Recent Results for SIMPL
An Integrated Solver

Tallys Yunes
Ionut Aron
John Hooker

ISMP 2009

SIMPL Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.

SIMPL Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.
- **Low-level** integration with **high-level** modeling.

SIMPL Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.
- **Low-level** integration with **high-level** modeling.
- Succinct modeling with **meta-constraints**.
  - Model communicates **problem structure** to the solver.

SIMPL Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.
- **Low-level** integration with **high-level** modeling.
- Succinct modeling with **meta-constraints**.
  - Model communicates **problem structure** to the solver.
- General **search-infer-relax** solution algorithm.
  - Enumerate problem restrictions.
  - Branching or logic-based Benders.
  - Underlying search/inference and search/relaxation dualities.

SIMPL Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.
- **Low-level** integration with **high-level** modeling.
- Succinct modeling with **meta-constraints**.
  - Model communicates **problem structure** to the solver.
- General **search-infer-relax** solution algorithm.
  - Enumerate problem restrictions.
  - Branching or logic-based Benders.
  - Underlying search/inference and search/relaxation dualities
- **Constraint-based** control.
  - Filtering, relaxation, branching.

SIMPL Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.
- **Low-level** integration with **high-level** modeling.
- Succinct modeling with **meta-constraints**.
  - Model communicates **problem structure** to the solver.
- General **search-infer-relax** solution algorithm.
  - Enumerate problem restrictions.
  - Branching or logic-based Benders.
  - Underlying search/inference and search/relaxation dualities
- **Constraint-based** control.
  - Filtering, relaxation, branching.
- Performance equal to or better than that of **hand-coded** integrated methods in the literature.

Evolution of the solver

- JNH, Logic-based methods for optimization, *CP 1994*.
- JNH and **M. A. Osorio**, Mixed logical/linear programming, *DAM 1999*.
- JNH, **G. Ottosson**, **E. Thorsteinsson**, **H.-J. Kim**, A scheme for unifying optimization and constraint satisfaction methods, *KER 2000*
- JNH, **H.-J. Kim**, and **G. Ottosson**, A declarative modeling framework that integrates solution methods, *Annals of OR* 2001
- **I. Aron**, JNH, **T. Yunes**, SIMPL: A system for integrating optimization techniques, *CPAIOR 2004*
- **T. Yunes**, **I. Aron**, JNH, An integrated solver for optimization problems, *Operations Research*, to appear.

Outline

- Production planning.
  - Semicontinuous piecewise linear functions
- Product configuration.
  - Variable indices
- Machine scheduling.
  - Logic-based Benders
- Truss structure design.
  - Global optimization.

Production Planning

Maximize profit, which is a piecewise linear function of output.

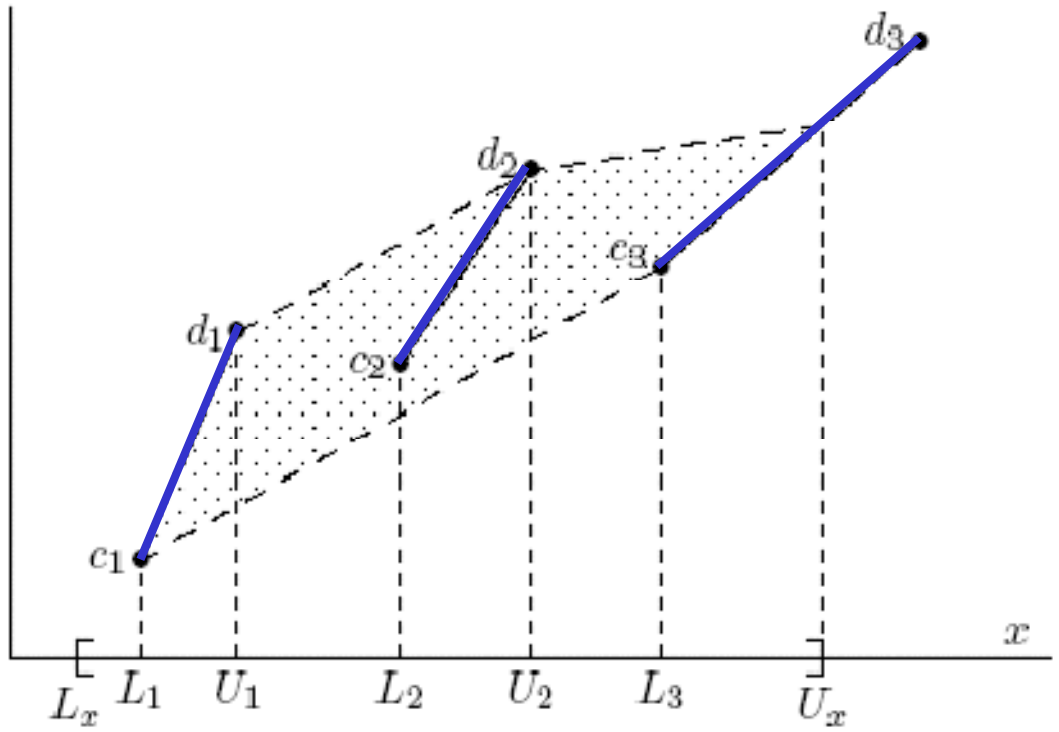
$$\max \sum_i f_i(x_i)$$

$$\sum_i x_i \le C$$

Each f_i is a piecewise linear semicontinuous function

Production Planning

Semicontinuous piecewise linear function $f(x)$



Production Planning

MILP model

SOS2 branching not useful

$$\min \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$

$$\sum_{i} x_i \le C$$

$$x_i = \sum_{k} \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \quad \text{all } i$$

$$\sum_{k} \lambda_{ik} + \mu_{ik} = 1, \quad \text{all } i$$

$$0 \le \lambda_{ik} \le y_{ik}, \quad \text{all } i,k$$

$$0 \le \mu_{ik} \le y_{ik}, \quad \text{all } i,k$$

$$\sum_{k} y_{ik} = 1, \quad \text{all } i$$

$$y_{ik} \in \{0,1\}, \quad \text{all } i,k$$

Production Planning

MILP model

$$\min \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$

$$\sum_{i} x_i \le C$$

SOS2 branching not useful

$$x_i = \sum_{k} \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \quad \text{all } i$$

$$\sum_{k} \lambda_{ik} + \mu_{ik} = 1, \quad \text{all } i$$

$$0 \le \lambda_{ik} \le y_{ik}, \quad \text{all } i,k$$

$$0 \le \mu_{ik} \le y_{ik}, \quad \text{all } i,k$$

$$\sum_{k} y_{ik} = 1, \quad \text{all } i$$

= 1 if x_i is in interval k

$$y_{ik} \in \{0,1\}, \quad \text{all } i,k$$

Production Planning

Integrated model

$$\max \sum_i u_i$$

$$\sum_i x_i \le C$$

$$\text{piecewise}(x_i, u_i, L_i, U_i, c_i, d_i), \quad \text{all } i$$

Metaconstraint

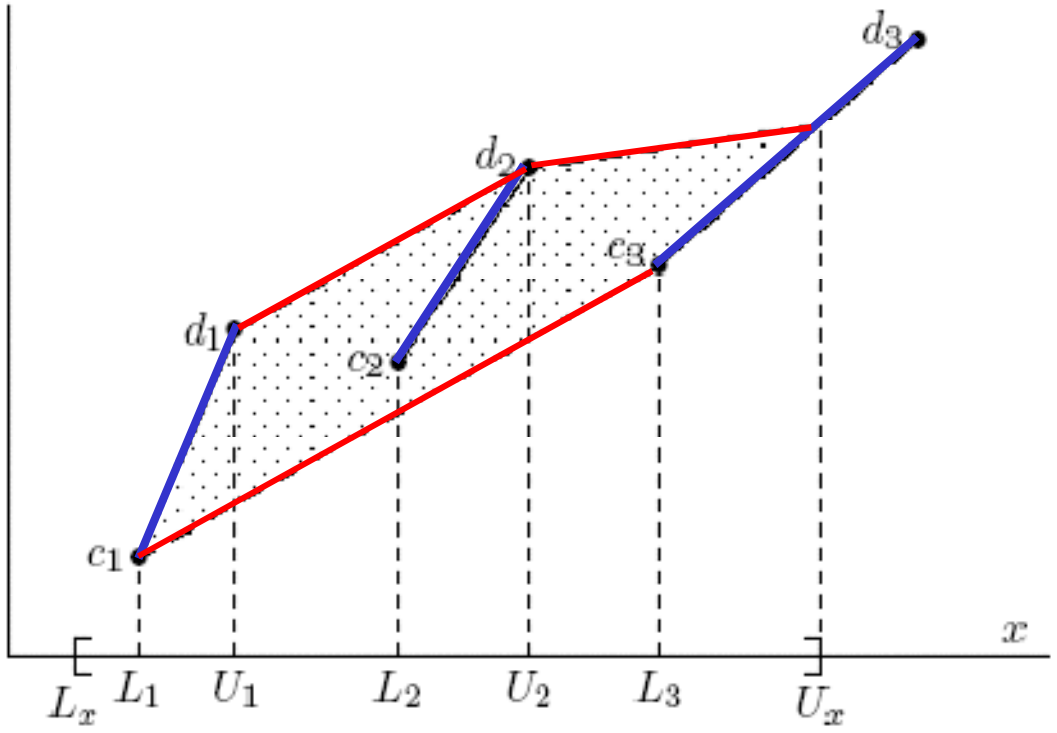
(global constraint in CP)

Original hand-coded method: Ottosson, Thorsteinsson and JNH 1999.

