

Approaches to Near-Optimally Solving Mixed-Integer Programs

Robert Fourer

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

AMPL Optimization LLC

`4er@northwestern.edu` – `4er@ampl.com`

What a Pivot — Workshop Honoring Bob Bixby's 65th Birthday
Erlangen, Germany, 26-28 September 2010

Outline

Breaking up

- Work scheduling
- Balanced dinner assignment
- Progressive party assignment

Cutting off

- Paint chip cutting
- Balanced team assignment

Throwing out

- Roll cutting

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?

Work Scheduling

Model using zero-one variables

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

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

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

subject to Least_Use1 {j in SCHEDES}:
    Work[j] >= least_assign * Use[j];

subject to Least_Use2 {j in SCHEDES}:
    Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
```

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
```

Work Scheduling

Branch & bound

least_assign	nodes	iterations	seconds
16	1345687	10113022	115
17			> 30000
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

Optimum of relaxation is always 265.6

≤ 16 : optimum of MIP is 266

≥ 20 : optimum is integral with **Work** variables relaxed

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 least_assign := 17;
ampl: option solver gurobi;

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

ampl: solve;

Gurobi 1.1.3: optimal solution; objective 266.5
7898786 simplex iterations;
1556653 branch-and-cut nodes

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

ampl: solve;

Gurobi 1.1.3: optimal solution; objective 267
4 simplex iterations;
0 branch-and-cut nodes
```

Work Scheduling

Two-step approach

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
21	76873	299338	7
22	61727	259012	5
23	111721	392251	8
24	82152	292187	6

In this example . . .

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

. . . hence optimal

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

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 default "";

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

        # 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
```

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

(variables and objective)

```
var Assign {i in PEOPLE, j in 1..numberGrps} binary;
    # assignments of people to groups

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);
    # min/max of each type over all groups

minimize TotalVariation:
    sum {k in CATEG, t in TYPES[k]}
        (MaxType[k,t] - MinType[k,t]);
    # Sum of variation over all types
```

Breaking Up 2

(constraints)

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

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

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

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

        # Defining constraints for
        # min and max type variables
```

Solving for Minimum Variation

1054 variables:

1000 binary variables

54 linear variables

560 constraints, all linear; 12200 nonzeros

1 linear objective; 54 nonzeros.

CPLEX 3.0:

Nodes		Objective	IInf	Best Integer	Cuts/
Node	Left				Best Node
0	0	17.0000	299		17.0000
10	10	17.0000	322		17.0000
20	20	17.0000	332		17.0000
30	30	17.0000	328		17.0000
40	40	17.0000	329		17.0000
50	50	17.0000	329		17.0000
60	60	17.0000	339		17.0000
70	70	17.0000	344		17.0000
80	80	17.0000	342		17.0000

.....

Breaking Up 2

(continued)

	Nodes					Cuts/
	Node	Left	Objective	IInf	Best Integer	Best Node
	250	250	43.6818	74		17.0000
	260	260	46.5000	58		17.0000
*	265	263	47.0000	0	47.0000	17.0000
	270	266	17.0000	314	47.0000	17.0000
	280	276	17.0000	351	47.0000	17.0000
	290	286	17.0000	340	47.0000	17.0000
	300	296	17.0000	337	47.0000	17.0000
	310	306	17.0000	341	47.0000	17.0000
					
	630	609	21.5208	243	47.0000	17.0000
	640	618	23.3028	244	47.0000	17.0000
	650	626	17.3796	269	47.0000	17.0000
	660	636	17.7981	271	47.0000	17.0000
*	666	440	19.0000	0	19.0000	17.0000
	670	440	17.0000	147	19.0000	17.0000
	680	446	17.0714	213	19.0000	17.0000
	690	454	17.5000	186	19.0000	17.0000

Breaking Up 2

(concluded)

	Nodes					Cuts/
Node	Left	Objective	IInf	Best	Integer	Best Node
700	461	17.1364	268		19.0000	17.0000
710	468	17.3117	267		19.0000	17.0000
720	475	17.0000	211		19.0000	17.0000
730	484	17.2652	226		19.0000	17.0000
740	490	17.0000	106		19.0000	17.0000
750	497	17.0000	24		19.0000	17.0000
* 752	0	17.0000	0		17.0000	

Times (seconds):
Input = 0.266667
Solve = 864.733
Output = 0.166667

CPLEX 3.0: optimal integer solution; objective 17
45621 simplex iterations
752 branch-and-bound nodes

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

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)

```
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

(variables, objective, assign 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;
```

(constraints to define “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];
```

(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]);
```

(constraints to rule 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];
```

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

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

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.

