Identifying Good Near-Optimal Formulations for Hard Mixed-Integer Programs

Robert Fourer

AMPL Optimization Inc.
4er@ampl.com

7th INFORMS Optimization Society Conference
Denver, CO — 23-25 March 2018

Session SA13: Tutorial

Identifying Good Near-Optimal Formulations for Hard Mixed-Integer Programs

When an exact mixed-integer programming formulation resists attempts at solution, sometimes much better results can be achieved by “cheating” a bit on the formulation. Typically, a judicious choice of reformulation, restriction, or decomposition serves to make the problem easier, in a way not guaranteed to preserve the solution's optimality but highly unlikely to make much of a difference given the model and data of interest. This tutorial illustrates such an approach through a series of case studies. All rely on trial and error, a flexible modeling language, and a good general-purpose solver, and each is seen to be founded on one or two simple ideas that have the potential to be more broadly applied.

Outline

Breaking up

- Work scheduling
- Balanced dinner assignment
- Progressive party assignment

Throwing out

- Roll cutting

Cutting off

- Paint chip cutting
- Balanced team assignment

Reformulating

- Optimization of integer quadratic objectives
- Roll cutting with constraints

Breaking Up 1

Work Scheduling

Cover demands for workers

- Each "shift" requires a certain number of employees
- Each employee works a certain "schedule" of shifts
- *If a schedule is used in the solution,*
  it must be assigned at least a certain minimum number of workers

Minimize total workers needed

- Which schedules are used?
- How many workers are assigned to each schedule?

Breaking Up 1

Work Scheduling

Sets and parameters

```
set SHIFTS;    # set of shifts
param Nsched;  # number of schedules

param required {SHIFTS} >= 0;

               # number of workers needed for each shift

set SCHED {1..Nsched} within SHIFTS;

               # subset of shifts that make up each schedule

param must_assign >= 0;

               # fewest workers that can be assigned
               # to each schedule that is used
```

Work Scheduling

Test data

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

param required :=  Mon1 100  Mon2 78  Mon3 52
                   Tue1 100  Tue2 78  Tue3 52
                   Wed1 100  Wed2 78  Wed3 52
                   Thu1 100  Thu2 78  Thu3 52
                   Fri1 100  Fri2 78  Fri3 52
                   Sat1 100  Sat2 78 ;

param Nsched := 126 ;

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

Work Scheduling

Model using zero-one variables

```
var Work {1..Nsched} >= 0 integer;
var Use  {1..Nsched} >= 0 binary;

minimize Total_Cost:
   sum {j in 1..Nsched} Work[j];

subject to Shift_Needs {i in SHIFTS}:
   sum {j in 1..Nsched: i in SCHED[j]} Work[j] >= required[i];

subject to Least_Use1 {j in 1..Nsched}:
   must_assign * Use[j] <= Work[j];
subject to Least_Use2 {j in 1..Nsched}:
   Work[j] <= (max {i in SCHED[j]} required[i]) * Use[j];
```

Work Scheduling

Branch & bound (CPLEX)

must_assign	nodes	iterations	seconds
14	3301	19914	< 1
15	4269	30559	< 1
16	11748	80476	1
17	1499038	9947799	132
18	1332555	9773201	133
19	41545429	345118936	4218
20	45801	251360	5
21	23989	139139	3
22	9944	63943	2
23	16733	102195	2
24	30968	152377	4

Work Scheduling

Branch & bound (Gurobi)

must_assign	nodes	iterations	seconds
14	435	7019	< 1
15	125	2651	< 1
16	1507	21529	1
17	310597	21726050	732
18	10187784	205022630	4729
19	11334581	176269773	2824
20	159613	2578818	54
21	180147	3725402	74
22	76365	1566823	31
23	147347	2551751	52
24	141423	3091532	55

Breaking Up 1

Work Scheduling

Branch & bound (Xpress)

must_assign	nodes	iterations	seconds
14	566		< 1
15	31894		6
16	361328		56
17	3349425		415
18	10883479		1702
19	17317835		2151
20	342415		40
21	132047		23
22	167631		23
23	85591		14
24	67609		13

Work Scheduling

Relaxation optimal values

must_assign	relax all	relax Work	relax none
14	265.6	265.6	266
15	265.6	265.6	266
16	265.6	265.6	266
17	265.6	266.5	267
18	265.6	267.5	268
19	265.6	268.5	269
20	265.6	269	269
21	265.6	269	269
22	265.6	269	269
23	265.6	269	269
24	265.6	269	269

Fractional solution ***Integer solution***

Work Scheduling

Two-step 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 . . .

Work Scheduling

Typical run of indirect approach

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

ampl: let must_assign := 19;
ampl: option solver cplex;

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

ampl: solve;

CPLEX 12.8.0.0: optimal integer solution; objective 268.5
31370114 MIP simplex iterations;
1990542 branch-and-cut nodes

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

ampl: solve;

CPLEX 12.8.0.0 : optimal solution; objective 269
5 MIP simplex iterations;
0 branch-and-cut nodes
```

Breaking Up 1

Work Scheduling

Two-step approach (CPLEX)

must_assign	one-step	two-step
14	< 1	< 1
15	< 1	2
16	1	2
17	132	62
18	133	175
19	4218	298
20	5	2
21	3	2
22	2	2
23	2	3
24	4	3

Balanced Dinner Assignment

Setting

- meeting of employees from around the world
  at New York offices of a Wall Street firm

Given

- title, location, department, sex,
  for each of about 1000 people

Assign

- these people to around 25 dinner groups

So that

- the groups are as “diverse” as possible

Minimum "Variation" Model

```
set PEOPLE;   # individuals to be assigned

set CATEG;
param type {PEOPLE,CATEG} symbolic;

              # categories by which people are classified;
              # type of each person in each category

param numberGrps integer > 0;
param minInGrp integer > 0;
param maxInGrp integer >= minInGrp;

              # number of groups; bounds on size of groups

var Assign {i in PEOPLE, j in 1..numberGrps} binary;

              # Assign[i,j] is 1 if and only if
              # person i is assigned to group j
```

A similar approach: "Market Sharing: Assigning Retailers to Company Divisions," in:
H.P. Williams, *Model Building in Mathematical Programming,* 3rd edition, Wiley (1990), pp. 259–260.

Thanks also to Collette Coullard.

Breaking Up 2

(definition of variation)

```
set TYPES {k in CATEG} := setof {i in PEOPLE} type[i,k];

              # all types found in each category

var MinType {k in CATEG, TYPES[k]};
var MaxType {k in CATEG, TYPES[k]};

              # fewest and most people of each type, over all groups

subj to MinTypeDefn {j in 1..numberGrps, k in CATEG, l in TYPES[k]}:
   MinType[k,l] <= sum {i in PEOPLE: type[i,k] = l} Assign[i,j];

subj to MaxTypeDefn {j in 1..numberGrps, k in CATEG, l in TYPES[k]}:
   MaxType[k,l] >= sum {i in PEOPLE: type[i,k] = l} Assign[i,j];

              # values of MinTypeDefn and MaxTypeDefn variables
              # must be consistent with values of Assign variables
```

Breaking Up 2

(objective, assignment constraints)

```
minimize TotalVariation:
   sum {k in CATEG, l in TYPES[k]} (MaxType[k,l] - MinType[k,l]);

              # Total variation over all types

subj to AssignAll {i in PEOPLE}:
   sum {j in 1..numberGrps} Assign[i,j] = 1;