Production Planning

Semicontinuous piecewise linear function $f(x)$

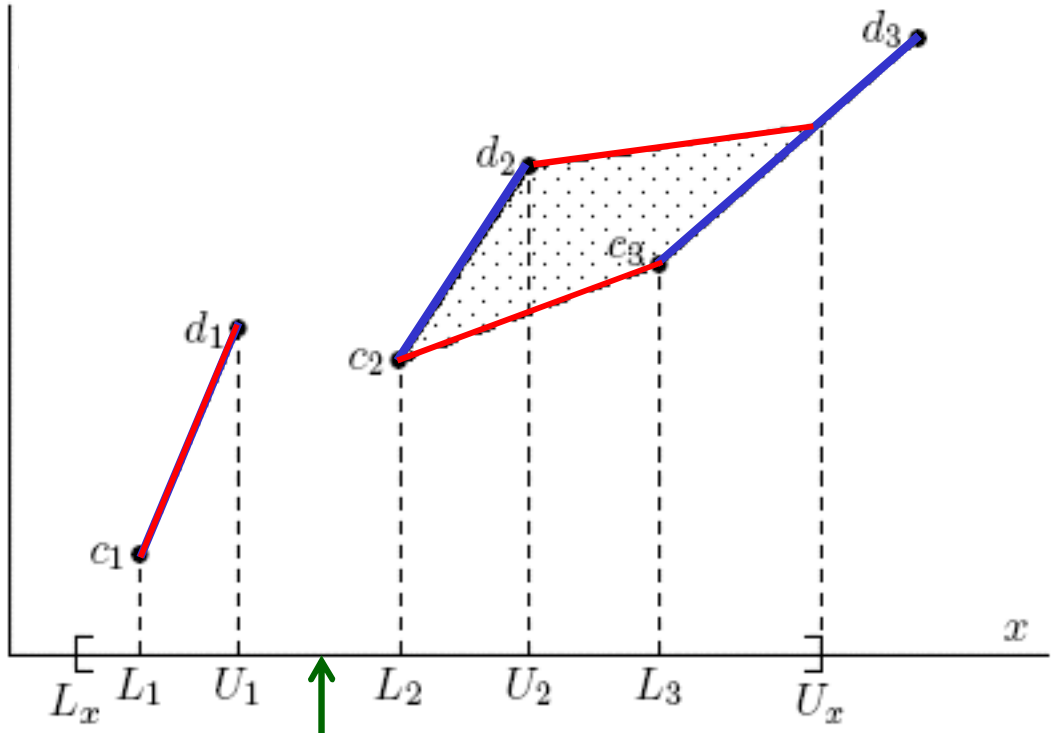
Tight linear relaxation



Production Planning

Semicontinuous piecewise linear function $f(x)$

Tighter relaxation after branching



Value of x in solution of current linear relaxation

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Recognized as a linear system.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Is its own LP relaxation.

CP relaxation propagates bounds.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Piecewise linear metaconstraint.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

LP relaxation is convex hull.

CP relaxation propagates bounds.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Branch-and-bound search.

Dive to leaf node from node with best lower bound.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Branch on piecewise constraint with greatest violation.

Production Planning

Computational Results (seconds)

Hand-coded integrated method was comparable to CPLEX 9

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
50	9.5	2.4	0.43
60	55	1.8	1.1
70	99	2.6	0.82
80	61	4.6	1.25
90	422	6.2	1.7
100	4458	4.4	2.8

Production Planning

CPLEX has become orders of magnitude faster,
but still slower than SIMPL

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
50	9.5	2.4	0.43
60	55	1.8	1.1
70	99	2.6	0.82
80	61	4.6	1.25
90	422	6.2	1.7
100	4458	4.4	2.8

SIMPL's advantage grows with the problem size

Seconds

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
300	82	376	19
300	701	372	19
600	3515	4509	39
600	214	9416	131

SIMPL's advantage grows with the problem size

Seconds

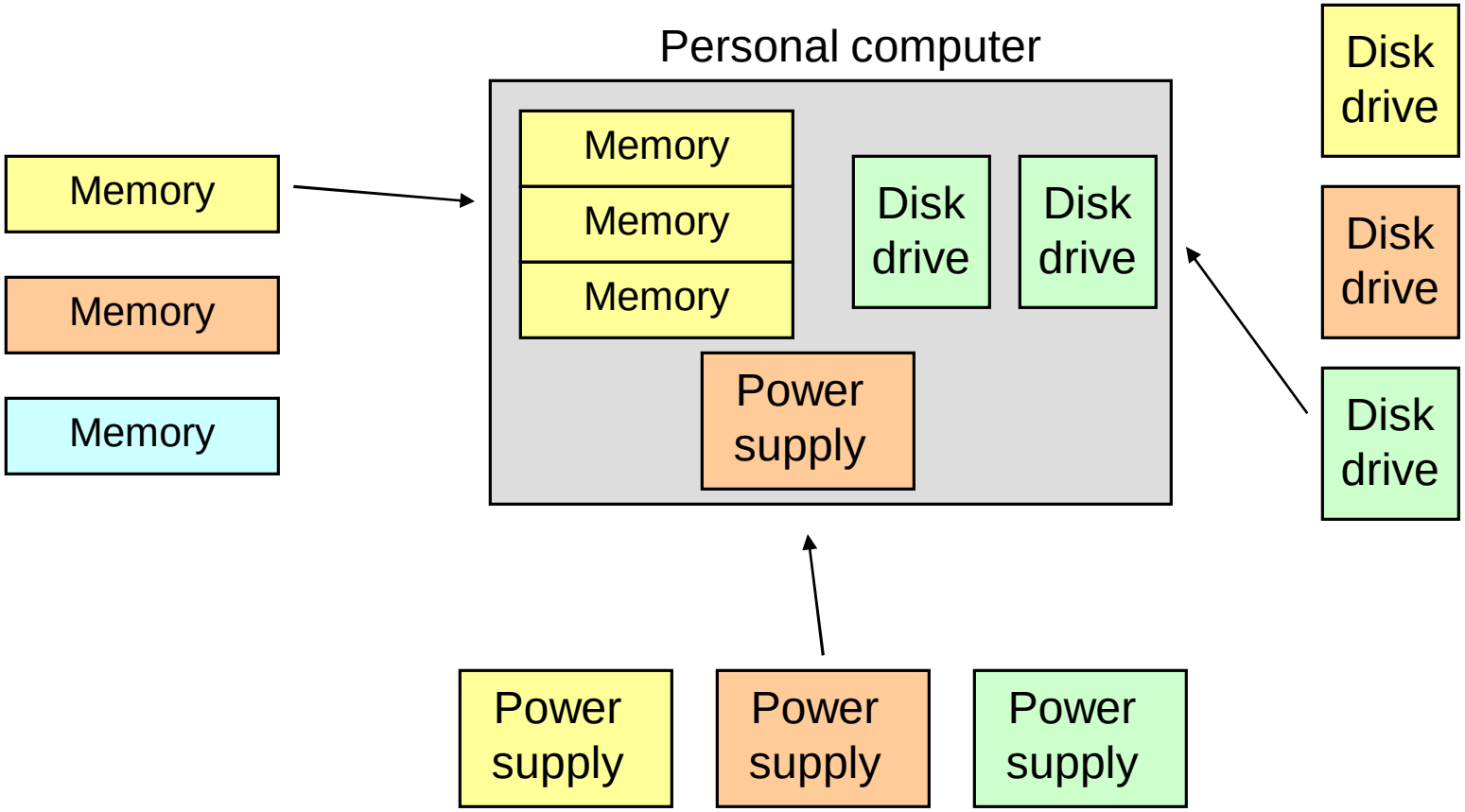
No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
300	82	376	19
300	701	372	19
600	3515	4509	39
600	214	9416	131

