Assigning People in Practice

Robert Fourer

Department of Industrial Engineering
and Management Sciences

Northwestern University
Evanston, IL 60208-3119, U.S.A.

4er@iems.northwestern.edu

CORS/INFORMS International Meeting
Banff, May 16-19, 2004

WC02: Optimization Modelling in Practice II

Outline

Classical assignment

- Assigning professor to offices
- Adjusting the results

Modified assignment

- Assigning students to project groups
- Modeling the complications

"Balanced" assignment

- Tests of formulations using sample data
- Scaling up to full data

Classical Assignment

Given
  P, a set of people
  Q, a set of *places*
  c_{ij}, cost of assigning person i to place j

Define
  $X_{ij} = 1$ if person i is assigned to place j
  $\quad = 0$ otherwise

Minimize $\quad \sum_{p \in P} \sum_{q \in Q} c_{pq} X_{pq}$

Subject to $\quad \sum_{q \in Q} X_{pq} = 1$, for each $p \in P$
$\quad \sum_{p \in P} X_{pq} \leq 1$, for each $q \in Q$
$\quad X_{pq} \geq 0$, for each $p \in P$ and $q \in Q$

. . . same, but in AMPL

```
set P;  # people
set Q;  # places

param c {P,Q} > 0;

var X {P,Q} binary;

minimize Z: sum {p in P} sum {q in Q} c[p,q] * X[p,q];

subject to P1 {p in P}: sum {q in Q} X[p,q] = 1;
subject to Q1 {q in Q}: sum {p in P} X[p,q] <= 1;
```

. . . same, but more readable

```
set PEOPLE;
set PLACES;

param pref {PEOPLE,PLACES} > 0;

var Assign {PEOPLE,PLACES} binary;

minimize TotalPref:
  sum {p in PEOPLE} sum {q in PLACES} pref[p,q] * Assign[p,q];

subj to OnePlacePerPerson {p in PEOPLE}:
  sum {q in PLACES} Assign[p,q] = 1;

subj to OnePersonPerPlace {q in PLACES}:
  sum {p in PEOPLE} Assign[p,q] <= 1;
```

Data for Professors and Offices

```
set PEOPLE := Bassok Coullard Frey Hazen Hopp Hurter
              Jones Mehrotra Rieders Rath Rubenstein Spearman
              Sun Tamhane Thompson Zazanis ;

set PLACES := 1021 1049 1053 1055 1083 1087
              2009 2019 2053 2083 2087
              3021 3041 3083 3087
              4083 4087 ;

param pref:  1021 1049 1053 1055 1083 1087 2009 2019 2053 2083 2087 :=
  Bassok       7    7    7    7    7    7    7    7    7    6    5
  Coullard    11   14   13   12   16   15   10   11    9    8    7
  Frey         4    4    4    3    4    4    4    4    1    4    4
  Hazen       17   14   13   12   15   16    6   11    9    7    8
  Hopp        15   16   17    4   10   11    5   12   13    8    9
  Hurter      17   15   14   16   11   10    4   13    9    7    8
  Jones        5    4   14   15   16   17    1   11   10   12   13
  Mehrotra    17   14   15    9    7    8   10   11   12    3    4
  Rieders     12   17   16   15   14   13    7    8   11   10    9
  .......
```

A First Assignment

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

ampl: solve;

MINOS 5.5: optimal solution found.
128 iterations, objective 49

ampl: display Assign;

Assign [*,*] (tr)
# $2 = Coullard
# $6 = Hurter

:     Bassok  '$2'        Frey             Hazen     Hopp  '$6'      Jones  :=

1021     0      0     1                 0             0     0    -6.38388e-17
1049     0      0     0                 0             0     0     0
1053     0      0     0                 0             0     0     0
1055     0      0     0                 0             1     0     0
1083     0      0    -7.20534e-17       0             0     0     0
1087     0      0     8.21457e-18       0             0     0     0
2009     0      0     0                 0             0     0     0
2019     1      0     0                 0             0     0     0
2053     0      0    -5.56242e-17       0             0     0     0   .......
```

(displayed legibly)

