Experimenting with Near-Optimal Formulations for Discrete Optimization Problems

Robert Fourer

Industrial Engineering & Management Sciences
Northwestern University, Evanston, IL, USA

AMPL Optimization LLC

4er@northwestern.edu — 4er@ampl.com

ALIO-INFORMS Joint International Meeting

Buenos Aires, June 6-9, 2010

Session WC08: OR in Practice III – Modeling

Outline

Examples

- 1. Paint chip ordering
- 2. Roll ordering
  - a. Huge number of patterns
  - b. Side constraints
- 3. Balanced assignment
- 4. Work scheduling

Modeling language observations . . .

- Convenience
- Generality of expressions & constraints
- Variety of solver support

Example 1: Paint Chip Ordering

Produce paint chips from rolls of material

- Several “groups” (types) of chips
- Various numbers of “colors” per group
- Numerous “patterns” of groups on rolls

Costs proportional to numbers of

- Patterns cut
- Pattern changes
- Width changes

. . . thanks to Collette Coullard for this application

Example 1

Paint Chip Ordering

Model (variables & objective)

```
var Cut {1..nPats} > = 0, integer;      # number of each pattern cut
var PatternChange {1..nPats} binary;    # 1 iff a pattern is used
var WebChange {WIDTHS} binary;          # 1 iff a width is used

minimize Total_Cost:
      sum {j in 1..nPats} cut_cost[j] * Cut[j] +
   pattern_changeover_factor *
      sum {j in 1..nPats} change_cost[j] * PatternChange[j] +
   web_change_factor *
      sum {w in WIDTHS} (coat_change_cost + slit_change_cost) WebChange[w];
```

Example 1

Paint Chip Ordering

Model (constraints)

```
subject to SatisfyDemand {g in GROUPS}:
   sum {j in 1..nPats} number_of[g,j] * Cut[j] >= ncolors[g];

subject to DefinePatternChange {j in 1..nPats}:
   Cut[j] <= maxuse[j] * PatternChange[j];

subject to DefineWebChange {j in 1..nPats}:
   PatternChange[j] <= WebChange[width[j]];
```

```
param maxuse {j in 1..nPats} :=
   max {g in GROUPS: number_of[g,j] > 0} ncolors[g] / number_of[g,j];
      # upper limit on Cut[j]
```

. . . very long solve times

Example 1

Paint Chip Ordering

Model (restricted)

```
subject to DefinePatternChange {j in 1..nPats}:
   Cut[j] <= maxuse[j] * PatternChange[j];

subject to MinPatternUse {j in 1..nPats}:
   Cut[j] >= ceil(minuse[j]) * PatternChange[j];
```

```
param minuse {j in 1..nPats} :=
   min {g in GROUPS: number_of[g,j] > 0} ncolors[g] / number_of[g,j];
      # if you use a pattern at all,
      # use it to cut all colors of at least one group
```

. . . not necessarily optimal, but . . .

Example 1

Paint Chip Ordering

Sample data

```
param: GROUPS: ncolors slitwidth cutoff  paint    finish     substrate :=

       grp1    8       3.8125    1.75    latex    flat       P40
       grp2    3       3.9375    1.75    latex    flat       P40
       grp3    32      1.6875    1.00    latex    flat       P40
       grp4    4       1.8125    1.00    latex    flat       P40
       grp5    3       1.75      1.00    latex    flat       P40
       grp6    2       1.75      1.00    latex    semi_gloss P40
       grp7    3       1.875     1.00    latex    flat       P40
       grp8    1       1.875     1.00    latex    gloss      P40  ;

param orderqty := 588500;

param spoilage_factor := .15;
```

Example 1

Paint Chip Ordering

Without restriction

- 1812 rows, 1807 columns, 5976 nonzeros
- 7,115,951 simplex iterations
- 221,368 branch-and-bound nodes
- 14,620.4 seconds

With restriction

- 2402 rows, 1656 columns, 7091 nonzeros
- 230,667 simplex iterations
- 9,892 branch-and-bound nodes
- 501.55 seconds

Objective value

- Same in both cases

Example 1

Paint Chip Orders *(today)*

Without restriction

- 1724 rows, 1719 columns, 5800 nonzeros
- 49,831 simplex iterations
- 3,157 branch-and-bound nodes
- 4.867 seconds