Nodes

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
300	10,164	101,756	73
300	43,242	128,333	58
600	363,740	646,907	74
600	7,732	1,297,071	214

Product configuration

Choose what type of each component, and how many



Product configuration

Unit cost of producing attribute *j*

Amount of attribute *j* produced (< 0 if consumed): *memory, heat, power, weight, etc.*

Quantity of component *i* installed

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{tk} A_{ijk} q_{ik}, \text{ all } j$$

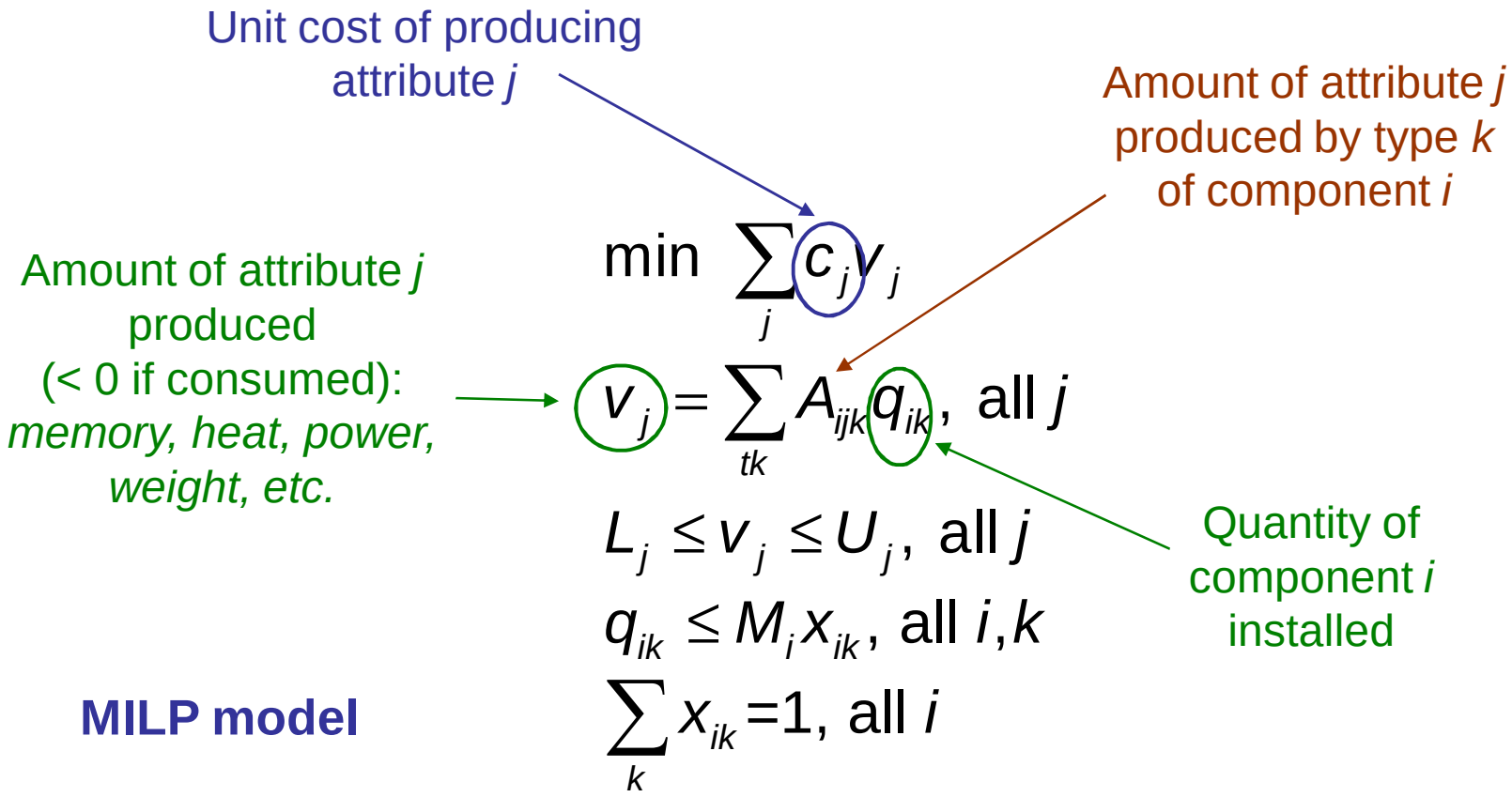
$$L_j \le v_j \le U_j, \text{ all } j$$

$$q_{ik} \le M_i x_{ik}, \text{ all } i,k$$

$$\sum_k x_{ik} = 1, \text{ all } i$$

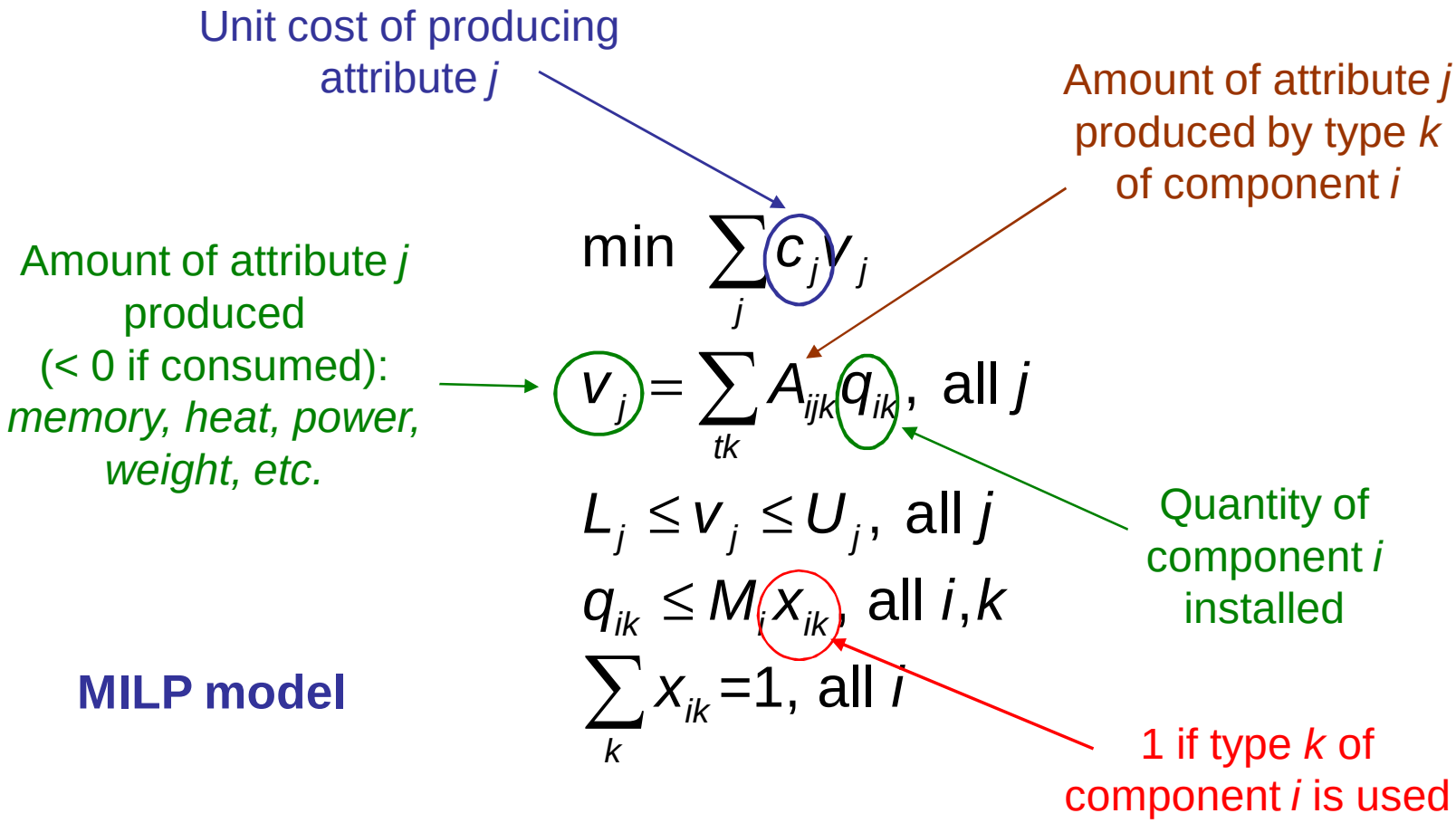
MILP model

Product configuration



MILP model

Product configuration



Product configuration

Unit cost of producing attribute *j*

Amount of attribute *j* produced (< 0 if consumed): *memory, heat, power, weight, etc.*