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

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

Meeting 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 ;
```

Direct Approach

```
ampl: solve;
```

```
Presolve eliminates 88078 constraints and 2892 variables.
```

```
Adjusted problem:
```

```
12648 variables:
```

```
    7482 binary variables
```

```
    5166 linear variables
```

```
131990 constraints, all linear; 410546 nonzeros
```

```
1 linear objective; 36 no
```

```
MIP Presolve eliminated 14317 rows and 721 columns.
```

```
MIP Presolve modified 6762 coefficients.
```

```
Reduced MIP has 117674 rows, 11928 columns, and 374126 nonzeros.
```

```
Reduced MIP has 7482 binaries, 0 generals, 0 SOSs, and 0 indicators.
```

```
Probing time =    0.03 sec.
```

```
MIP Presolve eliminated 978 rows and 0 columns.
```

```
MIP Presolve modified 1956 coefficients.
```

```
Reduced MIP has 116696 rows, 11928 columns, and 371192 nonzeros.
```

```
Reduced MIP has 7482 binaries, 0 generals, 0 SOSs, and 0 indicators.
```

```
Probing time =    0.03 sec.
```

```
Clique table members: 7283.
```

```
MIP emphasis: integer feasibility.
```

```
MIP search method: dynamic search.
```

```
Parallel mode: deterministic, using up to 8 threads.
```

```
Root relaxation solution time =    6.47 sec. nzeros.
```

Direct Approach (*branching*)

Nodes		Objective	IInf	Best Integer	Cuts/		Gap
Node	Left				Best Node	ItCnt	
0	0	12.2000	484		12.2000	6212	
0	0	12.2000	312		Cuts: 279	9730	
0	0	12.2000	332		Cuts: 643	13688	
0	0	12.2000	256		Cuts: 107	16645	
0	2	12.2000	150		12.2000	16645	
40	42	12.2222	280		12.2000	173207	
80	82	12.3333	253		12.2000	221027	
120	122	13.0000	285		12.2000	256891	
160	162	13.0000	216		12.2000	295682	
*	192+			14.0000	12.2000	334618	12.86%
	200	13.0000	167	14.0000	12.2000	343192	12.86%
	240	13.0000	165	14.0000	12.2000	366033	12.86%
	280	13.0000	274	14.0000	12.2000	379204	12.86%
	320	13.0000	69	14.0000	12.2000	393411	12.86%
	360	13.0000	65	14.0000	12.2000	404419	12.86%
*	367+			13.0000	12.2000	406283	6.15%
	380	13.0000	147	13.0000	12.2000	410635	6.15%
	400	13.0000	8	13.0000	12.2000	415294	6.15%
						

Direct Approach(*results*)

```
Clique cuts applied: 9
Cover cuts applied: 263
Implied bound cuts applied: 56
Zero-half cuts applied: 15

Root node processing (before b&c):
  Real time          = 234.06
Parallel b&c, 8 threads:
  Real time          = 377.09
  Sync time (average) = 38.18
  Wait time (average) = 168.18
                        -----
Total (root+branch&cut) = 611.15 sec.

Times (seconds):
Input = 0.156
Solve = 611.963
Output = 0.125

CPLEX 12.2.0.0: optimal integer solution; objective 13
418678 MIP simplex iterations
420 branch-and-bound nodes
```

. . . results highly variable across settings and solvers

Multi-Step Approach

Determine hosts

- solve 1-period problem
- fix hosts
- fix 1st-period visits

Determine visits: for $t = 2, 3, \dots$

- solve t^{th} -period problem
- fix t^{th} period visits

... hosts & previous $t-1$ periods already fixed

Multi-Step Script

```
model partyKA.mod;
data partyKA.dat;

option solver cplexamp;
option cplex_options 'branch 1 startalg 1 subalg 1 mipemphasis 1 timing 1';
option show_stats 1;