With restriction

- 2344 rows, 1598 columns, 6982 nonzeros
- 21,598 simplex iterations
- 568 branch-and-bound nodes
- 2.872 seconds

(Gurobi 1.1.3, 8 processors)

Example 1

Paint Chip Orders *(today, harder case)*

Without restriction

- 4019 rows, 4009 columns, 15198 nonzeros
- 60,122 simplex iterations
- 1,955 branch-and-bound nodes
- 20.626 seconds

With restriction

- 5667 rows, 4394 columns, 18464 nonzeros
- 14,468 simplex iterations
- 150 branch-and-bound nodes
- 5.464 seconds

(Gurobi 1.1.3, 8 processors)

Example 2a: Roll Ordering

Cut large "raw" rolls into smaller ones

- All raw rolls the same width
- Various smaller widths ordered
- Varying numbers of widths ordered

Minimize total raw rolls cut

- By generating patterns during optimization
- By enumerating patterns in advance

Example 2a

Roll Ordering

Cutting model

```
set WIDTHS;                                  # set of widths to be cut
param orders {WIDTHS} > 0;                   # number of each width to be cut

param nPAT integer >= 0;                     # number of patterns
param nbr {WIDTHS,1..nPAT} integer >= 0;     # rolls of width i in pattern j

var Cut {1..nPAT} integer >= 0;              # rolls cut using each pattern

minimize Number:

   sum {j in 1..nPAT} Cut[j];                # total raw rolls cut

subject to Fill {i in WIDTHS}:

   sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];

                                             # for each width,
                                             # rolls cut meet orders
```

Example 2a
Roll Ordering

Pattern generation model

```
param roll_width > 0;
param price {WIDTHS} default 0.0;

var Use {WIDTHS} integer >= 0;

minimize Reduced_Cost:

   1 - sum {i in WIDTHS} price[i] * Use[i];

subj to Width_Limit:

   sum {i in WIDTHS} i * Use[i] <= roll_width;
```

Example 2a

Roll Ordering

Pattern generation script

```
repeat {

   solve Cutting_Opt;

   let {i in WIDTHS} price[i] := Fill[i].dual;

   solve Pattern_Gen;

   if Reduced_Cost < -0.00001 then {
      let nPAT := nPAT + 1;
      let {i in WIDTHS} nbr[i,nPAT] := Use[i];
      }

   else break;

   };
```

Example 2a

Roll Ordering

Pattern enumeration script

```
repeat {
   if curr_sum + curr_width <= roll_width then {
      let pattern[curr_width] := floor((roll_width-curr_sum)/curr_width);
      let curr_sum := curr_sum + pattern[curr_width] * curr_width;
      }
   if curr_width != last(WIDTHS) then
      let curr_width := next(curr_width,WIDTHS);
   else {
      let nPAT := nPAT + 1;
      let {w in WIDTHS} nbr[w,nPAT] := pattern[w];
      let curr_sum := curr_sum - pattern[last(WIDTHS)] * last(WIDTHS);
      let pattern[last(WIDTHS)] := 0;
      let curr_width := min {w in WIDTHS: pattern[w] > 0} w;
      if curr_width < Infinity then {
         let curr_sum := curr_sum - curr_width;
         let pattern[curr_width] := pattern[curr_width] - 1;
         let curr_width := next(curr_width,WIDTHS);
         }
      else break;
      }
   }
```

Example 2a

Roll Ordering

Sample data

```
param roll_width := 172 ;

param: WIDTHS: orders :=

       25.000     5
       24.750    73
       18.000    14
       17.500     4
       15.500    23
       15.375     5
       13.875    29
       12.500    87
       12.250     9
       12.000    31
       10.250     6
       10.125    14
       10.000    43
        8.750    15
        8.500    21
        7.750     5 ;
```

. . . Robert W. Haessler, "Selection and Design of Heuristic Procedures for Solving Roll Trim Problems" Management Science 34 (1988) 1460–1471, Table 2

Example 2a

Roll Ordering

Patterns generated during optimization (Gilmore-Gomory procedure)

- 32.80 rolls in continuous relaxation
- 40 rolls rounded up to integer
- 34 rolls solving IP using generated patterns

All patterns enumerated in advance

- 27,338,021 non-dominated patterns — too big

Every 100th pattern saved

- 273,380 patterns
- 33 rolls solving IP using enumerated patterns
- 50 seconds: b&b heuristic solves at root (no cuts)