              # Each person must be assigned to one group

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

              # Each group must have an acceptable size
```

Data *(210 people)*

```
set PEOPLE :=
   BIW   AJH   FWI   IGN   KWR   KKI   HMN   SML   RSR   TBR
   KRS   CAE   MPO   CAR   PSL   BCG   DJA   AJT   JPY   HWG
   TLR   MRL   JDS   JAE   TEN   MKA   NMA   PAS   DLD   SCG
   VAA   FTR   GCY   OGZ   SME   KKA   MMY   API   ASA   JLN
   JRT   SJO   WMS   RLN   WLB   SGA   MRE   SDN   HAN   JSG
   AMR   DHY   JMS   AGI   RHE   BLE   SMA   BAN   JAP   HER
   MES   DHE   SWS   ACI   RJY   TWD   MMA   JJR   MFR   LHS
   JAD   CWU   PMY   CAH   SJH   EGR   JMQ   GGH   MMH   JWR
   MJR   EAZ   WAD   LVN   DHR   ABE   LSR   MBT   AJU   SAS
   JRS   RFS   TAR   DLT   HJO   SCR   CMY   GDE   MSL   CGS
   HCN   JWS   RPR   RCR   RLS   DSF   MNA   MSR   PSY   MET
   DAN   RVY   PWS   CTS   KLN   RDN   ANV   LMN   FSM   KWN
   CWT   PMO   EJD   AJS   SBK   JWB   SNN   PST   PSZ   AWN
   DCN   RGR   CPR   NHI   HKA   VMA   DMN   KRA   CSN   HRR
   SWR   LLR   AVI   RHA   KWY   MLE   FJL   ESO   TJY   WHF
   TBG   FEE   MTH   RMN   WFS   CEH   SOL   ASO   MDI   RGE
   LVO   ADS   CGH   RHD   MBM   MRH   RGF   PSA   TTI   HMG
   ECA   CFS   MKN   SBM   RCG   JMA   EGL   UJT   ETN   GWZ
   MAI   DBN   HFE   PSO   APT   JMT   RJE   MRZ   MRK   XYF
   JCO   PSN   SCS   RDL   TMN   CGY   GMR   SER   RMS   JEN
   DWO   REN   DGR   DET   FJT   RJZ   MBY   RSN   REZ   BLW ;
```

Breaking Up 2

Data *(4 categories, 18 types)*

```
set CATEG := dept loc rate title ;

param type:

     dept       loc     rate    title   :=

BIW   NNE   Peoria         A   Assistant
KRS   WSW   Springfield   B   Assistant
TLR   NNW   Peoria         B   Adjunct
VAA   NNW   Peoria         A   Deputy
JRT   NNE   Springfield   A   Deputy
AMR   SSE   Peoria         A   Deputy
MES   NNE   Peoria         A   Consultant
JAD   NNE   Peoria         A   Adjunct
MJR   NNE   Springfield   A   Assistant
JRS   NNE   Springfield   A   Assistant
HCN   SSE   Peoria         A   Deputy
DAN   NNE   Springfield   A   Adjunct

.......

param numberGrps := 12 ;
param minInGrp := 16 ;
param maxInGrp := 19 ;
```

Solving for Minimum Variation

```
ampl: model BalAssign.mod;
ampl: data BalAssign.dat;

ampl: option solver gurobi;
ampl: option show_stats 1;
ampl: solve;

2556 variables:
        2520 binary variables
        36 linear variables
654 constraints, all linear; 25632 nonzeros
        210 equality constraints
        432 inequality constraints
        12 range constraints
1 linear objective; 36 nonzeros.

Gurobi 7.5.0: optimal solution; objective 16
338028 simplex iterations
1751 branch-and-cut nodes
```

66.344 sec

Breaking Up 2

Revised Formulation

```
var MinType {k in CATEG, t in TYPES[k]}
   <= floor (card {i in PEOPLE: type[i,k] = t} / numberGrps);

var MaxType {k in CATEG, t in TYPES[k]
   >= ceil (card {i in PEOPLE: type[i,k] = t} / numberGrps);
```

```
ampl: include BalAssign+.run

Presolve eliminates 72 constraints.
...

Gurobi 7.5.0: optimal solution; objective 16
2203 simplex iterations
```

0.203 sec

Scaling Up

Real model was more complicated

- Rooms hold from 20–25 to 50–55 people
- Must avoid isolating assignments:
  - a person is "isolated" in a group that contains
    no one from the same location
    with the same or "adjacent" title

Problem was too big

- Aggregate people who match in all categories
  (986 people, but only 287 different kinds)
- Solve first for title and location only,
  then for refinement to department and sex
- Stop at first feasible solution to
  title-location problem

Breaking Up 2

Full "Title-Location" Model

```
set PEOPLE ordered;

param title {PEOPLE} symbolic;
param loc   {PEOPLE} symbolic;

set TITLE ordered;
   check {i in PEOPLE}: title[i] in TITLE;

set LOC = setof {i in PEOPLE} loc[i];

set TYPE2 = setof {i in PEOPLE} (title[i],loc[i]);
param number2 {(i1,i2) in TYPE2} =
   card {i in PEOPLE: title[i]=i1 and loc[i]=i2};

set REST ordered;

param loDine {REST} integer > 10;
param hiDine {j in REST} integer >= loDine[j];

param loCap := sum {j in REST} loDine[j];
param hiCap := sum {j in REST} hiDine[j];

param loFudge := ceil ((loCap less card {PEOPLE}) / card {REST});
param hiFudge := ceil ((card {PEOPLE} less hiCap) / card {REST});
```

Breaking Up 2

(variables, objective, assignment constraints)

```
var Assign2 {TYPE2,REST} integer >= 0;

var Dev2Title {TITLE} >= 0;
var Dev2Loc {LOC} >= 0;

minimize Deviation:
   sum {i1 in TITLE} Dev2Title[i1] + sum {i2 in LOC} Dev2Loc[i2];

subject to Assign2Type {(i1,i2) in TYPE2}:
   sum {j in REST} Assign2[i1,i2,j] = number2[i1,i2];

subject to Assign2Rest {j in REST}:
   loDine[j] - loFudge
      <= sum {(i1,i2) in TYPE2} Assign2[i1,i2,j]
         <= hiDine[j] + hiFudge;
```

Breaking Up 2

(parameters for defining "variation")

```
param frac2title {i1 in TITLE}
   = sum {(i1,i2) in TYPE2} number2[i1,i2] / card {PEOPLE};

param frac2loc {i2 in LOC}
   = sum {(i1,i2) in TYPE2} number2[i1,i2] / card {PEOPLE};

param expDine {j in REST}
   = if loFudge > 0 then loDine[j] else
     if hiFudge > 0 then hiDine[j] else (loDine[j] + hiDine[j]) / 2;

param loTargetTitle {i1 in TITLE, j in REST} =
   floor (round (frac2title[i1] * expDine[j], 6));
param hiTargetTitle {i1 in TITLE, j in REST} =
   ceil (round (frac2title[i1] * expDine[j], 6));

param loTargetLoc {i2 in LOC, j in REST} =
   floor (round (frac2loc[i2] * expDine[j], 6));
param hiTargetLoc {i2 in LOC, j in REST} =
   ceil (round (frac2loc[i2] * expDine[j], 6));
```

Breaking Up 2

(constraints defining "variation")