```
ampl: option display_1col 10000, omit_zero_rows 1;
ampl: option display_eps .000001;

ampl: display {p in PEOPLE, q in PLACES} pref[p,q] * Assign[p,q];
pref[p,q]*Assign[p,q] :=
Bassok     2019   7
Coullard   4083   2
Frey       1021   4
Hazen      3083   3
Hopp       1055   4
Hurter     3041   1
Jones      3021   3
Mehrotra   2083   3
Rath       1053   2
Rieders    3087   3
Rubenstein 1049   1
Spearman   2009   1
Sun        4087   1
Tamhane    1087   7
Thompson   2053   2
Zazanis    2087   5
```

A Seniority-Weighted Assignment

```
param base >= 1;
param weight {PEOPLE} > 0;
param pref {PEOPLE,PLACES} > 0;

var Assign {PEOPLE,PLACES} binary;

minimize TotalPref:
  sum {p in PEOPLE} base^weight[p] *
     sum {q in PLACES} pref[p,q] * Assign[p,q];
```

```
param base := 10 ;

param weight :=
    Bassok 1        Hopp 3         Rath 4         Sun 2
  Coullard 3      Hurter 4      Rieders 1     Tamhane 4
      Frey 4       Jones 4   Rubenstein 4    Thompson 4
     Hazen 3    Mehrotra 2     Spearman 2     Zazanis 2 ;
```

(results)

```
MINOS 5.5: optimal solution found.
128 iterations, objective 102330

ampl: display {p in PEOPLE, q in PLACES} pref[p,q] * Assign[p,q];

pref[p,q]*Assign[p,q] :=

Bassok      1087   7
Coullard    3087   3
Frey        2053   1
Hazen       3083   3
Hopp        1055   4
Hurter      4083   2
Jones       2009   1
Mehrotra    1083   7
Rath        1053   2
Rieders     3021   6
Rubenstein  1049   1
Spearman    2083   2
Sun         2019   8
Tamhane     4087   1
Thompson    3041   1
Zazanis     2087   5
```

A Politically-Sensitive Assignment

```
set GIVEN within {PEOPLE,PLACES};
.......
subj to PoliticalDecisions {(p,q) in GIVEN}:
  Assign[p,q] = 1;
```

```
set given := (Rubenstein,1049) (Rath,1053) (Frey,2019) ;
```

A More Equitable Assignment

```
param worst integer <= card {PLACES};

.......

subj to NotTooAwful
   {p in PEOPLE, q in PLACES: pref[p,q] > worst}:
      Assign[p,q] = 0;
```

```
ampl: let worst := 7;

ampl: solve;

MINOS 5.5: optimal solution found.
46 iterations, objective 130830

ampl: let worst := 6;

ampl: solve;

MINOS 5.5: infeasible problem.
4 iterations
```

Observation #1

Use a small assignment model
to generate assignments

Then go with the one you prefer

Observation #2

Generate the assignments for yourself

Announce only the assignment you choose

Modified Assignment

Given

- Students and projects
- Preferences of students for projects
- Subgroups of students wanting the same project
- List of students who have cars

Assign

- 3 or 4 students per project
- At least one car per project
- Students in each subgroup to the same project

. . . ***with preference to students not in subgroups***

Student Data

```
set STU ordered;
param car {STU} binary;

param ngroup integer >= 0;
set GRP = 1..ngroup;

set MEM {GRP} ordered by STU;
   check {g1 in GRP, g2 in g1+1..ngroup}:
      card (MEM[g1] inter MEM[g2]) = 0;

set SAMEGRP = union {g in GRP}
   {s1 in MEM[g], s2 in MEM[g]: ord(s1) < ord(s2)};
```

Project Data

```
set PRJ;
param cars_needed {PRJ} integer >= 0;
param min_team {PRJ} integer >= 0;
param max_team {p in PRJ} integer >= min_team[p];

param rank {STU,PRJ} integer >= 0, <= card {PRJ};
   check {(s1,s2) in SAMEGRP, p in PRJ}:
      rank[s1,p] = rank[s2,p];
```

Objective

```
var Assign {STU,PRJ} binary;

set GROUPED = union {g in GRP} MEM[g];
param group_weight >= 1;

minimize Total_Rank:
   sum {s in STU, p in PRJ} rank[s,p] * Assign[s,p] *
      (if s in GROUPED then group_weight else 1);
```

General Constraints

```
subject to Assign_Students {s in STU}:
   sum {p in PRJ} Assign[s,p] = 1;

subject to Assign_Projects {p in PRJ}:
   min_team[p] <= sum {s in STU} Assign[s,p] <= max_team[p];

subject to Enough_Cars {p in PRJ}:
   sum {s in STU} car[s] * Assign[s,p] >= cars_needed[p];

subject to Preserve_Groups {(s1,s2) in SAMEGRP, p in PRJ}:
   Assign[s1,p] = Assign[s2,p];
```

Ad Hoc Constraints