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} A_{ijt_i} q_i, \text{ all } j$$

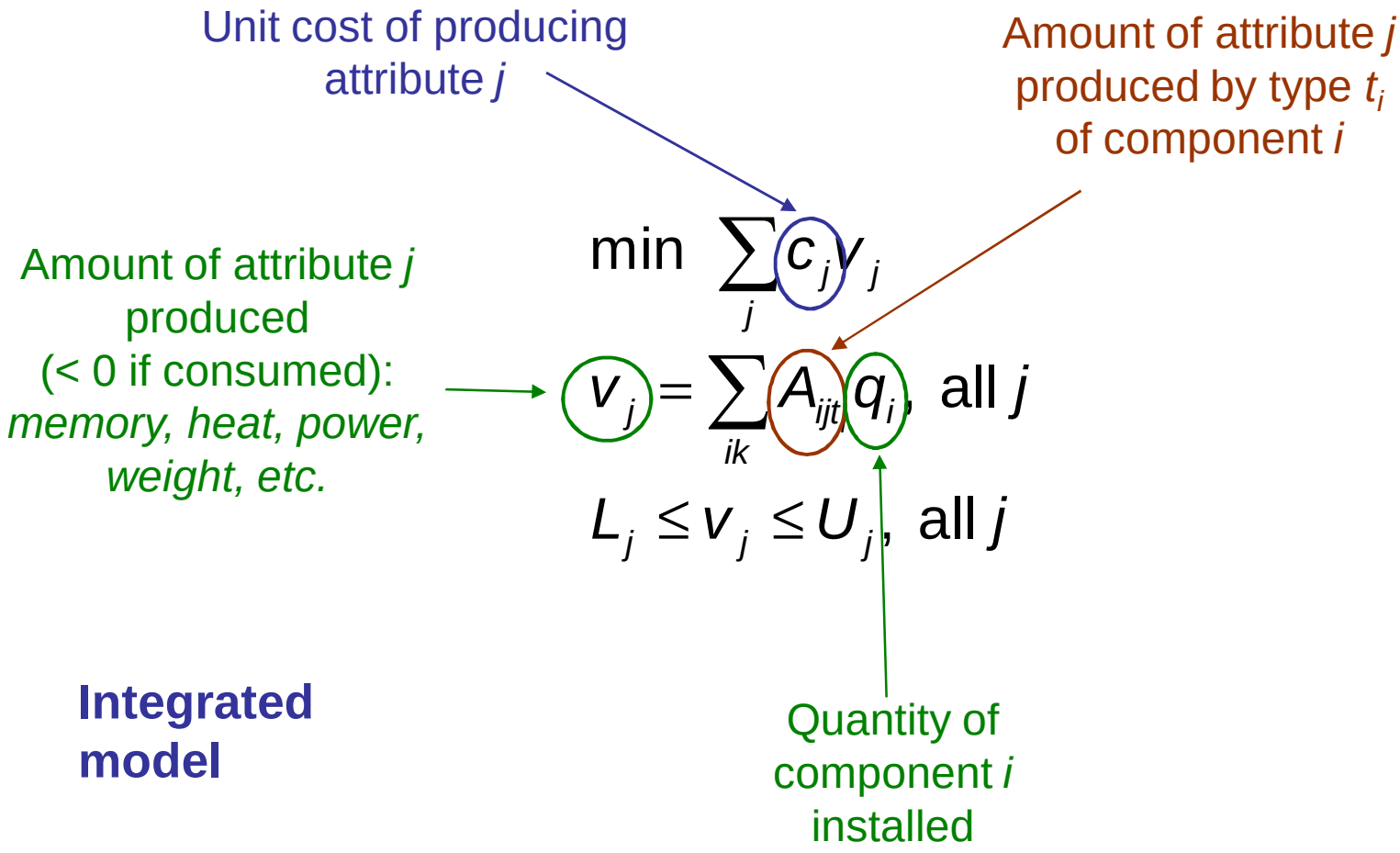
$$L_j \le v_j \le U_j, \text{ all } j$$

Integrated model

Quantity of component *i* installed

Original hand-coded method: Thorsteinsson and Ottosson 2001.

Product configuration



Original hand-coded method: Thorsteinsson and Ottosson 2001.

Product configuration

Unit cost of producing attribute *j*

Amount of attribute *j* produced by type t_i of component *i*

Amount of attribute *j* produced (< 0 if consumed): *memory, heat, power, weight, etc.*

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} A_{ijt_i} q_j, \text{ all } j$$

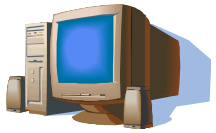
$$L_j \le v_j \le U_j, \text{ all } j$$

t_i is a **variable index**

Integrated model

Original hand-coded method: Thorsteinsson and Ottosson 2001.

Product configuration



Linear inequality
metaconstraint

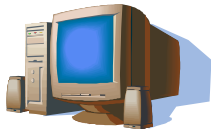
$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$L_j \le v_j \le U_j, \text{ all } j$$

Original hand-coded method: Thorsteinsson and Ottosson 2001.

Product configuration



$$\min \sum_j c_j v_j$$

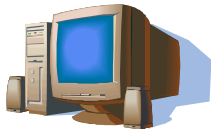
Indexed linear metaconstraint →

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$L_j \le v_j \le U_j, \text{ all } j$$

Original hand-coded method: Thorsteinsson and Ottosson 2001.

Product configuration



Propagation

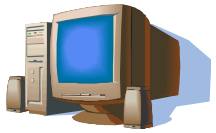
$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$\boxed{L_j \le v_j \le U_j, \text{ all } j}$$

This is propagated in the usual way

Product configuration



Propagation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

$$\min \sum_j c_j v_j$$

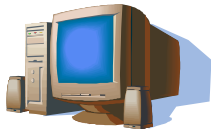
$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

This is rewritten as

$$L_j \le v_j \le U_j, \text{ all } j$$

This is propagated in the usual way

Product configuration



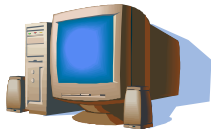
Propagation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

This is propagated by
(a) using specialized **filters** for *element* constraints of this form…

Product configuration



Propagation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

This is propagated by
(a) using specialized **filters** for *element* constraints of this form,
(b) adding **knapsack cuts** for the valid inequalities:

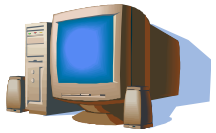
$$\sum_i \max_{k \in D_{t_i}} \{A_{ijk}\} q_i \geq \underline{v}_j, \text{ all } j$$

$$\sum_i \min_{k \in D_{t_i}} \{A_{ijk}\} q_i \leq \overline{v}_j, \text{ all } j$$

and (c) propagating the knapsack cuts.

$[\underline{v}_j, \overline{v}_j]$ is current domain of v_j

Product configuration



Relaxation

$$\min \sum_j c_j v_j$$

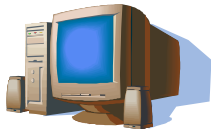
$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$\boxed{L_j \le v_j \le U_j, \text{ all } j}$$

This is relaxed as

$$\underline{v}_j \le v_j \le \bar{v}_j$$

Product configuration



Relaxation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

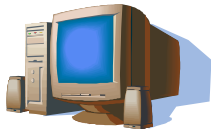
$$L_j \le v_j \le U_j, \text{ all } j$$

This is relaxed by relaxing *this* and adding the knapsack cuts.

This is relaxed as

$$\underline{v}_j \le v_j \le \overline{v}_j$$

Product configuration



Relaxation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

This is relaxed by writing each *element* constraint as a **disjunction** of linear systems and writing a **convex hull** relaxation of the disjunction:

$$z_i = \sum_{k \in D_{t_i}} A_{ijk} q_{ik}, \quad q_i = \sum_{k \in D_{t_i}} q_{ik}$$

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Recognized as indexed linear system

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

LP relaxation is convex hull of disjunction.

CP relaxation propagates bounds.

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Generate knapsack cuts from associated valid inequalities.

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Logical constraint on quantities

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

First branch on violated logical constraint on q_i variables

Product configuration

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

SIMPL model

Then branch on most violated t_i in-domain constraint.

Violated when domain of t_i is not a singleton, or two or more associated q_{ik}s are positive.

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Then branch on least violated q_i in-domain constraint.

Create three branches: q_i = nearest integer q_i', $q_i < q_i'$, $q_i > q_i'$

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Then branch on most violated logical constraint on t_i variables (omitted)

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Reduced-cost variable fixing for q_i's

Product configuration

Computational results

SIMPL matches hand-coded integrated method, which was orders of magnitude faster than CPLEX.