```
subject to Lo2TitleDefn {i1 in TITLE, j in REST}:
   Dev2Title[i1] >=
      loTargetTitle[i1,j] - sum {(i1,i2) in TYPE2} Assign2[i1,i2,j];

subject to Hi2TitleDefn {i1 in TITLE, j in REST}:
   Dev2Title[i1] >=
      sum {(i1,i2) in TYPE2} Assign2[i1,i2,j] - hiTargetTitle[i1,j];

subject to Lo2LocDefn {i2 in LOC, j in REST}:
   Dev2Loc[i2] >=
      loTargetLoc[i2,j] - sum {(i1,i2) in TYPE2} Assign2[i1,i2,j];

subject to Hi2LocDefn {i2 in LOC, j in REST}:
   Dev2Loc[i2] >=
      sum {(i1,i2) in TYPE2} Assign2[i1,i2,j] - hiTargetLoc[i2,j];
```

Breaking Up 2

(parameters for ruling out "isolation")

```
set ADJACENT {i1 in TITLE} =
   (if i1 <> first(TITLE) then {prev(i1)} else {}) union
   (if i1 <> last(TITLE) then {next(i1)} else {});

set ISO = {(i1,i2) in TYPE2: (i2 <> "Unknown") and
   ((number2[i1,i2] >= 2) or
    (number2[i1,i2] = 1 and
      sum {ii1 in ADJACENT[i1]: (ii1,i2) in TYPE2}
         number2[ii1,i2] > 0)) };

param give {ISO} default 2;
param giveTitle {TITLE} default 2;
param giveLoc {LOC} default 2;

param upperbnd {(i1,i2) in ISO, j in REST} =
   min (ceil((number2[i1,i2]/card {PEOPLE}) * hiDine[j]) + give[i1,i2],
        hiTargetTitle[i1,j] + giveTitle[i1],
        hiTargetLoc[i2,j] + giveLoc[i2],
        number2[i1,i2]);
```

Breaking Up 2

(constraints ruling out "isolation")

```
var Lone {(i1,i2) in ISO, j in REST} binary;

subj to Isolation1 {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] <= upperbnd[i1,i2,j] * Lone[i1,i2,j];

subj to Isolation2a {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] +
      sum {ii1 in ADJACENT[i1]: (ii1,i2) in TYPE2} Assign2[ii1,i2,j]
         >= 2 * Lone[i1,i2,j];

subj to Isolation2b {(i1,i2) in ISO, j in REST}:
   Assign2[i1,i2,j] >= Lone[i1,i2,j];
```

Original Success

First problem

- using OSL: 128 “supernodes”, 6.7 hours
- using CPLEX 2.1: took too long

Second problem

- using CPLEX 2.1: 864 nodes, 3.6 hours
- using OSL: 853 nodes, 4.3 hours

Finish

- Refine to individual assignments: a trivial LP
- Make table of assignments using AMPL printf command
- Ship table to client, who imports to database

Solver Improvements

CPLEX 3.0

- First problem: 1200 nodes, 1.1 hours
- Second problem: 1021 nodes, 1.3 hours

CPLEX 4.0

- First problem: 517 nodes, 5.4 minutes
- Second problem: 1021 nodes, 21.8 minutes

CPLEX 9.0

- First problem: 560 nodes, 83.1 seconds
- Second problem: 0 nodes, 17.9 seconds

Solver Improvements

CPLEX 12.1

- First problem: 0 nodes, 9.5 seconds
- Second problem: 0 nodes, 1.5 seconds

Gurobi 2.0

- First problem: 0 nodes, 13.5 seconds
- Second problem: 0 nodes, 1.6 seconds

Performance Today

CPLEX 12.8

- First problem: 0 nodes, 4.4 seconds
- Second problem: 0 nodes, 0.8 seconds

Gurobi 7.5

- First problem: 0 nodes, 3.8 seconds
- Second problem: 0 nodes, 0.6 seconds

Objective = total deviation from balance

- First problem = 12
- Second problem = 4
- *Total = 16*

The Original Problem Today

Gurobi 7.5

Objective values

- 16: 9 seconds
- 15: 54 seconds
- 14: 83 seconds
- 13: 99 seconds

Lower bounds

- 3: 10 seconds
- 10: 60 seconds
- 12: 4517 seconds

Progressive Party Assignment

Setting

- yacht club holding a party
- each boat has a certain crew size & guest capacity

Decisions

- choose a minimal number yachts as “hosts”
- assign each non-host crew to visit a host yacht
- . . . in each of 6 periods

Requirements

- no yacht’s capacity is exceeded
- no crew visits the same yacht more than once
- no two crews meet more than once

Breaking Up 3

Progressive Party Problem

Parameters & variables

```
param B > 0, integer;
set BOATS := 1 .. B;

param capacity {BOATS} integer >= 0;
param crew {BOATS} integer > 0;
param guest_cap {i in BOATS} := capacity[i] less crew[i];

param T > 0, integer;
set TIMES := 1..T;


var Host {i in BOATS} binary;       # i is a host boat

var Visit {i in BOATS, j in BOATS, t in TIMES: i <> j} binary;
                                    # crew of j visits party on i at t

var Meet {i in BOATS, j in BOATS, t in TIMES: i < j} >= 0, <= 1;
                                    # crews of i and j meet at t
```

Erwin Kalvelagen, On Solving the Progressive Party Problem as a MIP.
Computers & Operations Research **30** (2003) 1713-1726.

Breaking Up 3

Progressive Party Problem

Host objective and constraints

```
minimize TotalHosts: sum {i in BOATS} Host[i];

        # minimize total host boats

set MUST_BE_HOST within BOATS;

subj to MustBeHost {i in MUST_BE_HOST}: Host[i] = 1;

       # some boats are designated host boats

set MUST_BE_GUEST within BOATS;

subj to MustBeGuest {i in MUST_BE_GUEST}: Host[i] = 0;

       # some boats (the virtual boats) are designated guest boats

param mincrew := min {j in BOATS} crew[j];

subj to NeverHost {i in BOATS: guest_cap[i] < mincrew}: Host[i] = 0;

       # boats with very limited guest capacity can never be hosts
```

Progressive Party Problem

Host-Visit constraints

```
subj to PartyHost {i in BOATS, j in BOATS, t in TIMES: i <> j}:
   Visit[i,j,t] <= Host[i];

       # parties must occur on host boats

subj to Cap {i in BOATS, t in TIMES}:
   sum {j in BOATS: j <> i} crew[j] * Visit[i,j,t] <= guest_cap[i] * Host[i];

       # boats may not have more visitors than they can handle

subj to CrewHost {j in BOATS, t in TIMES}:
   Host[j] + sum {i in BOATS: i <> j} Visit[i,j,t] = 1;

       # every crew is either hosting or visiting a party

subj to VisitOnce {i in BOATS, j in BOATS: i <> j}:
   sum {t in TIMES} Visit[i,j,t] <= Host[i];

       # a crew may visit a host at most once
```

Progressive Party Problem

Meet-Visit constraints

```
subj to Link {i in BOATS,
      j in BOATS, jj in BOATS, t in TIMES: i <> j and i <> jj and j < jj}:
   Meet[j,jj,t] >= Visit[i,j,t] + Visit[i,jj,t] - 1;

       # meetings occur when two crews are on same host at same time

subj to MeetOnce {j in BOATS, jj in BOATS: j < jj}:
   sum {t in TIMES} Meet[j,jj,t] <= 1;