. . . takes much longer to generate than solve

Example 2b: Roll Ordering with Side Constraints

Additional restrictions on cutting solution

- No overage (fill all orders exactly) *and also* . . .
- At most 2% waste per pattern
- At most 8 widths per pattern
- At most 8 widths and 10% waste per pattern

Example 2b

Roll Ordering with Side Constraints

Sample data

```
param roll_width := 349 ;

param: WIDTHS: orders :=

       28.75     7
       33.75    23
       34.75    23
       37.75    31
       38.75    10
       39.75    39
       40.75    58
       41.75    47
       42.25    19
       44.75    13
       45.75    26 ;
```

. . . Zeger Degraeve and Linus Schrage,
"Optimal Integer Solutions to Industrial Cutting Stock Problems"
INFORMS Journal on Computing 11 (1999) 406–419, Table VIII

Example 2b

Roll Ordering with Side Constraints

Patterns generated during opt (without side constr)

- 33.78 rolls in continuous relaxation
- 41 rolls rounded up to integer
- 35 rolls solving IP using generated patterns

All patterns enumerated in advance

- 54,508 non-dominated patterns
- 34 rolls solving IP using enumerated patterns
- 200 branch-and-bound nodes

No overage: change >= to =

- 34 rolls solving IP using enumerated patterns
- 0 branch-and-bound nodes

. . . ***all subsequent tests include this condition***

Example 2b

Roll Ordering with Side Constraints

At most 2% waste in any pattern

- 16,362 non-dominated patterns
- branch-and-bound ran out of memory
- *no feasible solutions found!*

Minimize total cut rolls instead: keep >=

```
minimize Number:
  sum {j in 1..nPAT} Cut[j];

minimize Over:
  sum {j in 1..nPAT} (sum {i in WIDTHS} nbr[i,j]) * Cut[j];
```

- 296 cut rolls (= 296 orders) solving IP
- 34 raw rolls in that solution
- 1279 branch-and-bound nodes

. . . overage is feasible, just not optimal

Example 2b

Roll Ordering with Side Constraints

At most 8 widths in any pattern

- 13,877 non-dominated patterns having at most 8 widths
- 312 cut rolls (> 296 orders) solving IP
- 39 raw rolls in that solution
- *all feasible solutions have overage!*

Allow more patterns

- generate 9-width patterns with one width removed
- 200,186 patterns, some dominated
- 296 cut rolls (= 296 orders) solving IP
- 37 raw rolls in that solution
- 113 branch-and-bound nodes

Example 2b

Roll Ordering with Side Constraints

At most 8 widths and 10% waste in any pattern

- 21,098 patterns, some dominated
- 296 cut rolls (= 296 orders) solving IP
- 37 raw rolls in that solution
- 142 branch-and-bound nodes

Example 3: Balanced Assignment

Partition people into groups

- diversity measured by several characteristics
- each characteristic has several values

Make groups as diverse as possible

- count "overlaps" for each person in their assigned group
  - for each other in group, count # of matching characteristics
  - sum over all others in group
- minimize sum of overlaps

Test data

- 26 people
- 4 characteristics (4, 4, 4, 2 values)
- 5 groups

Example 3

Balanced Assignment

History

- One attempt at modeling a real application
- Class example of where branch-and-bound fails
    - ∗ steadily growing tree
    - ∗ terrible initial lower bound
    - ∗ gap scarcely grows

```
CPLEX 11.2.0:

Reduced MIP has 161 rows, 265 columns, and 3725 nonzeros.
Reduced MIP has 130 binaries, 0 generals, 0 SOSs, and 0 indicators.

Clique table members: 26.

MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.

Parallel mode: none, using 1 thread.

Root relaxation solution time =   -0.00 sec.
```

Example 3

Balanced Assignment

Active start . . .