Again, CPLEX has become much faster, now somewhat faster than SIMPL.

MILP (CPLEX 11)		SIMPL	
Nodes	Sec.	Nodes	Sec.
1	0.07	56	0.49
1	0.10	32	0.25
31	0.68	186	1.67
1	0.02	28	0.24
1	0.12	32	0.33
1	0.07	9	0.09
10	0.18	35	0.30
1	0.10	32	0.25
1	0.05	28	0.22
1	0.12	14	0.13

Machine scheduling

- Assign jobs to machines, and schedule the machines assigned to each machine within time windows.
- The objective is to minimize **processing cost**.

Machine scheduling

Job Data

Job j	*Release time* r_j	*Dead-line* d_j	*Processing time* p_{Aj}	p_{Bj}
1	0	9	1	5
2	0	9	3	6
3	2	7	3	7
4	2	9	4	6
5	4	7	2	5

Machine A

Machine B

Example

Assign 5 jobs to 2 machines.

Schedule jobs assigned to each machine without overlap.

Machine scheduling

MILP continuous-time model

from Jain & Grossmann

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

$$r_j \le s_j \le d_j - \sum_i p_{ij} x_{ij}, \quad \text{all } j$$

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

$$y_{jj'} + y_{j'j} \le 1, \quad \text{all } j' > j$$

$$y_{jj'} + y_{j'j} + x_{ij} + x_{i'j'} \le 2, \quad \text{all } j' > j, i' \ne i$$

$$y_{jj'} + y_{j'j} \ge x_{ij} + x_{ij'} - 1, \quad \text{all } j' > j, i$$

$$s_{j'} \ge s_j + \sum_i p_{ij} x_{ij} - M(1 - y_{jj'}), \quad \text{all } j' \ne j$$

$$\sum_j p_{ij} x_{ij} \le \max_j \{d_j\} - \min_j \{r_j\}, \quad \text{all } i$$

$$x_{ij} \in \{0,1\}, \quad y_{jj'} \in \{0,1\}, \text{all } j' \ne j$$

Machine scheduling

MILP continuous-time model

from Jain & Grossmann

= 1 if job *j* assigned to machine *i*

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

$$r_j \le s_j \le d_j - \sum_i p_{ij} x_{ij}, \quad \text{all } j$$

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

= 1 of job j precedes j′

$$y_{jj'} + y_{j'j} \le 1, \quad \text{all } j' > j$$

$$y_{jj'} + y_{j'j} + x_{ij} + x_{i'j'} \le 2, \quad \text{all } j' > j, i' \ne i$$

$$y_{jj'} + y_{j'j} \ge x_{ij} + x_{ij'} - 1, \quad \text{all } j' > j, i$$

job start time

$$s_{j'} \ge s_j + \sum_i p_{ij} x_{ij} - M(1 - y_{jj'}), \quad \text{all } j' \ne j$$

$$\sum_j p_{ij} x_{ij} \le \max_j \{d_j\} - \min_j \{r_j\}, \quad \text{all } i$$

$$x_{ij} \in \{0,1\}, \quad y_{jj'} \in \{0,1\}, \text{all } j' \ne j$$

Machine scheduling

Integrated model

$$\min \sum_j c_{x_j j}$$

Start time of job *j*

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

Time windows

Jobs cannot overlap

$$\text{disjunctive}\left((s_j \mid x_j = i), (p_{ij} \mid x_j = i)\right), \text{ all } i$$

Machine assigned to job *j*

Original hand-coded method: Jain and Grossman 2001.

Machine scheduling

Integrated approach

- Assign the jobs in the **master problem**, to be solved by **MILP**.
- Schedule the jobs in the **subproblem**, to be solved by **CP**.

Original hand-coded method: Jain and Grossman 2001.

Machine scheduling

Integrated approach

- Assign the jobs in the **master problem**, to be solved by **MILP**.
- Schedule the jobs in the **subproblem**, to be solved by **CP**.

The subproblem decouples into a separate scheduling problem on each machine.

In this problem, the subproblem is a feasibility problem.

Original hand-coded method: Jain and Grossman 2001.

Machine scheduling

Integrated model

$$\min \sum_j c_{x_j j}$$

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

$$\text{disjunctive}\left((s_j \mid x_j = i),(p_{ij} \mid x_j = i)\right), \text{ all } i$$

Indexed linear metaconstraint

Machine scheduling

Integrated model

$$\min \sum_j c_{x_j j}$$

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

$$\text{disjunctive}\left((s_j \mid x_j = i),(p_{ij} \mid x_j = i)\right), \text{ all } i$$

Indexed linear metaconstraint

Disjunctive scheduling metaconstraint

Machine scheduling

Integrated model

Start time of job j

$$\min \ M$$

$$M \geq s_j + p_{x_j j}, \ \text{all } j$$

Time windows

$$r_j \leq s_j \leq d_j - p_{x_j j}, \ \text{all } j$$

Jobs cannot overlap

$$\text{disjunctive}\left((s_j \,|\, x_j = i),(p_{ij} \,|\, x_j = i)\right), \ \text{all } i$$

For a fixed assignment $\bar{x}$ the subproblem on each machine i is

$$\min \ M$$

$$M \geq s_j + p_{\bar{x}_j j}, \ \text{all } j \ \text{with } \bar{x}_j = i$$

$$r_j \leq s_j \leq d_j - p_{\bar{x}_j j}, \ \text{all } j \ \text{with } \bar{x}_j = i$$

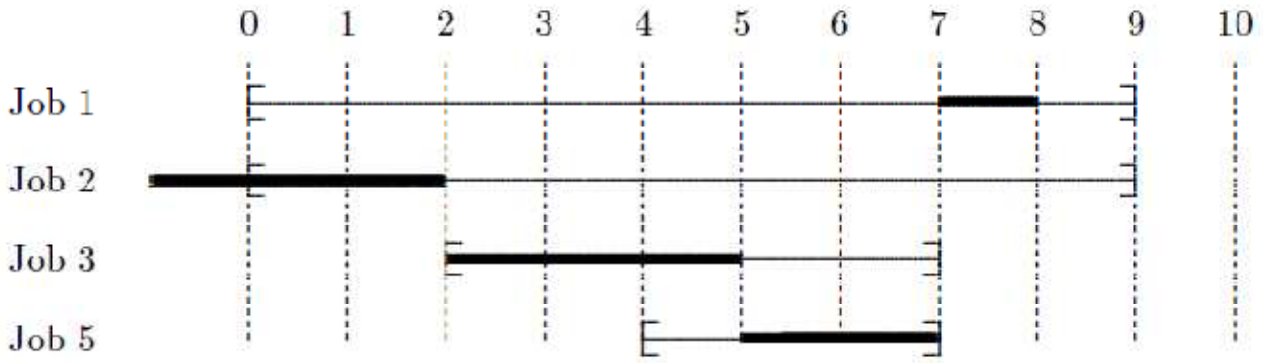
$$\text{disjunctive}\left((s_j \,|\, \bar{x}_j = i),(p_{ij} \,|\, \bar{x}_j = i)\right)$$

Machine scheduling

Logic-based Benders approach

Suppose we assign jobs 1,2,3,5 to machine A in iteration *k*.

We can prove that there is no feasible schedule.



Edge finding derives infeasibility by reasoning only with jobs 2,3,5.
So these jobs alone create infeasibility.

So we have a Benders cut $\neg(x_2 = x_3 = x_5 = A)$

Machine scheduling

Logic-based Benders approach

We want the master problem to be an MILP, which is good for assignment problems.

So we write the Benders cut $\neg(x_2 = x_3 = x_5 = A)$

Using 0-1 variables: $x_{A2} + x_{A3} + \boxed{x_{A5}} \leq 2$

= 1 if job 5 is assigned to machine A

Machine scheduling

The master problem is a **relaxation**, formulated as an MILP:

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

$$\sum_{j=1}^{5} p_{Aj} x_{Aj} \leq 9, \text{ etc.}$$

$$\sum_{j=1}^{5} p_{Bj} x_{Bj} \leq 9, \text{ etc.}$$

Constraints derived from time windows

$$x_{A2} + x_{A3} + x_{A5} \leq 2$$

Benders cut from machine A

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

Machine scheduling

The master problem is a **relaxation**, formulated as an MILP:

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

$$\sum_{j=1}^{5} p_{Aj} x_{Aj} \leq 9, \text{ etc.}$$

$$\sum_{j=1}^{5} p_{Bj} x_{Bj} \leq 9, \text{ etc.}$$

$$x_{A2} + x_{A3} + x_{A5} \leq 2$$

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

Constraints derived from time windows

Benders cut from machine A

Benders cuts have been developed for min **makespan** and min **tardiness** (subproblem is an optimization problem)

Also for **cumulative** scheduling.