       # two crews may meet at most once
```

Progressive Party Problem

Data

```
param B := 42;
param T := 6;

param:  capacity  crew :=
    1        6      2
    2        8      2
    3       12      2
    4       12      2
    5       12      4
    6       12      4
    7       12      4
........
   37        6      4
   38        6      5
   39        9      7
   40        0      2
   41        0      3
   42        0      4 ;

set MUST_BE_HOST := 1  2  3 ;
set MUST_BE_GUEST := 40  41  42 ;
```

Breaking Up 3

Direct Approach

rounds	variables	constraints	nodes	iterations	seconds	objective
1	1272	983	0	540	< 1	13
2	3259	39479	0	2752	5	13
3	5982	59339	0	8389	19	13
4	7964	78458	0	7243	32	13
5	9946	97577	0	26478	158	13
6	11928	116696	0	73796	443	13
7	13910	135815	781	760617	4306	13
8	15892	154934	> *1273*	> *1898720*	> *25446*	*14*
9	17874	174053				
10	19856	193172				

Multi-Step Approach

Determine hosts

- solve 1-period problem
- fix hosts
- fix 1^{st}-period visits

Determine visits: for round t = 2, 3, . . .

- solve period-*t* problem
- fix period-*t* visits

Multi-Step Script

```
model partyS.mod;
data partyS.dat;

option solver cplex;

# -------

let T := 1;

repeat {

   solve;

   if T = 1 then fix Host;

   if solve_result = "solved" then {
      let T := T + 1;
      fix {i in BOATS, j in BOATS: i <> j} Visit[i,j,T-1];
   }
   else break;

};
```

Multi-Step Run *(periods 1 to 3)*

```
ampl: include partyM.run

Reduced MIP has 983 rows, 1272 columns, and 4364 nonzeros.
Reduced MIP has 1272 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.20 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
540 MIP simplex iterations
0 branch-and-bound nodes

Reduced MIP has 138 rows, 329 columns, and 927 nonzeros.
Reduced MIP has 329 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.02 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
0 MIP simplex iterations
0 branch-and-bound nodes

Reduced MIP has 231 rows, 300 columns, and 1082 nonzeros.
Reduced MIP has 300 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.02 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
0 MIP simplex iterations
0 branch-and-bound nodes
```

Breaking Up 3

Multi-Step Run *(periods 4 to 6)*

```
Reduced MIP has 299 rows, 271 columns, and 1188 nonzeros.
Reduced MIP has 271 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.02 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
0 MIP simplex iterations
0 branch-and-bound nodes

Reduced MIP has 321 rows, 242 columns, and 1199 nonzeros.
Reduced MIP has 242 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.02 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
0 MIP simplex iterations
0 branch-and-bound nodes

Reduced MIP has 302 rows, 213 columns, and 1120 nonzeros.
Reduced MIP has 213 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.02 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
0 MIP simplex iterations
0 branch-and-bound nodes
```

Breaking Up 3

Multi-Step Run *(periods 7 to 9)*

```
Reduced MIP has 269 rows, 184 columns, and 999 nonzeros.
Reduced MIP has 184 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.02 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
0 MIP simplex iterations
0 branch-and-bound nodes

Reduced MIP has 213 rows, 154 columns, and 793 nonzeros.
Reduced MIP has 154 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) = 0.02 sec

CPLEX 12.8.0.0: optimal integer solution; objective 13
0 MIP simplex iterations
0 branch-and-bound nodes

Reduced MIP has 160 rows, 117 columns, and 564 nonzeros.
Reduced MIP has 117 binaries, 0 generals, 0 SOSs, and 0 indicators.

Total (root+branch&cut) =    0.05 sec. (28.42 ticks)

CPLEX 12.8.0.0: integer infeasible.
136 MIP simplex iterations
0 branch-and-bound nodes
```

Multi-Step Approach

Is this optimal?

- No
- Solver returns one of many period-1 solutions
- Different period-1 solutions
  produce different period-2 solutions . . .
- Different period 1, 2, 3, . . . solutions
  result in different numbers of feasible stages

Results of 100 runs with different seeds

feasible stages	number of occurrences
7	0
8	4
9	94
10	2
11	0

Roll Cutting

Cut large "raw" rolls into smaller ones

- All raw rolls the same width
- Various smaller widths ordered, in varied amounts

Minimize total raw rolls cut

- Define cutting patterns
- Choose how many raw rolls to cut with each pattern . . .
  - generate patterns iteratively based on dual values (the Gilmore-Gomory method)
  - enumerate all nondominated patterns in advance

Roll Cutting

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
```

Throwing Out

Roll Cutting

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;
```

Roll Cutting

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;
   };
```

Throwing Out

Roll Cutting

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;
      }
   }
```

Roll Cutting

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

Throwing Out

Roll Cutting

Pattern generation: Gilmore-Gomory approach

```
ampl: include cutPatGen.run
  35.36  -1.39e-01    0  6  0  0  0  0  0  1  0  0  0  1  0  0  0  0
  33.66  -1.30e-01    6  0  0  0  0  0  0  0  0  1  0  0  1  0  0  0
  33.55  -8.33e-02    0  0  0  9  0  0  1  0  0  0  0  0  0  0  0  0
  33.52  -6.01e-02    0  0  5  0  0  0  0  1  0  0  0  6  0  1  0  0
  33.50  -5.98e-02    0  0  4  0  0  0  0  8  0  0  0  0  0  0  0  0
  33.31  -5.88e-02    0  0  0  0  0  0  0  0  0  0  0 16  1  0  0  0
  33.30  -5.45e-02    0  0  0  0  0  0  1 11  0  0  2  0  0  0  0  0
  33.14  -5.33e-02    0  0  0  0  0  0  0 12  0  1  0  0  1  0  0  0
  33.07  -3.25e-02    0  0  0  0  0  0  6  0  0  0  0  0  1  9  0  0
  33.02  -2.92e-02    0  0  0  0  0  1  9  0  0  2  0  0  0  0  0  1
  32.97  -2.66e-02    0  0  0  0  0  0  0  0  0  0 16  0  0  0  0  1
  32.96  -2.11e-02    0  0  0  0  0  0  0  0  0 12  1  0  1  0  0  1
  32.92  -1.46e-02    0  0  0  0  0  8  0  0  0  1  0  0  2  0  2  0
  32.92  -1.18e-02    0  0  0  0  0  0  0  0  0  0  0  0  7  0 12  0
  32.90  -1.09e-02    0  0  0  0  0  0  0  0  0  1  0  0 16  0  0  0
  32.88  -8.39e-03    0  1  0  0  0  0  0  0  0  0  0  0  0  0  0 19
  32.88  -7.94e-03    0  1  0  0  9  0  0  0  0  0  0  0  0  0  0  1
  32.87  -8.93e-03    0  0  0  0  8  0  0  0  0  4  0  0  0  0  0  0
  32.86  -5.04e-03    0  5  0  1  0  0  0  0  0  0  3  0  0  0  0  0
  32.85  -4.91e-03    0  5  0  0  0  0  0  0  1  3  0  0  0  0  0  0
  32.82  -4.92e-03    0  4  0  1  0  0  4  0  0  0  0  0  0  0  0  0
```

Roll Cutting

Pattern generation: Continuous relaxation