	Node	Nodes Left	Objective	IInf	Best Integer	Cuts/ Best Node	ItCnt	Gap
	0	0	0.0000	61		0.0000	99	
*	0+	0			232.0000	0.0000	99	100.00%
	0	0	0.0000	60	232.0000	Cuts: 55	174	100.00%
	0	0	0.0000	66	232.0000	Flowcuts: 17	250	100.00%
	0	0	0.0000	58	232.0000	Flowcuts: 9	300	100.00%
	0	0	0.0000	57	232.0000	Flowcuts: 13	326	100.00%
*	0+	0			230.0000	0.0000	326	100.00%
*	0+	0			216.0000	0.0000	326	100.00%
	0	2	0.0000	57	216.0000	0.0000	326	100.00%
*	440+	403			214.0000	0.0000	7938	100.00%
*	552+	339			212.0000	0.0000	10797	100.00%
	1000	556	69.9315	50	212.0000	0.0000	16491	100.00%
	2000	1332	42.8547	47	212.0000	0.0000	25669	100.00%
	3000	2276	81.6541	49	212.0000	5.0928	37332	97.60%
	4000	3214	77.9166	49	212.0000	5.1140	47933	97.59%
	5000	4160	71.0567	52	212.0000	6.4918	57582	96.94%
	6000	5089	97.3040	47	212.0000	7.8042	66662	96.32%
	7000	6021	158.4869	37	212.0000	9.3981	75348	95.57%
	8000	6942	157.5392	36	212.0000	11.2257	84237	94.70%

....

Example 3

Balanced Assignment

. . . bogs down completely

```
        Nodes                                         Cuts/
   Node    Left  Objective  IInf  Best Integer    Best Node     ItCnt     Gap
 .................
 6244000 5769420    91.8882    46      212.0000      55.4261  37227229   73.86%
 6245000 5770348   123.4752    34      212.0000      55.4272  37233744   73.86%
 6246000 5771270    63.5603    48      212.0000      55.4289  37239584   73.85%
 6247000 5772192   106.5663    43      212.0000      55.4294  37245120   73.85%
 6248000 5773112    64.0217    47      212.0000      55.4308  37251128   73.85%
 6249000 5774034   181.2576    31      212.0000      55.4310  37257940   73.85%
 6250000 5774954   119.4546    35      212.0000      55.4320  37263877   73.85%
Elapsed time = 9116.25 sec. (tree size = 1616.65 MB)
Nodefile size = 1488.81 MB (685.88 MB after compression)
 6251000 5775885   182.0327    29      212.0000      55.4328  37270210   73.85%
 6252000 5776807   140.1960    39      212.0000      55.4330  37275647   73.85%
 6253000 5777720    91.9423    43      212.0000      55.4346  37281516   73.85%
 6254000 5778648   127.8185    35      212.0000      55.4355  37286884   73.85%
8 flow-cover cuts
2 Gomory cuts
1 zero-half cut
9 mixed-integer rounding cuts
CPLEX 11.2.0: ran out of memory.
```

Example 3

Balanced Assignment

Definition of overlap for person i

```
minimize TotalOverlap:
   sum {i in PEOPLE} Overlap[i];

subj to OverlapDefn {i in PEOPLE, j in 1..numberGrps}:
   Overlap[i] >=
      sum {i2 in PEOPLE diff {i}: title[i2] = title[i]} Assign[i2,j] +
      sum {i2 in PEOPLE diff {i}: loc[i2] = loc[i]} Assign[i2,j] +
      sum {i2 in PEOPLE diff {i}: dept[i2] = dept[i]} Assign[i2,j] +
      sum {i2 in PEOPLE diff {i}: sex[i2] = sex[i]} Assign[i2,j]
         - maxOverlap[i] * (1 - Assign[i,j]);
```

- `maxOverlap[i]` must be ≥ greatest overlap possible
- Smaller values give stronger b&b lower bounds
  - ∗ theoretically correct: `4 * (maxInGrp-1)` → `0.0`
  - ∗ empirically justified: `1 * (maxInGrp-1)` → `156.8`

Example 3

Balanced Assignment

Symmetry constraint 1

```
subject to BreakSymm1
   {i in FIRST_PEOPLE, j in ord(i,FIRST_PEOPLE)+1..numberGrps}:
      Assign[i,j] = 0;
```

- choose the first `(numberGrps-1)` people in some way
- assign the *i*th person to one of the first *i* groups

Example 3

Balanced Assignment

Symmetry constraint 2

```
set TYPES = setof {i in PEOPLE} (title[i],loc[i],dept[i],sex[i]);
set TYPEpeople {(t1,t2,t3,t4) in TYPES} =
   {i in PEOPLE: title[i]=t1 and loc[i]=t2 and
                 dept[i]=t3 and sex[i]=t4} ordered by PEOPLE;

subject to BreakSymm2 {(t1,t2,t3,t4) in TYPES,
   pnum in 1..card(TYPEpeople[t1,t2,t3,t4])-1, j in 1..numberGrps, k in 1..j-1}:
      Assign[member(pnum+1,TYPEpeople[t1,t2,t3,t4]),k]
         <= 1 - Assign[member(pnum,TYPEpeople[t1,t2,t3,t4]),j];
```