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

SIMPL model

Machine assignment constraint

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

SIMPL model

MILP relaxation of the constraint (which is the constraint itself) goes into master problem

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

SIMPL model

Definition of x_{ij} variables for MILP master problem

Machine scheduling

SIMPL model

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

CP-based propagation

Machine scheduling

SIMPL model

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

Time window constraints recognized as indexed linear system

Machine scheduling

SIMPL model

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

MILP formulation goes into master problem

CP-based propagation

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

SIMPL model

Disjunctive scheduling constraint written as special case of cumulative scheduling constraint (resource consumption = 1, capacity = 1)

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

SIMPL model

The CP problem goes into the Benders subproblem.

A relaxation of the constraint goes into the master

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

SIMPL model

Generate logic-based Benders cuts when the subproblem is infeasible

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = [ benders ]
```

SIMPL model

Benders-based search, where problem restrictions are Benders subproblems and problem relaxations are master problems.

Machine scheduling

Computational results – Long processing times

Jobs	Machines	MILP (CPLEX 11) Nodes	MILP (CPLEX 11) Sec.	SIMPL Benders Iter.	SIMPL Benders Cuts	SIMPL Benders Sec.
3	2	1	0.00	2	1	0.00
7	3	1	0.00	13	16	0.12
12	3	3,351	6.6	26	35	0.73
15	5	2,779	8.8	20	29	0.83
20	5	33,321	882	13	82	5.4
22	5	352,309	10,563	69	98	9.6

SIMPL results are similar to original hand-coded results.

Machine scheduling

Computational results – Short processing times

Jobs	Machines	MILP (CPLEX 11) Nodes	MILP (CPLEX 11) Sec.	SIMPL Benders Iter.	SIMPL Benders Cuts	SIMPL Benders Sec.
3	2	1	0.01	1	0	0.00
7	3	1	0.02	1	0	0.00
12	3	499	0.98	1	0	0.01
15	5	529	2.6	2	1	0.06
20	5	250,047	369	6	5	0.28
22	5	> 27.5 mil.	> 48 hr	9	12	0.42
25	5	> 5.4 mil.	> 19 hr*	17	21	1.09

*out of memory

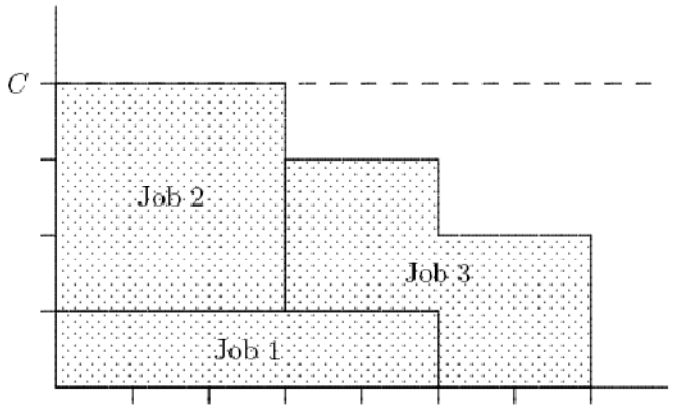
Machine scheduling

Benders cut for minimum **makespan**, with a **cumulative scheduling** subproblem

$$M \geq M_i^* - \left(\sum_{j \in J_i} p_{ij}(1 - x_{ij}) + \max_{j \in J_i}\{d_j\} - \min_{j \in J_i}\{d_j\} \right)$$

Jobs currently assigned to machine *i*

Minimum makespan on machine *i* for jobs currently assigned



Logic-based Benders Decomposition

- In general, Benders cuts are obtained by solving the **inference dual** of the subproblem.
  - The dual solution is a **proof** of optimality.
  - LP dual is a special case, where the proof is encoded by dual multipliers.

Logic-based Benders Decomposition

- In general, Benders cuts are obtained by solving the **inference dual** of the subproblem.
  - The dual solution is a **proof** of optimality.
  - LP dual is a special case, where the proof is encoded by dual multipliers.
- The Benders cut states conditions on the master problem variables under which the proof **remains valid**.
  - Classical Benders cut is a special case.

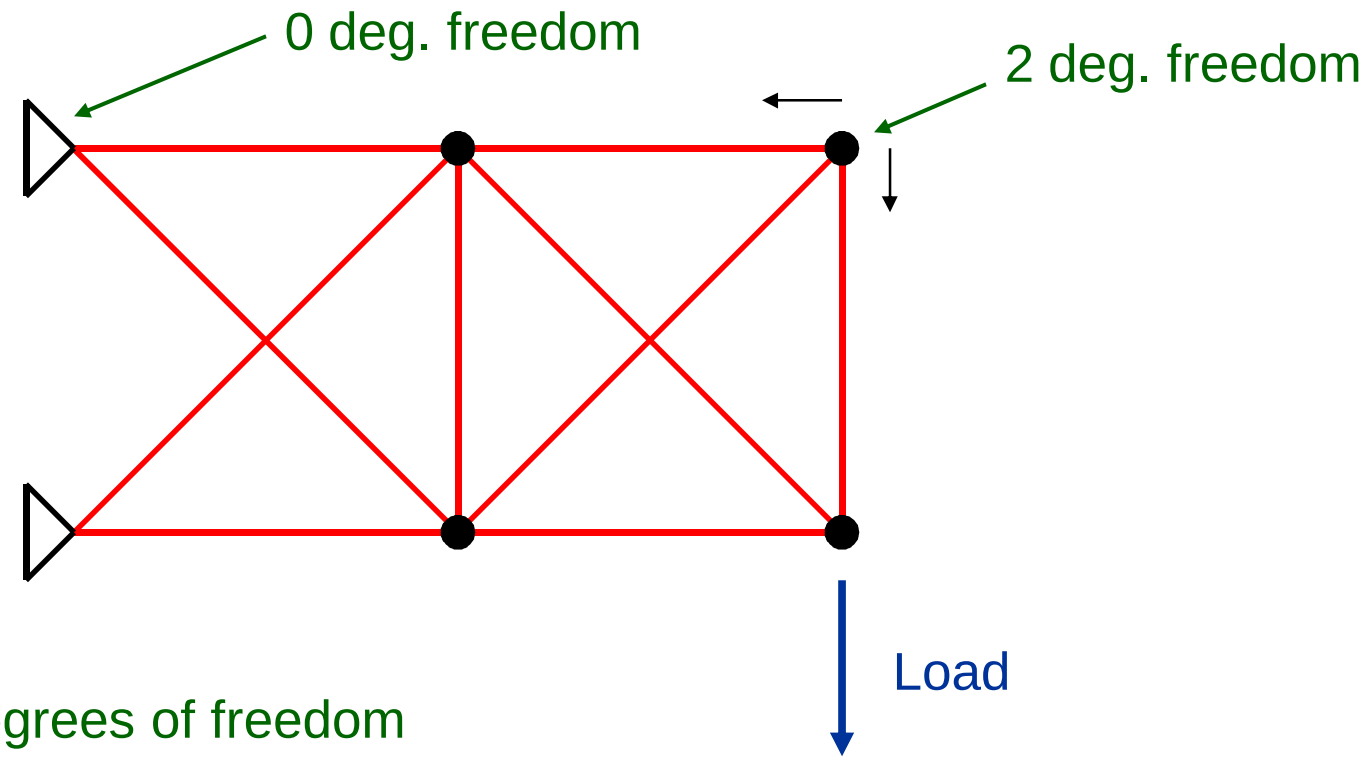
Logic-based Benders Decomposition

- In general, Benders cuts are obtained by solving the **inference dual** of the subproblem.
  - The dual solution is a **proof** of optimality.