```
  32.81  -4.51e-03    0  4  0  0  0  0  2  0  3  0  0  0  0  0  1  0
  32.81  -4.25e-03    0  5  0  0  0  1  0  1  0  0  1  1  0  0  0  0
  32.80  -7.00e-06
Optimal relaxation:  32.7965 rolls
  0.8333 of:  6 x 25.000  1 x 12.000  1 x 10.000
  3.5000 of:  4 x 18.000  8 x 12.500
  0.5662 of: 16 x 10.125  1 x 10.000
  4.5049 of: 12 x 12.500  1 x 12.000  1 x 10.000
  1.6667 of:  6 x 13.875  1 x 10.000  9 x  8.750
  0.0584 of:  1 x 15.375  9 x 13.875  2 x 12.000  1 x  7.750
  0.8100 of: 12 x 12.000  1 x 10.250  1 x 10.000  1 x  7.750
  1.6331 of:  7 x 10.000 12 x  8.500
  1.4492 of:  1 x 12.000 16 x 10.000
  0.0829 of:  1 x 24.750 19 x  7.750
  2.5556 of:  1 x 24.750  9 x 15.500  1 x  7.750
  0.0828 of:  5 x 24.750  1 x 17.500  3 x 10.250
  4.7921 of:  5 x 24.750  1 x 12.250  3 x 12.000
  3.9172 of:  4 x 24.750  1 x 17.500  4 x 13.875
  1.4026 of:  4 x 24.750  2 x 13.875  3 x 12.250  1 x  8.500
  4.9416 of:  5 x 24.750  1 x 15.375  1 x 12.500  1 x 10.250  1 x 10.125
WASTE =  0.00%
```

Throwing Out

Roll Cutting

Pattern generation: Rounded up to integer

```
Rounded up to integer:  40 rolls
  Cut      1  4  1  5  2  1  1  2  2  1  3  1  5  4  2  5

  25.00    6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  24.75    0  0  0  0  0  0  0  0  0  1  1  5  5  4  4  5
  18.00    0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  17.50    0  0  0  0  0  0  0  0  0  0  0  1  0  1  0  0
  15.50    0  0  0  0  0  0  0  0  0  0  9  0  0  0  0  0
  15.38    0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  1
  13.88    0  0  0  0  6  9  0  0  0  0  0  0  0  4  2  0
  12.50    0  8  0 12  0  0  0  0  0  0  0  0  0  0  0  1
  12.25    0  0  0  0  0  0  0  0  0  0  0  0  1  0  3  0
  12.00    1  0  0  1  0  2 12  0  1  0  0  0  3  0  0  0
  10.25    0  0  0  0  0  0  1  0  0  0  0  3  0  0  0  1
  10.12    0  0 16  0  0  0  0  0  0  0  0  0  0  0  0  1
  10.00    1  0  1  1  1  0  1  7 16  0  0  0  0  0  0  0
   8.75    0  0  0  0  9  0  0  0  0  0  0  0  0  0  0  0
   8.50    0  0  0  0  0  0  0 12  0  0  0  0  0  0  1  0
   7.75    0  0  0  0  0  1  1  0  0 19  1  0  0  0  0  0

WASTE = 18.01%
```

Throwing Out

Roll Cutting

Pattern generation: Integer solution from patterns generated

Best integer: 35 rolls

Cut	1	1	2	1	1	4	1	1	1	6	1	2	2	4	3	4
25.00	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0
24.75	0	0	0	0	0	6	0	0	0	0	0	0	1	4	4	5
18.00	9	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0
17.50	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
15.50	0	11	0	0	0	0	0	0	0	0	0	0	9	0	0	0
15.38	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1
13.88	0	0	0	0	0	0	0	0	1	0	9	0	0	4	2	0
12.50	0	0	0	0	0	1	0	1	11	12	0	0	0	0	0	1
12.25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
12.00	0	0	0	0	0	0	1	0	0	1	2	12	0	0	0	0
10.25	0	0	0	0	0	0	0	0	2	0	0	1	0	0	0	1
10.12	0	0	0	0	0	1	0	6	0	0	0	0	0	0	0	1
10.00	0	0	17	0	0	0	1	0	0	1	0	1	0	0	0	0
8.75	0	0	0	19	0	0	0	1	0	0	0	0	0	0	0	0
8.50	0	0	0	0	20	0	0	0	0	0	0	0	0	0	1	0
7.75	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0

WASTE = 6.30%

Roll Cutting

Pattern enumeration: All non-dominated patterns

```
ampl: include cutPatEnum100.run
   10000   4   0   0   0   1   0   0   0   0   0   0   0   1   0   1   4
   20000   3   1   1   2   0   0   0   0   0   0   0   0   0   0   2   0
   30000   3   1   0   0   0   0   2   0   0   0   3   0   1   0   0   0
   40000   3   0   2   0   0   0   0   1   0   0   4   0   0   0   0   0
   50000   3   0   1   0   1   0   0   0   2   1   0   0   1   1   0   1
   60000   3   0   1   0   0   0   0   2   0   0   0   2   0   2   0   2
   70000   3   0   0   2   0   0   0   1   0   0   0   0   0   3   0   3
.......
27270000   0   0   0   0   0   0   0   0   0   2   6   1   4   4   0   0
27280000   0   0   0   0   0   0   0   0   0   2   1   0   3   1  11   0
27290000   0   0   0   0   0   0   0   0   0   1   4   2   0   4   2   6
27300000   0   0   0   0   0   0   0   0   0   1   1   1   4   4   0   8
27310000   0   0   0   0   0   0   0   0   0   0   7   0   4   3   2   2
27320000   0   0   0   0   0   0   0   0   0   0   3   0   1   7   8   0
27330000   0   0   0   0   0   0   0   0   0   0   1   0   4   6   3   5
```

. . . too many columns for my computer

Throwing Out

Roll Cutting

Pattern enumeration: Every 100th non-dominated pattern

```
ampl: include cutPatEnum100.run
   10000  4  0  0  0  1  0  0  0  0  0  0  0  1  0  1  4
   20000  3  1  1  2  0  0  0  0  0  0  0  0  0  0  2  0
   30000  3  1  0  0  0  0  2  0  0  0  3  0  1  0  0  0
   40000  3  0  2  0  0  0  0  1  0  0  4  0  0  0  0  0
   50000  3  0  1  0  1  0  0  0  2  1  0  0  1  1  0  1
   60000  3  0  1  0  0  0  0  2  0  0  0  2  0  2  0  2
   70000  3  0  0  2  0  0  0  1  0  0  0  0  0  3  0  3
.......
27270000  0  0  0  0  0  0  0  0  0  2  6  1  4  4  0  0
27280000  0  0  0  0  0  0  0  0  0  2  1  0  3  1 11  0
27290000  0  0  0  0  0  0  0  0  0  1  4  2  0  4  2  6
27300000  0  0  0  0  0  0  0  0  0  1  1  1  4  4  0  8
27310000  0  0  0  0  0  0  0  0  0  0  7  0  4  3  2  2
27320000  0  0  0  0  0  0  0  0  0  0  3  0  1  7  8  0
27330000  0  0  0  0  0  0  0  0  0  0  1  0  4  6  3  5

Gurobi 7.5.0: outlev 1
Optimize a model with 16 rows, 273380 columns and 2024052 nonzeros
Variable types: 0 continuous, 273380 integer (0 binary)
.......
```

Roll Cutting

Pattern enumeration: Every 100th (cont'd)

```
Starting sifting (using dual simplex for sub-problems)...
    Iter     Pivots    Primal Obj      Dual Obj        Time
       0          0     infinity      0.0000000e+00      3s
       1         22    1.1474445e+02   1.2283242e+01      3s
       2         58    4.2188397e+01   2.1998550e+01      3s
       3         95    3.5002421e+01   2.4861376e+01      3s
       4        145    3.3538200e+01   3.1080827e+01      3s
       5        180    3.2923675e+01   3.2599109e+01      3s
       6        242    3.2802177e+01   3.2746499e+01      3s
       7        283    3.2796512e+01   3.2796512e+01      3s
Root relaxation: objective 3.279651e+01, 283 iterations, 0.52 seconds
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0   32.79651    0   15  343.00000   32.79651  90.4%     -    4s
H    0     0                      35.0000000   32.79651  6.30%     -    4s
H    0     0                      34.0000000   32.79651  3.54%     -    5s
H    0     0                      33.0000000   32.79651  0.62%     -   10s

Gurobi 7.5.0: optimal solution; objective 33
362 simplex iterations
1 branch-and-cut nodes
```

Paint Chip Cutting

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

Courtesy of Color Communications Innovations and Collette Coullard.

Chip Cutting

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];
```

Chip Cutting

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

Cutting Off 1

Chip Cutting

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 . . .