- identify "types" of people
  who are identical in all four characteristics
- order the people of each type, and order the groups
- with each type, assign
  higher-numbered people to higher-numbered groups

Example 3

Balanced Assignment

Symmetry strategies

- `BreakSymm1` increases the b&b lower bound a bit
- `BreakSymm2` does not increase the lower bound
- CPLEX's `symmetry` directive is more effective
  - set `symmetry=5` for greatest symmetry-breaking effort

Example 3

Balanced Assignment

Group size limits

```
subj to GroupSize {j in 1..numberGrps}:
   minInGrp <= sum {i in PEOPLE} Assign[i,j] <= maxInGrp;
```

- `minInGrp` must be smaller than group size average
- `maxInGrp` must be larger than group size average

- Tighter limits give stronger b&b lower bounds
  - `floor(card(PEOPLE)/numberGrps) - 1`
    `ceil (card(PEOPLE)/numberGrps) + 1` → 156.8
  - `floor(card(PEOPLE)/numberGrps)`
    `ceil (card(PEOPLE)/numberGrps)` → 177.6

Example 3

Balanced Assignment

Group sizes

```
param minInGrp := floor (card(PEOPLE)/numberGrps);
param nMinInGrp := numberGrps - card{PEOPLE} mod numberGrps;

subj to GroupSizeMin {j in 1..nMinInGrp}:
   sum {i in PEOPLE} Assign[i,j] = minInGrp;

subj to GroupSizeMax {j in nMinInGrp+1..numberGrps}:
   sum {i in PEOPLE} Assign[i,j] = minInGrp + 1;
```

- Compute exact sizes of all groups
- b&b lower bound increases from 177.6 to 183.36
  - 16.2% to 13.5% below best known solution of 212

Example 3

Balanced Assignment

Incorporating enhancements . . .

```
ampl: model gs1f.mod;
ampl: data gs1b.dat;
ampl: option solver cplex;
ampl: option cplex_options 'symmetry 5 mipdisplay 2 mipinterval 1000';
ampl: solve;

MIP Presolve eliminated 54 rows and 0 columns.
MIP Presolve modified 2636 coefficients.
Reduced MIP has 197 rows, 156 columns, and 2585 nonzeros.
Reduced MIP has 130 binaries, 0 generals, 0 SOSs, and 0 indicators.
Clique table members: 62.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: none, using 1 thread.
Root relaxation solution time =    0.03 sec.

        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap

*     0+    0                          252.0000                      0     ---
      0     0      183.3626   134      252.0000      183.3626      262   27.24%
.......
```

Example 3

Balanced Assignment

Much more promising start . . .

	Node	Nodes Left	Objective	IInf	Best Integer	Cuts/ Best Node	ItCnt	Gap
	0	0	189.1865	100	252.0000	Cuts: 49	445	24.93%
	0	0	189.7246	96	252.0000	Cuts: 12	558	24.71%
*	0+	0			240.0000	189.7246	558	20.95%
	0	0	189.7964	96	240.0000	ZeroHalf: 5	664	20.92%
	0	0	189.8864	97	240.0000	ZeroHalf: 8	782	20.88%
	0	0	189.9590	96	240.0000	ZeroHalf: 6	1002	20.85%
	0	0	189.9768	100	240.0000	ZeroHalf: 7	1166	20.84%
	0	0	189.9769	99	240.0000	ZeroHalf: 4	1184	20.84%
*	0+	0			220.0000	189.9769	1203	13.65%
*	0+	0			216.0000	189.9769	1203	12.05%
	0	2	192.8299	78	216.0000	192.8299	1203	10.73%
*	100+	80			212.0000	193.0563	6092	8.94%
	1000	479	200.3732	83	212.0000	195.6130	36233	7.73%
	2000	1242	205.1626	64	212.0000	195.9832	65307	7.56%
	3000	2103	205.8520	59	212.0000	196.4174	93546	7.35%
	4000	2946	205.5224	57	212.0000	196.8495	120479	7.15%
	5000	3790	201.5651	53	212.0000	197.1664	145209	7.00%
	6000	4624	210.5546	34	212.0000	197.4648	169658	6.86%
	7000	5468	201.2841	60	212.0000	197.6005	195286	6.79%

....

