.

- CPLEX 7.0 stopped after 100,000 nodes on 7 of the 10 instances.
- It solved the remaining 3 with 77,000 nodes.
- Thorsteinsson and Ottosson (2001) used the integrated method to solve these instances in about the same time as SIMPL.
Example 3: Machine Scheduling

- Given $n$ tasks and $m$ machines (disjunctive)
- It costs $c_{ij}$ to process task $i$ on machine $j$
- Processing time of task $i$ on machine $j$ is $p_{ij}$
- Task $i$ has release date $r_i$ and due date $d_i$
- **Goal:** schedule all tasks and minimize total cost

- Use Hybrid IP/CP Benders Decomposition approach
Benders Decomposition

- **Master Problem**
  - Assign tasks to machines at minimum cost
  - Regardless of release dates and due dates
  - $x_{ij} = 1$ if task $i$ assigned to machine $j$

- **Subproblems**
  - Jobs in $I_j$ assigned to machine $j$.
  - Try to find feasible schedule (with given tasks)
  - If *infeasible* for mach $j$, generate the Benders cut:
    \[
    \sum_{i \in I_j} x_{ij} \leq |I_j| - 1
    \]
OBJECTIVE
   \[ \text{min} \ \text{sum}_{i,j} \ \text{of} \ c[i][j] \times x[i][j]; \]
CONSTRAINTS
assign means {
    \text{sum \ of} \ x[i][j] = 1 \ \text{forall} \ j;
    \text{relaxation} = \{ \text{ip:master} \} }
xy means {
    x[i][j] = 1 \iff y[j] = i \ \text{forall} \ i, j;
    \text{relaxation} = \{ \text{cp} \} }
tbounds means {
    r[j] \leq t[j] \ \text{forall} \ j;
    t[j] \leq d[j] - p[y[j]][j] \ \text{forall} \ j;
    \text{relaxation} = \{ \text{ip:master, cp} \} }
machinecap means {
    \text{cumulative}({ t[j], p[i][j], 1 } \ \text{forall} \ j | \ x[i][j] = 1, 1) \ \text{forall} \ i;
    \text{relaxation} = \{ \text{cp:subproblem, ip:master} \}
    \text{inference} = \{ \text{feasibility} \} }
SEARCH
  type = \{ \text{benders} \}
Results compared with:
- CPLEX 9.0 (ILOG Scheduler is generally slower).
## Computational Results: Time (s)

<table>
<thead>
<tr>
<th>n, m</th>
<th>( p_{ij} )</th>
<th>Best Comm</th>
<th>J&amp;G</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3, 2</td>
<td>long</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>short</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7, 3</td>
<td>long</td>
<td>0.10</td>
<td>0.52</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>short</td>
<td>0.27</td>
<td>0.02</td>
<td>0.00</td>
</tr>
<tr>
<td>12, 3</td>
<td>long</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>short</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15, 5</td>
<td>long</td>
<td>91.59</td>
<td>2.25</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>short</td>
<td>5.58</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>20, 5</td>
<td>long</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>short</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Never more than 31 iterations and 60 cuts in Benders approach
Machine Scheduling in SIMPL

- **Extensions:**
  - **Cumulative** (resource-constrained) scheduling in subproblem.
  - **Other objective functions** use more interesting logic-based Benders cuts and relaxations.
    - Makespan *(JNH 2004)*
    - Number of late jobs *(JNH 2005)*
    - Total tardiness *(JNH 2005, 2007)*
  - Equally good speedups in most cases.
Future Work

- Increase SIMPL’s **functionality**: add new constraints, solvers, search mechanisms, etc.
  - Easy to add new constraints.

- Find valid **propagation methods** and **relaxations** for more constraints.

- Release a beta version.

- Allow users to contribute their own constraints.
  - Create a **modeling “Wikipedia”**?