Chip Cutting

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;
```

Results *(distant past)*

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

Results *(past)*

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)

Results *(today)*

Without restriction

- 1724 rows, 1719 columns, 5800 nonzeros
- 7,924 simplex iterations
- 159 branch-and-bound nodes
- 1.64 seconds

With restriction

- 2344 rows, 1598 columns, 6975 nonzeros
- 5,336 simplex iterations
- 121 branch-and-bound nodes
- 0.83 seconds

(Gurobi 7.5, 4 threads, 2 processors)

Balanced Team Assignment

Same assignment idea

- Partition people into groups
  - diversity measured by several characteristics
  - each characteristic has several values
- Make groups as diverse as possible

Different formulation

- *Overlap of a person with another person* is the number of characteristics for which they have the same value
- *Total overlap of a person* is the sum of their overlaps with all the other people assigned to the same group
- Minimize the sum of total overlap over all people

Balanced Team Assignment

Small example where branching takes "forever"

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

```
CPLEX 12.8.0.0:

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: deterministic, using up to 4 threads.

Root relaxation solution time = 0.00 sec. (2.05 ticks)
```

Balanced Team Assignment

Active start . . .

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

*     0+    0                        7040.0000       0.0000             100.00%
*     0+    0                        5350.0000       0.0000             100.00%
*     0+    0                        3663.0000       0.0000             100.00%
*     0+    0                        2046.0000       0.0000             100.00%
      0     0        0.0000    59    2046.0000       0.0000        96  100.00%
      0     0        0.0000    61    2046.0000     Cuts: 53       160  100.00%
      0     0        0.0000    58    2046.0000     Cuts: 43       198  100.00%
      0     0        0.0000    59    2046.0000     Cuts: 46       239  100.00%
*     0+    0                         250.0000       0.0000             100.00%
*     0+    0                         214.0000       0.0000             100.00%
      0     2        0.0000    59     214.0000       0.0000       239  100.00%
Elapsed time = 0.22 sec. (119.41 ticks, tree = 0.01 MB)
    400   313      146.6960    36     214.0000       8.1151      7686   96.21%
   1560  1382        0.0000    58     214.0000      13.1250     24013   93.87%
   2897  1009       94.3370    44     214.0000      19.5907     36284   90.85%
   5964  3822      123.6801    34     214.0000      29.8294     62569   86.06%
   9387  6483      189.0373    31     214.0000      34.5987     85532   83.83%
  13094  9402      182.9750    25     214.0000      37.4645    113694   82.49%
  16884 13240       65.0250    48     214.0000      39.4557    150255   81.56%
.................
```

Balanced Team Assignment

. . . but after a day, the tree is still growing . . .

```
                Nodes                                           Cuts/
      Node       Left    Objective IInf  Best Integer   Best Node    ItCnt      Gap
 .................
 239571977 208155760     139.9859    49     212.0000     131.3092  1.84e+09   38.06%
 239710829 208285367     204.9129    20     212.0000     131.3145  1.85e+09   38.06%
 239843771 208395047     137.6135    42     212.0000     131.3193  1.85e+09   38.06%
 239955358 208492610     145.4060    44     212.0000     131.3234  1.85e+09   38.06%
 240087477 208609769     171.3730    28     212.0000     131.3282  1.85e+09   38.05%
 240195933 208699779     172.5904    39     212.0000     131.3322  1.85e+09   38.05%
 240314799 208804386     190.5755    30     212.0000     131.3364  1.85e+09   38.05%
 240409481 208885021     197.7286    36     212.0000     131.3400  1.85e+09   38.05%
 240533493 208992546     173.2190    36     212.0000     131.3443  1.85e+09   38.05%
Elapsed time = 92376.55 sec. (44895207.38 ticks, tree = 102570.01 MB)
Nodefile size = 100522.01 MB (50524.20 MB after compression)
 240665098 209102490       cutoff           212.0000     131.3490  1.85e+09   38.04%
 240767864 209195103     180.6965    30     212.0000     131.3528  1.85e+09   38.04%
 240872761 209278303     156.8931    34     212.0000     131.3566  1.85e+09   38.04%
 240969723 209369979     197.2533    23     212.0000     131.3600  1.85e+09   38.04%
 241071358 209456164     173.0975    36     212.0000     131.3639  1.85e+09   38.04%
<BREAK> (cplex)
```

Balanced Team 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 lower bounds
  - theoretically correct: `4 * (maxInGrp-1)` → `0.0`
  - empirically justified: `1 * (maxInGrp-1)` → `160.5`

Balanced Team 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 lower bounds
  - `floor(card(PEOPLE)/numberGrps) - 1`
    `ceil (card(PEOPLE)/numberGrps) + 1` → 160.5
  - `floor(card(PEOPLE)/numberGrps)`
    `ceil (card(PEOPLE)/numberGrps)` → 179.6

Balanced Team 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;
```

- Specify exact sizes of all groups
- Exact sizes give stronger lower bounds
  - tightened limits on group sizes → `179.6`
  - exact sizes → `183.4`

Balanced Team Assignment

Incorporating enhancements . . .

```
ampl: model gs1f.mod;
ampl: data gs1b.dat;
ampl: option solver cplex;
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: deterministic, using up to 4 threads.
Root relaxation solution time = 0.00 sec. (7.44 ticks)

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

*     0+    0                      252.0000      67.0000             73.41%
      0     0      183.3626   134      252.0000    183.3626      221    27.24%
.......
```

Balanced Team Assignment

Much more promising start . . .