Example 3

Balanced Assignment

. . . leads to successful conclusion

```
           Nodes                                       Cuts/
   Node     Left     Objective  IInf  Best Integer   Best Node      ItCnt     Gap
 30287000   8802        cutoff              212.0000    211.0000 416705257   0.47%
 30288000   7927        cutoff              212.0000    211.0000 416709767   0.47%
 30289000   7021    infeasible              212.0000    211.0000 416714199   0.47%
 30290000   6101    infeasible              212.0000    211.0000 416718973   0.47%
Elapsed time = 46415.00 sec. (tree size = 12.94 MB)
 30291000   5249        cutoff              212.0000    211.0000 416724639   0.47%
 30292000   4407    infeasible              212.0000    211.0000 416730198   0.47%
 30293000   3519    infeasible              212.0000    211.0000 416735118   0.47%
 30294000   2636        cutoff              212.0000    211.0000 416740781   0.47%
 30295000   1758    infeasible              212.0000    211.0000 416746255   0.47%
 30296000    863    infeasible              212.0000    211.0000 416748900   0.47%
3 cover cuts
8 implied bound cuts
23 mixed-integer rounding cuts
35 zero-half cuts
12 Gomory fractional cuts
CPLEX 11.2.0: optimal integer solution; objective 212
416751729 MIP simplex iterations
30296965 branch-and-bound nodes
```

Example 4: Work Scheduling

Cover demands for workers

- Each "shift" requires a certain number of employees
- Each employee works a certain "schedule" of shifts
- *Each schedule that is worked by anyone must be worked by a fixed minimum number*

Minimize total workers needed

- Which schedules are used?
- How many work each of schedule?

Example 4

Work Scheduling

Model using zero-one variables

```
var Work {SCHEDS} >= 0 integer;
var Use  {SCHEDS} >= 0 binary;

minimize Total_Cost:

   sum {j in SCHEDS} Work[j];

subject to Shift_Needs {i in SHIFTS}:

   sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j] >= required[i];

subject to Least_Use1 {j in SCHEDS}:

   Work[j] >= least_assign * Use[j];

subject to Least_Use2 {j in SCHEDS}:

   Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

Example 4

Work Scheduling

Test data

```
set SHIFTS := Mon1 Tue1 Wed1 Thu1 Fri1 Sat1
              Mon2 Tue2 Wed2 Thu2 Fri2 Sat2
              Mon3 Tue3 Wed3 Thu3 Fri3 ;

param Nsched := 126 ;

set SHIFT_LIST[1] := Mon1 Tue1 Wed1 Thu1 Fri1 ;
set SHIFT_LIST[2] := Mon1 Tue1 Wed1 Thu1 Fri2 ;
set SHIFT_LIST[3] := Mon1 Tue1 Wed1 Thu1 Fri3 ;
set SHIFT_LIST[4] := Mon1 Tue1 Wed1 Thu1 Sat1 ;
set SHIFT_LIST[5] := Mon1 Tue1 Wed1 Thu1 Sat2 ;
set SHIFT_LIST[6] := Mon1 Tue1 Wed1 Thu2 Fri2 ;
set SHIFT_LIST[7] := Mon1 Tue1 Wed1 Thu2 Fri3 ;

.......

param required :=  Mon1 100  Mon2 78  Mon3 52
                    Tue1 100  Tue2 78  Tue3 52
                    Wed1 100  Wed2 78  Wed3 52
```

Example 4

Work Scheduling

Direct approach

- Apply branch-and-bound to whole problem
- Branch "up" first

Indirect approach

- Step 1: Relax integrality of `Work` variables
  Solve for zero-one `Use` variables
- Step 2: Fix `Use` variables
  Solve for integer `Work` variables

. . . not necessarily optimal, but . . .

Example 4

Work Scheduling

Typical run of indirect approach

```
ampl: model sched1.mod; data sched.dat;

ampl: let least_assign := 16;

ampl: option solver cplex;
ampl: option cplex_options 'branch 1';

ampl: let {j in SCHEDS} Work[j].relax := 1;

ampl: solve;