```
param cutoff >= 1, <= card {PRJ};

subject to Not_Too_Bad
      {s in STU, p in PRJ: rank[s,p] > cutoff}:
   Assign[s,p] = 0;

set PRJ_PREF within {STU,PRJ};

subject to Project_Preference {(s,p) in PRJ_PREF}:
   Assign[s,p] = 1;
```

Project and Student Data

```
param: PRJ:   cars_needed min_team max_team :=

    "Ameritech"   1          4        4
    "DSC"         1          4        4
    "Motorola"    1          4        4
    "NMH"         1          4        4
    "S&C Elec"    1          4        4
    "TreeHouse"   1          4        4
    "UPS"         1          4        4 ;

param: STU: car :=

   Bhandari_Elsa 0
   Black_Andrew 1
   Croke_Michael 0
   Ellis_Mary_Beth 1
   Fernandez_Jason 0
   Friedlander_Jeffrey 1
   Gambell_Anthony 1
   Iwase_Yoshinori 0
   Katen_Philip 1
   .......
```

Subgroup and Rank Data

```
param ngroup := 7 ;

set MEM[1] := Bhandari_Elsa Vargas_Lorena Wise_David ;
set MEM[2] := Friedland_Jeffrey Katen_Philip Kemp_Charles Pain_Lucas ;
set MEM[3] := Ellis_Mary_Beth Xu_Ping ;
set MEM[4] := Kim_Linda Pan_Shaio-Tien Subudhayan_Suppachok Lee_Danny ;
set MEM[5] := Gambell_Anthony McCune_Christopher McCune_Jason ;
set MEM[6] := Kim_Rita Black_Andrew Shemluck_Matt Fernandez_Jason ;
set MEM[7] := Sit_Danny Wang_Jensen ;

param rank:
    "Ameritech" "DSC" "Motorola" "NMH" "S&C Elec" "TreeHouse" "UPS" :=
  Bhandari_Elsa              1 4 5 6 2 7 3
  Black_Andrew               7 3 2 6 4 5 1
  Croke_Michael              5 1 2 6 3 7 4
  Ellis_Mary_Beth            7 2 1 3 5 6 4
  Fernandez_Jason            7 3 2 6 4 5 1
  Friedlander_Jeffrey        7 2 5 3 4 1 6
  Gambell_Anthony            1 7 6 3 4 2 5
  Iwase_Yoshinori            4 5 1 7 2 6 3
  .......
```

Miscellaneous Data

```
param group_weight 3 ;

param cutoff := 4 ;

set PRJ_PREF := "McCune_Christopher" Ameritech ;
```

Solution

```
ampl: option display_1col 10000, omit_zero_rows 1;
ampl: option display_eps .000001;

ampl: solve;
MINOS 5.5: optimal solution found.
13 iterations, objective 101

ampl: display {p in PRJ, s in STU} Assign[s,p];

Assign[s,p] :=

Ameritech  Bhandari_Elsa           0.333333
Ameritech  Gambell_Anthony         1
Ameritech  McCune_Christopher      1
Ameritech  McCune_Jason            1
Ameritech  Vargas_Lorena           0.333333
Ameritech  Wise_David              0.333333
DSC        Bhandari_Elsa           0.666667
DSC        Black_Andrew            0.25
DSC        Croke_Michael           1
DSC        Fernandez_Jason         0.25
.......
```

(with integer variables)

```
CPLEX 9.0.0: optimal integer solution; objective 116
22 MIP simplex iterations
0 branch-and-bound nodes