```
        Nodes                                               Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt     Gap
.......
      0     0      188.5842   101      252.0000      Fract: 6      312  25.16%
      0     0      190.3775   102      252.0000      Cuts: 34      661  24.45%
      0     0      190.4360   102      252.0000      Cuts: 34      718  24.43%
*     0+    0                          213.0000      190.4360           10.59%
      0     0      190.4566   108      213.0000      Cuts: 21      742  10.58%
      0     0      190.4836   106      213.0000   ZeroHalf: 6      762  10.57%
      0     0      190.4996   109      213.0000       Cuts: 8      896  10.56%
      0     0      190.4996   108      213.0000       Cuts: 6      966  10.56%
      0     0      190.5034   103      213.0000   ZeroHalf: 5     1114  10.56%
*     0+    0                          212.0000      190.5034           10.14%
      0     2      191.1850    96      212.0000      191.2729     1114   9.78%
Elapsed time = 0.45 sec. (223.73 ticks, tree = 0.01 MB)
    400   217      196.0433    84      212.0000      192.1349    15455   9.37%
   1066   837      194.3365    83      212.0000      192.9949    46312   8.96%
   2125  1634      204.7708    61      212.0000      193.8977    79334   8.54%
   2563  2144      193.6414    85      212.0000      194.3378   103542   8.33%
   2937   252        cutoff            212.0000      194.8663   114249   8.08%
   3980  1077      198.4457    54      212.0000      196.0000   139800   7.55%
.......
```

Balanced Team Assignment

. . . leads to successful conclusion, in about an hour

```
          Nodes                                         Cuts/
   Node    Left     Objective  IInf  Best Integer    Best Node     ItCnt     Gap
.......
 9667211  250649       cutoff             212.0000    210.6795  1.41e+08    0.62%
 9692898  226492       cutoff             212.0000    210.7083  1.41e+08    0.61%
Elapsed time = 4110.01 sec. (2381795.04 ticks, tree = 231.60 MB)
 9718729  201471       cutoff             212.0000    210.7384  1.41e+08    0.60%
 9745282  176469       cutoff             212.0000    210.7647  1.41e+08    0.58%
 9772348  151900     210.8483    28       212.0000    210.8000  1.41e+08    0.57%
 9799557  124671       cutoff             212.0000    210.8333  1.41e+08    0.55%
 9827583   95183       cutoff             212.0000    210.8765  1.41e+08    0.53%
 9856180   69947       cutoff             212.0000    210.9271  1.41e+08    0.51%
 9885302   43185       cutoff             212.0000    211.0000  1.42e+08    0.47%
 9911861   19735       cutoff             212.0000    211.0000  1.42e+08    0.47%

Mixed integer rounding cuts applied:  948
Zero-half cuts applied:  16
Lift and project cuts applied:  19
Gomory fractional cuts applied:  5

CPLEX 12.8.0.0: optimal integer solution; objective 212
141832373 MIP simplex iterations
9931504 branch-and-bound nodes
```

Integer Quadratic Objectives

General form

- Minimize $x^T Qx + qx$

Convex case

- Q positive semi-definite
- Test *numerically* using elimination on Q

Binary Convex

Sample model . . .

```
param n > 0;
param c {1..n} > 0;

var X {1..n} binary;

minimize Obj:
   (sum {j in 1..n} c[j]*X[j])^2;

subject to SumX: sum {j in 1..n} j * X[j] >= 50*n+3;
```

Binary Convex *(cont'd)*

CPLEX 12.5

```
ampl: solve;

.......

Cover cuts applied:  2
Zero-half cuts applied:  1

.......

Total (root+branch&cut) = 0.42 sec.

CPLEX 12.5.0: optimal integer solution within mipgap or absmipgap;
objective 29576.27517

286 MIP simplex iterations
102 branch-and-bound nodes
```

(n = 200)

Binary Convex *(cont'd)*

CPLEX 12.6

```
ampl: solve;

MIP Presolve added 39800 rows and 19900 columns.
Reduced MIP has 39801 rows, 20100 columns, and 79800 nonzeros.
Reduced MIP has 20100 binaries, 0 generals, and 0 indicators.

.......

Cover cuts applied:  8
Zero-half cuts applied:  5218
Gomory fractional cuts applied:  6

.......

Total (root+branch&cut) = 2112.63 sec.

CPLEX 12.6.0: optimal integer solution; objective 29576.27517

474330 MIP simplex iterations
294 branch-and-bound nodes
```

Binary Convex Strategies

Quadratic branch-and-bound (CPLEX 12.5)

- Solve a continuous QP at each node

Conversion to linear (CPLEX 12.6)

- Replace each objective term $x_i x_j$ by binary $y_{ij} \geq x_i + x_j - 1$
- Solve a larger continuous LP at each node

. . . option for 12.5 behavior added to 12.6.1

Binary Nonconvex

Sample model . . .

```
param n > 0;
param c {1..n} > 0;
param d {1..n} > 0;

var X {1..n} binary;
var Y {1..n} binary;

minimize Obj:
   (sum {i in 1..n} c[i]*X[i]) * (sum {j in 1..n} d[j]*Y[j]);

subject to SumX: sum {i in 1..n} j * X[i] >= 2*n+3;
subject to SumY: sum {j in 1..n} j * Y[j] >= 2*n+3;

subject to SumXY: sum {k in 1..n} (X[k] + Y[k]) = n;
```

Binary Nonconvex *(cont'd)*

CPLEX 12.5

```
ampl: solve;

Repairing indefinite Q in the objective.

. . . . . . .

Total (root+branch&cut) = 1264.34 sec.

CPLEX 12.5.0: optimal integer solution within mipgap or absmipgap;
objective 290.1853405

23890588 MIP simplex iterations
14092725 branch-and-bound nodes
```

(n = 50)

Binary Nonconvex *(cont'd)*

CPLEX 12.6

```
ampl: solve;

MIP Presolve added 5000 rows and 2500 columns.
Reduced MIP has 5003 rows, 2600 columns, and 10200 nonzeros.
Reduced MIP has 2600 binaries, 0 generals, and 0 indicators.

. . . . . . .

Total (root+branch&cut) = 6.05 sec.

CPLEX 12.6.0: optimal integer solution; objective 290.1853405

126643 MIP simplex iterations
1926 branch-and-bound nodes
```

Binary Nonconvex Strategies

Conversion to convex quadratic (CPLEX 12.5)

- Add $M_j(x_j^2 - x_j)$ to objective as needed to convexify
- Solve a continuous QP at each node

Conversion to linear (CPLEX 12.6)

- Replace each objective term $x_i x_j$ by binary $y_{ij} \geq x_i + x_j - 1$
- Solve a larger continuous LP at each node

. . . algorithms same as before

Binary × General Nonconvex

Reformulation of sample model . . .

```
param n > 0;
param c {1..n} > 0;
param d {1..n} > 0;

var X {1..n} binary;
var Y {1..n} binary;
var Ysum;

# minimize Obj:
#    (sum {i in 1..n} c[i]*X[i]) * (sum {j in 1..n} d[j]*Y[j]);

minimize Obj:
   (sum {i in 1..n} c[i]*X[i]) * Ysum;

subj to YsumDefn: Ysum = sum {j in 1..n} d[j]*Y[j];

subject to SumX: sum {i in 1..n} j * X[i] >= 2*n+3;
subject to SumY: sum {j in 1..n} j * Y[j] >= 2*n+3;

subject to SumXY: sum {k in 1..n} (X[k] + Y[k]) = n;
```

Binary × General Nonconvex *(cont'd)*

CPLEX 12.5

```
ampl: solve;
CPLEX 12.5.0: QP Hessian is not positive semi-definite.
```

Binary × General Nonconvex *(cont'd)*

CPLEX 12.6

```
ampl: solve;

MIP Presolve added 100 rows and 50 columns.
Reduced MIP has 104 rows, 151 columns, and 451 nonzeros.
Reduced MIP has 100 binaries, 0 generals, and 0 indicators.
.......
Total (root+branch&cut) = 0.17 sec.