CPLEX 11.2.0: optimal integer solution; objective 265.6
870496 MIP simplex iterations
55911 branch-and-bound nodes

ampl: fix {j in SCHEDS} Use[j];
ampl: let {j in SCHEDS} Work[j].relax := 0;

ampl: solve;

CPLEX 11.2.0: optimal integer solution; objective 266
24 MIP simplex iterations
4 branch-and-bound nodes
```

Example 4

Work Scheduling

Direct approach (CPLEX 11.2, 1 processor)

least_assign	nodes	iterations	seconds	
16	113214	1097779	122	
17				*gave up*
18	6063049	139707354	8568	
19				*gave up*
20	50823	839531	48	
23	1316985	25751165	1428	
24	23386	315922	21	

- gave up because tree still growing after 5+ hours

Example 4

Work Scheduling

Indirect approach (CPLEX 11.2, 1 processor)

least_assign	nodes	iterations	seconds
16	55911	870496	73
17	1082098	18664635	1364
18	969105	17605901	1276
19	2759853	51802234	3699
20	84325	1530127	89
23	92779	1415715	90
24	72215	1062010	66

- step 2 always trivially easy
- step 2 objective always rounds up step 1 objective

*. . . **hence optimal***

Example 4

Work Scheduling

Direct approach (Gurobi 1.1.3, 8 processors)

least_assign	nodes	iterations	seconds	
16	1345687	10113022	115	
17				*gave up*
18	15870199	125799234	1566	
19	206355833	1619459036	11747	
20	232603	1105751	19	
21	273837	1262181	21	
22	96277	533727	10	
23	129899	632361	10	
24	99489	483954	8	

- gave up because tree still growing after 8+ hours

Example 4

Work Scheduling

Indirect approach (Gurobi 1.1.3, 8 processors)

least_assign	nodes	iterations	seconds	
16	71924	285172	5	
17	1556653	7898786	120	
18	5538287	33278060	305	
19	6866450	47120495	388	
20	117970	440182	9	*integer*
21	76873	299338	7	*integer*
22	61727	259012	5	*integer*
23	111721	392251	8	*integer*
24	82152	292187	6	*integer*

- step 1 sometimes gives an integer solution
- step 2 always trivially easy
- step 2 objective always rounds up step 1 objective

. . . hence optimal

Example 4

Work Scheduling: More on Case "17"

CPLEX 12.1

- Direct: 465,596,558 nodes, 112013 seconds
- Indirect: 6,886,122 nodes, 617 seconds

Gurobi 3.0 beta

- Direct: 1,330,555,419 nodes, 69945 seconds
- Indirect: 6,354,683 nodes, 299 seconds

Observations

- step 1 gives fractional solution
- step 2 trivially easy and rounds up step 1 objective

. . . ***hence optimal***

Observations

Convenience

- quick formulation changes
- simple scripts

Generality of expressions & constraints

- more than arithmetic expressions
- more than “range” constraints

Variety of solver support

- constraint programming
- nondifferentiable optimization
- global optimization

. . . ***diversity of interfaces***

Generality

Scheduling model using "implies" operator

```
var Work {SCHEDS} >= 0 integer;
var Use  {SCHEDS} >= 0 binary;

minimize Total_Cost:

   sum {j in SCHEDS} Work[j];

subject to Shift_Needs {i in SHIFTS}:

   sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j] >= required[i];

subject to Least_Use1_logical {j in SCHEDS}:

   Use[j] = 1 ==> Work[j] >= least_assign;

subject to Least_Use2_logical {j in SCHEDS}:

   Use[j] = 0 ==> Work[j] = 0;
```

. . . don't need upper bounds on integer variables

Observations

Generality

Scheduling model using variable ranges

```
var Work {j in SCHEDS} integer, in {0} union
   interval [least_assign, (max {i in SHIFT_LIST[j]} required[i])];

minimize Total_Cost:
   sum {j in SCHEDS} Work[j];

subject to Shift_Needs {i in SHIFTS}:
   sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j] >= required[i];
```

. . . don't need zero-one variables

Generality

Other possibilities

- `and`, `or`, `not`
- `count`, `atleast`, `atmost`
- `alldifferent`, `numberof` (“global” constraints)
- variables in subscripts (“element” constraints)
- object-valued, set-valued variables

Implementations

- Comet (Dynadec)
- LINGO (LINDO Systems)
- OPL (ILOG)

. . . ***only “captive” languages so far***