ampl: display {p in PRJ, s in STU} rank[s,p] * Assign[s,p];
rank[s,p]*Assign[s,p] :=
Ameritech  Gambell_Anthony         1
Ameritech  Iwase_Yoshinori         4
Ameritech  McCune_Christopher      1
Ameritech  McCune_Jason            1
DSC        Bhandari_Elsa           4
DSC        Croke_Michael           1
DSC        Vargas_Lorena           4
DSC        Wise_David              4
NMH        King_Nancy              1
NMH        Mehawich_Michael        1
NMH        Starr_Cathy             1
NMH        Terrell_Eric            3
.......
```

Observation #3

Assignment problems
are seldom linear programs

They require
discrete optimization technologies

Observation #4

Assignment models
make intensive use of sets

Their modeling language formulations
make extensive use of set features

“Balanced” 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,
- but no one is unduly “isolated”

Plan of Attack

Year 1

- Dump it on a (human) database administrator
- Apply some *ad hoc* heuristics, by hand

Year 2

- Hire a consultant (me), to:
  - *build some optimization models*
  - *test simple models on small subsets of data*
  - *scale up to more complex models on the full data*

Year 3, 4, 5, . . .

- Re-run with new complications

Minimum “Sameness” 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

set SAMETYPE = {i1 in PEOPLE, i2 in PEOPLE diff {i1},
   k in CATEG: type[i1,k] = type[i2,k]};

              # set of triples (i1,i2,k) such that individuals
              # i1 and i2 have the same type in category k

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

              # number of groups; bounds on size of groups
```

(quadratic objective)

```
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

minimize TotalSameness:
   sum {(i1,i2,k) in SAMETYPE, j in 1..numberGrps}
      Assign[i1,j] * Assign[i2,j];

              # Product of variables is 1 iff both are 1

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

              # Each person assigned to one group

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

              # Each group has an acceptable size
```

(linearized objectives)

Simple Linearization

```
minimize TotalSameness:
   sum {j in GRP} sum {(i1,i2,k) in SAMETYPE} Same[i1,i2,j];

subj to SameDefn
      {i1 in PEOPLE, i2 in PEOPLE, j in 1..numberGrps}:
   Same[i1,i2,j] >= Assign[i1,j] + Assign[i2,j] - 1;
```

Concise Linearization

```
minimize TotalSameness:
   sum {j in GRP} sum {i in PEOPLE} Sameness[i,j];

subj to SamenessDefn {i in PEOPLE, j in GRP}:
   Sameness[i,j] >= sum {(i,i2,k) in SAMETYPE} Assign[i2,j]
      - maxSameness * (1 - Assign[i,j]);
```

Solving as Continuous Quadratic

100 people, 10 groups

```
ampl: solve;

1000 variables, all nonlinear
110 constraints, all linear; 2000 nonzeros
1 nonlinear objective; 1000 linear nonzeros.

MINOS 5.4: ignoring integrality of 1000 variables

 MINOS times:
 read:      11.35
 solve:    279.73       excluding minos setup: 279.67
 write:      0.02
 total:    291.10

MINOS 5.4: optimal solution found.
349 iterations, objective 1744
```

. . . all variables turn out integer !!!

Solving as Continuous Quadratic

100 people, 10 groups (more recent run)

```
ampl: solve;

1000 variables, all nonlinear
110 constraints, all linear; 2000 nonzeros
1 nonlinear objective; 1000 linear nonzeros.

MINOS 5.5: ignoring integrality of 1000 variables

 MINOS times:
 read:       0.29
 solve:      3.10      excluding minos setup: 3.10
 write:      0.00
 total:      3.39

MINOS 5.5: optimal solution found.
279 iterations, objective 1714
```

. . . all variables turn out integer !!!

Solving as Integer Quadratic

```
ampl: solve;

1000 variables, all nonlinear
110 constraints, all linear; 2000 nonzeros
1 nonlinear objective; 1000 nonzeros.

.......

  73900 73196   1573.1903  383   1724.0000   1343.6630   454152  22.06%
  74000 73296   1695.0051  119   1724.0000   1343.6630   455487  22.06%
* 74000+72612                0   1720.0000   1343.6630   455487  21.88%

Times (seconds):
Input =  0.981
Solve =  6458.19
Output = 0.411

CPLEX 9.0.0: feasible integer solution; objective 1720
455487 MIP simplex iterations
74000 branch-and-bound nodes
```

. . . is this convex ???

Solving the Simple Linearization

```
ampl: solve;

96520 variables:
        1000 binary variables
        95520 linear variables
95630 constraints, all linear; 288560 nonzeros
1 linear objective; 95520 nonzeros.

CPLEX 3.0:

.......
```

. . . wait forever with no solution !!!