CPLEX 12.6.0: optimal integer solution; objective 290.1853405

7850 MIP simplex iterations
1667 branch-and-bound nodes
```

Binary × General Nonconvex Strategies

Conversion to binary × general linear

- Replace sum of binaries by general $y_{\text{sum}} = \sum_{j=1}^{n} d_j y_j$
- Replace each objective term $x_i y_{\text{sum}}$ by
  $z_i \geq L x_i,\ z_i \geq y_{\text{sum}} - U(1 - x_i)$, where $L \leq y_{\text{sum}} \leq U$
- Introduce fewer but more complex variables, constraints

Many refinements and generalizations

- F. Glover and E. Woolsey, Further reduction of zero-one polynomial programming problems to zero-one linear programming problems (1973)
- F. Glover, Improved linear integer programming formulations of nonlinear integer problems. *Management Science* 22 (1975) 455-460.
- M. Oral and O. Kettani, A linearization procedure for quadratic and cubic mixed-integer problems. *Operations Research* 40 (1992) S109-S116.
- W.P. Adams and R.J. Forrester, A simple recipe for concise mixed 0-1 linearizations. *Operations Research Letters* 33 (2005) 55-61.

General Nonconvex

Neither integer variable is binary

```
param n > 0;
param c {1..n} > 0;
param d {1..n} > 0;
var X {1..n} integer >= 0, <= 2;
var Y {1..n} integer >= 0, <= 2;

minimize Obj:
   (sum {j in 1..n} c[j]*X[j]) * (sum {j in 1..n} d[j]*Y[j]);

subject to SumX: sum {i in 1..n} j * X[i] >= 2*n+3;
subject to SumY: sum {j in 1..n} j * Y[j] >= 2*n+3;
subject to SumXY: sum {k in 1..n} (X[k] + Y[k]) = n;
```

General Nonconvex *(cont'd)*

CPLEX default setting

```
ampl: solve;
CPLEX 12.6.3: QP Hessian is not positive semi-definite.
```

General Nonconvex *(cont'd)*

CPLEX setting to request nonconvex solve

```
ampl: solve;

CPLEX 12.6.3.0: reqconvex 3
mipdisplay 2
mipinterval 1000

Reduced MIQP has 3 rows, 440 columns, and 80 nonzeros.
Reduced MIQP has 0 binaries, 40 generals, 0 SOSs, and 0 indicators.
Reduced MIQP objective Q matrix has 800 nonzeros.

.......

Total (root+branch&cut) =  758.41 sec.

CPLEX 12.6.3: optimal integer solution within mipgap or absmipgap;
   objective 69.30360303

8447893 MIP simplex iterations
637937 branch-and-bound nodes

absmipgap = 0.00675848, relmipgap = 9.75199e-05
```

(n = 20)

General Nonconvex *(cont'd)*

BARON (general nonlinear global solver)

```
ampl: solve;

BARON 16.7.29 (2016.07.29)

This BARON run may utilize the following subsolver(s)
For LP/MIP: CLP/CBC
For NLP: IPOPT, FILTERSD

.......

Wall clock time:                    50.69
Total CPU time used:                29.92

BARON 16.7.29 (2016.07.29): 708 iterations,
   optimal within tolerances.

Objective 69.30360303
```

General Nonconvex *(cont'd)*

BARON using CPLEX

```
ampl: solve;

BARON 16.7.29 (2016.07.29): lpsolver cplex

This BARON run may utilize the following subsolver(s)
For LP/MIP: ILOG CPLEX
For NLP: IPOPT, FILTERSD

.......

Wall clock time:                    0.41
Total CPU time used:                0.38

BARON 16.7.29 (2016.07.29): 15 iterations,
   optimal within tolerances.

Objective 69.30360303
```

General Nonconvex *(cont'd)*

Knitro

```
ampl: solve;

Times (seconds):
Input =  0
Solve =  0.046875
Output = 0

Knitro 10.3.0: Locally optimal solution.
objective 69.30360303; integrality gap -29.8

7 nodes; 14 subproblem solves; feasibility error 0
0 iterations; 161 function evaluations
```

General Nonconvex Strategies

Nonconvex extension to quadratic MIP solver

Global nonlinear solver

- Using built-in open source solvers
- Using commercial solvers
  - For linear MIP subproblems
  - For nonlinear subproblems

Local nonlinear solver

- Solving once from default initial values
- Solving many times from generated initial values

Constrained Roll Cutting

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
  - ∗ . . . *and also* at most 10% waste per pattern

Constrained Roll Cutting

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

Constrained Roll Cutting (CPLEX)

Pattern generation

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

Pattern enumeration

- 54,508 non-dominated patterns
- 34 rolls solving IP using enumerated patterns
- 778 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

Constrained Roll Cutting (Gurobi)

Pattern generation

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

Pattern enumeration

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

No overage: change >= to =

- 34 rolls solving IP using enumerated patterns
- 1198 branch-and-bound nodes

. . . all subsequent tests include this condition

Roll Ordering with Side Constraints

At most 2% waste in any pattern

- 16,362 non-dominated patterns
- CPLEX: No feasible solution???

```
CPLEX 12.8.0.0: mipdisplay 2
Reduced MIP has 11 rows, 16280 columns, and 85544 nonzeros.
Reduced MIP has 1619 binaries, 14661 generals, 0 SOSs, and 0 indicators.
      Nodes                                          Cuts/
 Node  Left     Objective       IInf  Best Integer    Best Bound    ItCnt Gap
    0     0      33.7825         11                     33.7825      130
    0     0      33.8056         11                    Cuts: 14      138
.......
 2975980 2609535       33.8039     6                    33.8889  7368912
 2980187 2612999       33.7831     8                    33.8889  7378677
Elapsed time = 5754.84 sec. (2808509.95 ticks, tree = 36829.29 MB)
Nodefile size = 34781.51 MB (3761.09 MB after compression)
 2984257 2614892       33.7896     6                    33.8889  7383663
 2988310 2617413       33.7913     6                    33.8889  7391068
 2992415 2621640       33.8100     7                    33.8889  7403124
 2996658 2627718       33.8039     6                    33.8889  7420016
 3000749 2630289    infeasible                          33.8889  7426966
```

Roll Ordering with Side Constraints

At most 2% waste in any pattern

- 16,362 non-dominated patterns
- Gurobi: Feasible solution eventually . . .

```
Gurobi 7.5.0: outlev 1
Optimize a model with 11 rows, 16280 columns and 85544 nonzeros
Variable types: 0 continuous, 16280 integer (1619 binary)
 Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0   33.78247    0    9          -   33.78247     -     -    0s
     0     0   33.78247    0   13          -   33.78247     -     -    1s
.......
 63086 42651   33.79214  492    9          -   33.78355     -   3.5  170s
 65932 45329   33.80299  513    8          -   33.78355     -   3.4  177s
 67710 46954   33.80000  313    5          -   33.78355     -   3.4  180s
 70780 49741   33.80187  488    9          -   33.78355     -   3.4  185s
H71489    13                     34.0000000   33.78355  0.64%   3.4  186s
.......
Gurobi 7.5.0: optimal solution; objective 34
245585 simplex iterations
71587 branch-and-cut nodes
```

Roll Ordering with Side Constraints

At most 2% waste in any pattern

- Minimize total cut rolls instead

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

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

- 296 cut rolls (= 296 orders) in optimal solution
- 34 raw rolls in that solution

Solution times

	CPLEX	Gurobi
RawRollsCut (=)	> 5000	187
OrderedWidthsCut (=)	1	19
OrderedWidthsCut (>=)	261	< 1

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) in optimal solution
- 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) in optimal solution
- 37 raw rolls in that solution
- 2 seconds solution time, solved at root node

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) in optimal solution
- 37 raw rolls in that solution
- < 1 second solution time, solved at root node