  - LP dual is a special case, where the proof is encoded by dual multipliers.
- The Benders cut states conditions on the master problem variables under which the proof **remains valid**.
  - Classical Benders cut is a special case.
- LP, surrogate, Lagrangean, and superadditive duals are special cases of **inference duality** and **relaxation duality**.
  - Whence the prevalence of **relaxation** and **inference** dualities in problem solving.

Truss Structure Design

Select size of each bar (possibly zero) to support the load while minimizing weight.

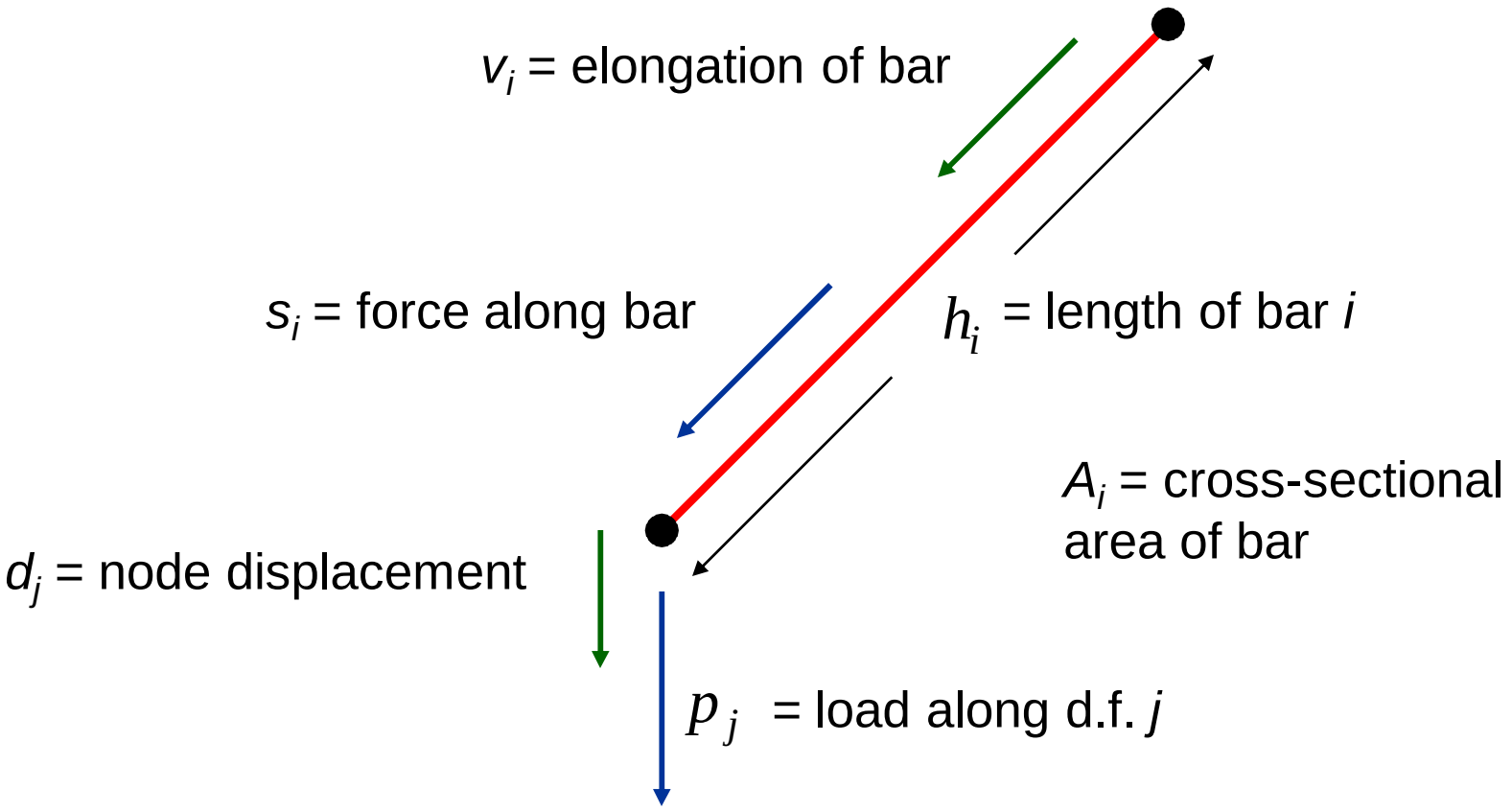
10-bar cantilever truss



Total 8 degrees of freedom

Truss Structure Design

Notation



Truss Structure Design

$$\min \quad \sum_i h_i A_i$$ } Minimize total weight

$$s.t. \quad \sum_i \cos\theta_{ij} s_i = p_j, \text{ all } j$$ } Equilibrium

$$\sum_j \cos\theta_{ij} d_j = v_i, \text{ all } i$$ } Compatibility

nonlinear →

$$\frac{E_i}{h_i} A_i v_i = s_i, \text{ all } i$$ } Hooke's law

$$v_i^L \le v_i \le v_i^U, \text{ all } i$$ } Elongation bounds

$$d_j^L \le d_j \le d_j^U, \text{ all } j$$ } Displacement bounds

$$\bigvee_k (A_i = A_{ik})$$ } Logical disjunction

Area must be one of several discrete values A_{ik}

Constraints can be imposed for multiple loading conditions

Truss Structure Design

Introducing new variables linearizes the problem but makes it much larger.

MILP model

$$\min \quad \sum_i h_i \sum_k A_{ik} y_{ik}$$

$$s.t. \quad \sum_i \cos\theta_{ij} s_i = p_j, \text{ all } j$$

$$\sum_j \cos\theta_{ij} d_j = \sum_k v_{ik}, \text{ all } i$$

$$\frac{E_i}{h_i} \sum_k A_{ik} v_{ik} = s_i, \text{ all } i$$

$$v_i^L \le v_i \le v_i^U, \text{ all } i$$

$$d_j^L \le d_j \le d_j^U, \text{ all } j$$

$$\sum_k y_{ik} = 1, \text{ all } i$$

0-1 variables indicating size of bar *i*

Elongation variable disaggregated by bar size

Hooke's law becomes linear

Truss Structure Design

Integrated approach

- Use the **original model** (don't introduce new variables)
- Branch by **splitting** the range of areas A_i
- Generate a **quasi-relaxation**, which is linear and much smaller than MILP model.

Original hand-coded method: Bollapragada, Ghattas, and JNH 2001.

Truss Structure Design

Theorem (JNH 2005)

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$
$$\alpha x^L \leq x^1 \leq \alpha x^U$$
$$(1-\alpha)x^L \leq x^2 \leq (1-\alpha)x^U$$
$$x = x^1 + x^2$$

Truss Structure Design

Theorem (JNH 2005)

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$
$$\alpha x^L \le x^1 \le \alpha x^U$$
$$(1-\alpha)x^L \le x^2 \le (1-\alpha)x^U$$
$$x = x^1 + x^2$$

Its optimal value is a lower bound on the optimal value of the original problem, if cost is a function of x alone.

Truss Structure Design

Theorem (JNH 2005)

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$
$$\alpha x^L \le x^1 \le \alpha x^U$$
$$(1-\alpha)x^L \le x^2 \le (1-\alpha)x^U$$
$$x = x^1 + x^2$$

$g(\alpha x, y) \le \alpha g(x,y)$ for all x, y and $\alpha \in [0,1]$

$g(0,y) = 0$ for all y

Truss Structure Design

Theorem (JNH 2005)

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$
$$\alpha x^L \leq x^1 \leq \alpha x^U$$
$$(1-\alpha)x^L \leq x^2 \leq (1-\alpha)x^U$$
$$x = x^1 + x^2$$

Bounds on y

Truss Structure Design

Theorem (JNH 2005)

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$

$$\alpha x^L \le x^1 \le \alpha x^U$$

$$(1-\alpha) x^L \le x^2 \le (1-\alpha) x^U$$

$$x = x^1 + x^2$$

Bounds on x

Truss Structure Design

Theorem (JNH 2005)

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$