Solving the Concise Linearization

```
ampl: solve;

2000 variables:
        1000 binary variables
        1000 linear variables
1110 constraints, all linear; 99520 nonzeros
1 linear objective; 1000 nonzeros.

CPLEX 3.0:

No MIP presolve or aggregator reductions.
Elapsed time =   30.30 sec.

.......
```

. . . ***now branch-and-bound begins*** →

(continued)

	Node	Nodes Left	Objective	IInf	Best Integer	Cuts/ Best Node
	0	0	0.0000	274		0.0000
	20	20	78.0862	300		0.0000
	40	40	123.6578	288		0.0000
	60	60	208.7162	271		0.0000
	80	80	348.8889	241		0.0000
	100	100	447.9946	219		0.0000
	....					
	260	260	1416.4902	53		0.0000
	280	280	1561.0237	34		0.0000
	300	300	1757.6146	8		0.0000
*	305	305	1792.0000	0	1792.0000	0.0000
	320	316	32.0996	310	1792.0000	0.0000
	....					

. . . continues for a long time with no improvement

Applying a Greedy Heuristic

```
param conflict; param min_conflict; param min_group;

for {p in PEOPLE} {
   let min_conflict := Infinity;

   for {j in 1..numberGrps} {
      let conflict := sum {(p,i,k) in SAMETYPE} Assign[i,j];

      if conflict < min_conflict then {
         let min_conflict := conflict;
         let min_group := j;
         }
      }

   let Assign[p,min_group] := 1;
}
```

```
ampl: include balAssignGreedy.run;

TotalSameness = 1762
```

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.

(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
```

(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                                             Cuts/
   Node  Left     Objective  IInf  Best Integer     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

    .......
```

(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
```

(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
```

Solving for Minimum Variation

```
1054 variables:
        1000 binary variables
        54 linear variables
560 constraints, all linear; 12200 nonzeros
1 linear objective; 54 nonzeros.

CPLEX 9.0.0:
Clique table members: 100
MIP emphasis: balance optimality and feasibility

        Nodes                                                Cuts/
   Node  Left     Objective  IInf  Best Integer      Best Node

      0     0       17.0000   212                      17.0000
                    17.0000   228                   Fract:  49
*     0+    0                   0       19.0000        17.0000
      1     1       17.0000   183       19.0000        17.0000
      2     2       17.0000   146       19.0000        17.0000
      3     3       17.0000   170       19.0000        17.0000
      4     4       17.0000   153       19.0000        17.0000
      5     5       17.0000   103       19.0000        17.0000
      6     6       17.0000    81       19.0000        17.0000
      7     7       17.0000    72       19.0000        17.0000
```

Solving for Minimum Variation

```
      8     8        17.0000     88        19.0000       17.0000
      9     9        17.0000     69        19.0000       17.0000
     10    10        17.0000     69        19.0000       17.0000
     11    11        17.0000     77        19.0000       17.0000
     12    12        17.0000     73        19.0000       17.0000
     13    13        17.0000     68        19.0000       17.0000
     14    14        17.0000     99        19.0000       17.0000
     15    15        17.0000     98        19.0000       17.0000
     16    16        17.0000    128        19.0000       17.0000
     17    17        17.0000    140        19.0000       17.0000
     18    18        17.0000     95        19.0000       17.0000
*    19     1                     0        17.0000       17.0000

Gomory fractional cuts applied: 10

Times (seconds):
Input =  0.02
Solve =  16.844
Output = 0.02

CPLEX 9.0.0: optimal integer solution; objective 17
5624 MIP simplex iterations
19 branch-and-bound nodes
```

Summary of Results

Original	Total sameness	Max variation	Total variation	Time
Greedy	***1762***	***3***	***45***	***seconds***
Quadratic	***1744***	***2***	***39***	***4.7 min***
Min total variation	***1706***	***1***	***17***	***14.4 min***
New				
Quadr continuous	***1752***	***2***	***44***	***4.07 sec***
Quadr integer	***1720***	***2***	***27***	***107 min ****
Min total variation	***1706***	***1***	***17***	***16.8 sec***

Scaling Up

Model is 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 is 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 “Location-Rank” 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});
```

(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));
```

(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

Observation #5

Clients can invent and change the rules as they wish

Assignment of people is a social, not physical, problem

“Oh, we forgot to mention . . .”

One more complication

- No group may have only 1 woman

Not a problem, though

- Women are between 18% and 22%
  of every group in solution already sent!

. . . ***client’s* ad hoc *solutions***
must have been pretty bad

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

More Recent Cases

Balanced series of assignments

Sequence of workshop assignments

Balanced class seat assignments

Observation #6

Subsequent problems may get harder

But they may just as well get easier