# -----
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 partyKB.run
```

```
Reduced MIP has 983 rows, 1272 columns, and 4364 nonzeros.
```

```
Reduced MIP has 1272 binaries, 0 generals, 0 SOSs, and 0 indicators.
```

```
Solve = 0.249
```

```
CPLEX 12.2.0.0: optimal integer solution; objective 13
```

```
189 MIP simplex iterations
```

```
0 branch-and-bound nodes
```

```
Reduced MIP has 169 rows, 342 columns, and 997 nonzeros.
```

```
Reduced MIP has 342 binaries, 0 generals, 0 SOSs, and 0 indicators.
```

```
Solve = 0.063
```

```
CPLEX 12.2.0.0: optimal integer solution; objective 13
```

```
76 MIP simplex iterations
```

```
0 branch-and-bound nodes
```

```
Reduced MIP has 258 rows, 313 columns, and 1162 nonzeros.
```

```
Reduced MIP has 313 binaries, 0 generals, 0 SOSs, and 0 indicators.
```

```
Solve = 0.062
```

```
CPLEX 12.2.0.0: optimal integer solution; objective 13
```

```
77 MIP simplex iterations
```

```
0 branch-and-bound nodes
```

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

Reduced MIP has 319 rows, 284 columns, and 1284 nonzeros.

Reduced MIP has 284 binaries, 0 generals, 0 SOSs, and 0 indicators.

Solve = 0.093

CPLEX 12.2.0.0: optimal integer solution; objective 13

64 MIP simplex iterations

0 branch-and-bound nodes

Reduced MIP has 328 rows, 255 columns, and 1289 nonzeros.

Reduced MIP has 255 binaries, 0 generals, 0 SOSs, and 0 indicators.

Solve = 0.047

CPLEX 12.2.0.0: optimal integer solution; objective 13

65 MIP simplex iterations

0 branch-and-bound nodes

Reduced MIP has 327 rows, 226 columns, and 1264 nonzeros.

Reduced MIP has 226 binaries, 0 generals, 0 SOSs, and 0 indicators.

Solve = 0.031

CPLEX 12.2.0.0: optimal integer solution; objective 13

58 MIP simplex iterations

0 branch-and-bound nodes

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

Reduced MIP has 281 rows, 197 columns, and 1103 nonzeros.

Reduced MIP has 197 binaries, 0 generals, 0 SOSs, and 0 indicators.

Solve = 0.094

CPLEX 12.2.0.0: optimal integer solution; objective 13

69 MIP simplex iterations

0 branch-and-bound nodes

Reduced MIP has 232 rows, 168 columns, and 914 nonzeros.

Reduced MIP has 168 binaries, 0 generals, 0 SOSs, and 0 indicators.

Solve = 0.094

CPLEX 12.2.0.0: optimal integer solution; objective 13

126 MIP simplex iterations

0 branch-and-bound nodes

Reduced MIP has 174 rows, 133 columns, and 672 nonzeros.

Reduced MIP has 133 binaries, 0 generals, 0 SOSs, and 0 indicators.

Solve = 0.187

CPLEX 12.2.0.0: optimal integer solution; objective 13

2009 MIP simplex iterations

50 branch-and-bound nodes

Breaking Up 3

Multi-Step Run (*no period 10*)

Reduced MIP has 120 rows, 102 columns, and 469 nonzeros.

Reduced MIP has 102 binaries, 0 generals, 0 SOSs, and 0 indicators.

Solve = 0.062

CPLEX 12.2.0.0: **integer infeasible.**

75 MIP simplex iterations

0 branch-and-bound 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

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

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

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 *(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)

Results *(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)

Balanced Team Assignment

Same idea, different formulation

- Class example of where branch-and-bound fails
 - * steadily growing tree
 - * terrible initial lower bound
 - * gap scarcely grows

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

Balanced Assignment

Test data

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

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.

Balanced Assignment

Active start . . .

	Nodes				Cuts/			
	Node	Left	Objective	IInf	Best Integer	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%
							

Balanced Assignment

... bogs down completely

Node	Nodes Left	Objective	IInf	Best Integer	Cuts/ 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.							

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]);
```

- $\text{maxOverlap}[i]$ must be \geq greatest overlap possible
- Smaller values give stronger b&b lower bounds
 - * theoretically correct: $4 * (\text{maxInGrp}-1) \rightarrow 0.0$
 - * empirically justified: $1 * (\text{maxInGrp}-1) \rightarrow 156.8$

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
 - * $\text{floor}(\text{card}(\text{PEOPLE})/\text{numberGrps}) - 1$
 $\text{ceil}(\text{card}(\text{PEOPLE})/\text{numberGrps}) + 1 \rightarrow 156.8$
 - * $\text{floor}(\text{card}(\text{PEOPLE})/\text{numberGrps})$
 $\text{ceil}(\text{card}(\text{PEOPLE})/\text{numberGrps}) \rightarrow 177.6$

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;
```

- Specify exact sizes of all groups
- Exact sizes give stronger b&b lower bounds
 - * min & max sizes for every g → 177.6
 - * exact sizes → 183.36

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%
.....								

Balanced Assignment

Much more promising start . . .

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

Balanced Assignment

... leads to successful conclusion

Node	Nodes Left	Objective	IInf	Best Integer	Cuts/ 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							

Throwing Out

Roll Cutting

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

- Solve the pattern-choice MIP using either of . . .
 - * patterns generated by the Gilmore-Gomory method (for solving the relaxation)
 - * all nondominated patterns

Throwing Out

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;
```

Throwing Out

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;
}
}
```

Throwing Out

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

*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