$$\alpha x^L \le x^1 \le \alpha x^U$$
$$(1-\alpha)x^L \le x^2 \le (1-\alpha)x^U$$
$$x = x^1 + x^2$$

$\frac{E_i}{h_i} A_i v_i = s_i$ has the form $g(x,y) = 0$ with g semihomogenous in x because we can write it $\frac{E_i}{h_i} A_i v_i - s_i = 0$

with $x = (A_i, s_i)$, $y = v_i$.

Truss Structure Design

So we have a quasi-relaxation of the truss problem:

$$\min \quad \sum_i h_i [A_i^L y_i + A_i^U (1 - y_i)]$$

$$s.t. \quad \sum_i \cos\theta_{ij} s_i = p_j, \text{ all } j$$

$$\sum_j \cos\theta_{ij} d_j = v_{i0} + v_{i1}, \text{ all } i$$

$$\frac{E_i}{h_i}(A_i^L v_{i0} + A_i^U v_{i1}) = s_i, \text{ all } i$$

$$v_i^L y_i \le v_{i0} \le v_i^U y_i, \text{ all } i$$

$$v_i^L (1 - y_i) \le v_{i1} \le v_i^U (1 - y_i), \text{ all } i$$

$$d_j^L \le d_j \le d_j^U, \text{ all } j$$

$$0 \le y_i \le 1, \text{ all } i$$

Hooke's law is linearized

Elongation bounds split into 2 sets of bounds

Truss Structure Design

Logic cuts

v_{i0} and v_{i1} must have same sign in a feasible solution.

If not, we branch by adding logic cuts

$$v_{i0}, v_{i1} \leq 0, \qquad v_{i0}, v_{i1} \geq 0$$

Truss Structure Design

In general, we can have a **metaconstraint** to represent the semihomogeneous constraint $g(x,y) \leq 0$ and generate a quasi-relaxation.

Since a bilinear constraint $xy = \alpha$ is always semihomogeneous, we will use a **bilinear** metaconstraint with a quasi-relaxation option.

Truss Structure Design

SIMPL model

Recognized as linear systems

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Recognized as bilinear system

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Generate quasi-relaxation for semihomogenous function

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Branch first on violated logic cuts for quasi-relaxation

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,1] = p[j,1] forall j,1
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,1] = v[i,1] forall i,1
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,1] = s[i,1] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

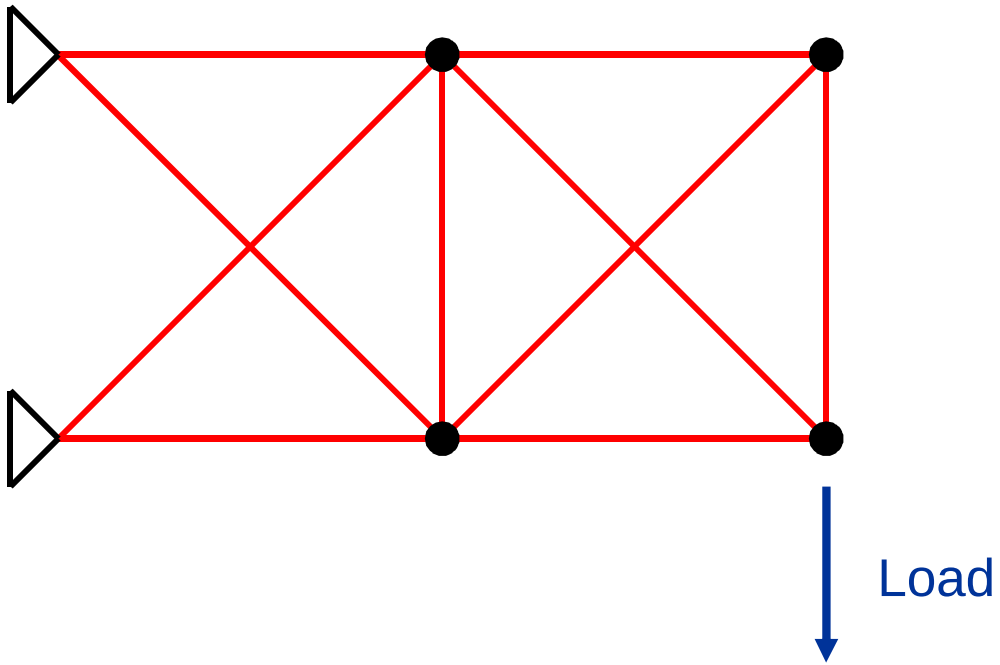
Then branch on A_i in-domain constraint.

Violated when A_i is not one of the discrete bar sizes.

Take upper branch first.

Truss Structure Design

10-bar cantilever truss



Truss Structure Design

Hand-coded integrated method

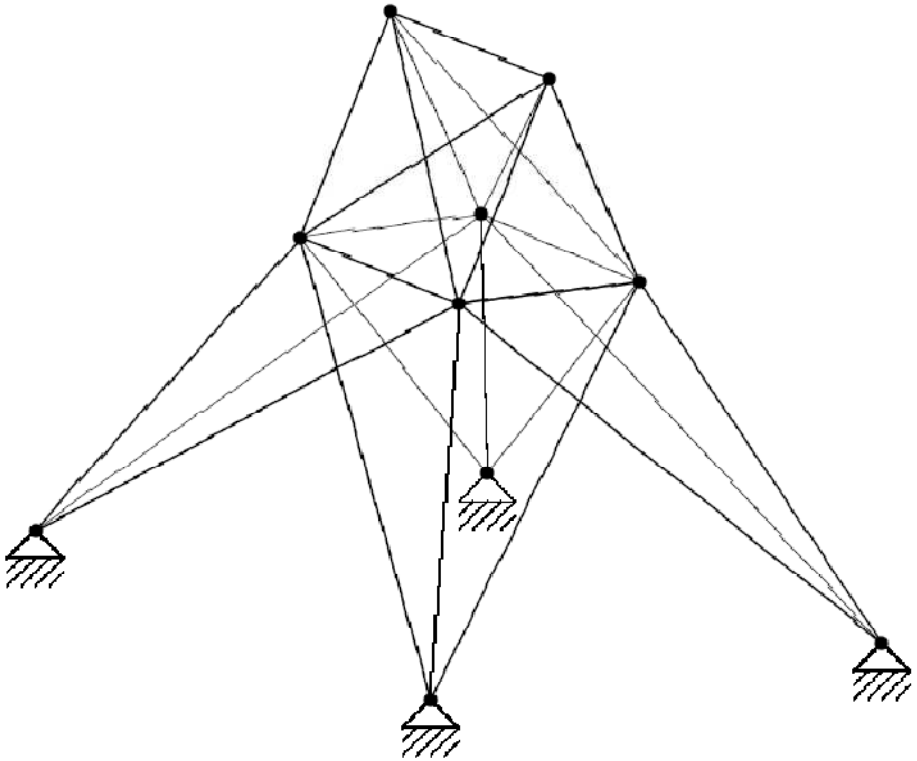
Computational results (seconds)

No. bars	Loads	BARON	CPLEX 11	Hand coded	SIMPL
10	1	5.3	0.40	0.03	0.08
10	1	3.8	0.26	0.02	0.07
10	1	8.1	0.83	0.16	0.49
10	1	8.8	1.2	0.22	0.63
10	2	24	4.9	0.64	1.84
10	2*	327	146	145	65
10	2*	2067	1087	600	651

*plus displacement bounds

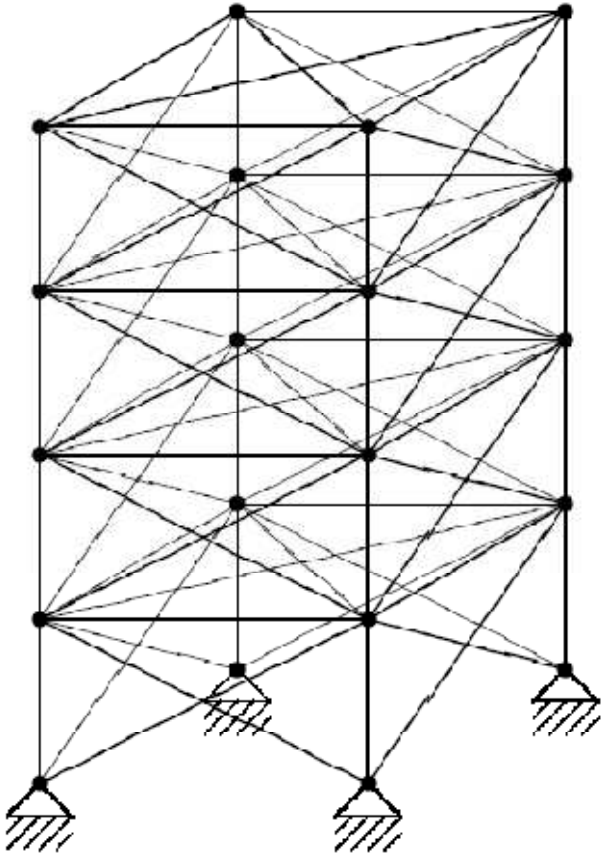
Truss Structure Design

25-bar problem



Truss Structure Design

72-bar problem



Truss Structure Design

Hand-coded integrated method

Computational results (seconds)

No. bars	Loads	BARON	CPLEX 11	Hand coded	SIMPL
25	2	3,302	44	44	20
72	2	3,376	208	33	28
90	2	21,011	570	131	92
108	2	> 24 hr*	3208	1907	1720
200	2	> 24 hr*	> 24 hr*	> 24 hr**	> 24 hr***

* no feasible solution found

** best feasible solution has cost 32,748

*** best feasible solution has cost 32,700

Current Version of SIMPL

- To download:
  - Click the link to SIMPL on John Hooker's website.
  - See readme file for complete instructions.
  - Download executable and associated files
- Operational on GNU/Linux only
- Requires subsidiary solvers
  - CPLEX (version 9, 10, or 11)
  - Eclipse (any version 5.8.80 or later), free download
- Download problem instances
  - Including all reported in this talk.