IE418: Integer Programming

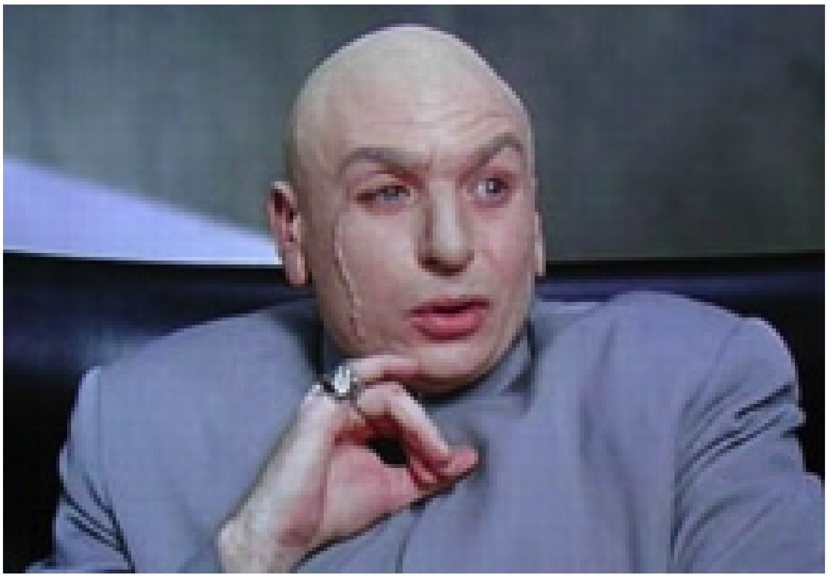
Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

9th February 2005

Review

- Hand in Homeworks!



A Simple LP

- The WorldLight Company produces two types of light fixtures (products 1 and 2) that require both metal frame parts and electrical components.
- For each unit of product 1, 1 unit of frame parts and 2 units of electrical components are required.
- For each unit of product 2, 3 units of frame parts and 2 units of electrical components are required.
- The company has 200 units of frame parts and 300 units of electrical components.
- Each unit of product 1 gives a net profit of \$1, and each unit of product 2, up to 60 units, gives a profit of \$2.
- Any excess over 60 units of product 2 brings no profit, so such an excess has been rules out explicity.

LP Instance

$$\max x_1 + 2x_2$$

subject to

$$\begin{aligned} x_1 + 3x_2 &\leq 200 \\ 2x_1 + 2x_2 &\leq 300 \\ x_2 &\leq 60 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Communicating Instances to a Solver

1. Formulate the model
2. Gather all the data
3. Generate the constraint matrix for your instance and data. (A, b, c, etc)
4. Type the entire constraint matrix into a file using a "standard format"
5. Pass the file to a solver
6. Get the answer and interpret it in terms of the original model

Problems with this approach

- The constraint matrices can be huge!!!
  - Maybe write a "matrix generation" program to create the constraint matrix file.
- If you want to modify the model parameters or data, you have to retype the entire matrix.
- The "standard" file format, called *MPS Format* is...
  - Old.
  - So very, very ugly.

How Ugly Is It?

```
NAME
ROWS
 N   obj
 L   c1
 L   c2
 L   c3
COLUMNS
    x1        obj                -1    c1                   1
    x1        c2                  2
    x2        obj                -2    c1                   3
    x2        c2                  2    c3                   1
RHS
    rhs       c1                200    c2                 300
    rhs       c3                 60
ENDATA
```


Recognize this problem?

- It's your old friend WorldLight!

maximize

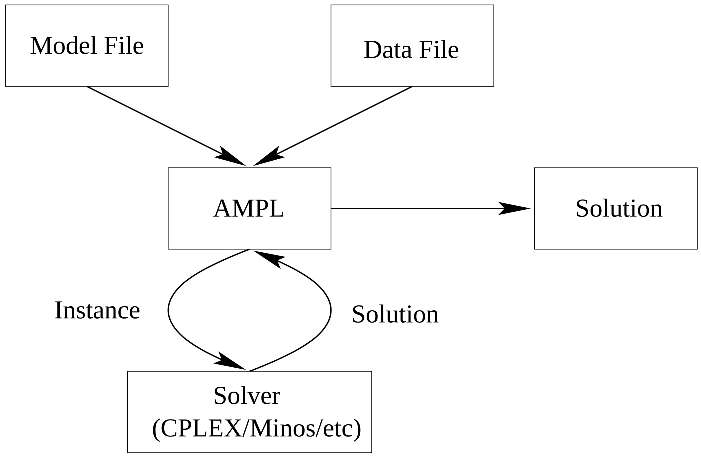
$$x_1 + 2x_2$$

subject to

$$\begin{array}{rcll} x_1 + 3x_2 & \leq & 200 & \text{Frame Part Units} \\ 2x_1 + 2x_2 & \leq & 300 & \text{Electrical Components} \\ x_2 & \leq & 60 & \text{Rule out production over 60 units} \\ x_1 & \geq & 0 & \text{The immutable laws of physics} \\ x_2 & \geq & 0 & \text{The immutable laws of physics} \end{array}$$

AMPL Concepts

- AMPL is an Algebraic Modeling Language
- In many ways, AMPL is like any other programming language.
- It just has special syntax that helps us create an optimization instance and interact with optimization solvers.



Modeling and Solving in AMPL

```
ampl: option solver cplexamp;
ampl: var x1;
ampl: var x2;
ampl: maximize profit: x1 + 2 * x2;
ampl: subject to frame_parts: x1 + 3 * x2 <= 200;
ampl: subject to electrial_components: 2 * x1 + 2 * x2 <= 300;
ampl: subject to x2_prod_limit: x2 <= 60;
ampl: subject to x2_lb: x2 >= 0;
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 175
3 simplex iterations (0 in phase I)
ampl: display x1;
x1 = 125
ampl: display x2;
x2 = 25
ampl: quit;
```

Generalizing the Model

- Suppose we want to generalize the model to more than two products
  - AMPL (and all "real" modeling environments) allow the model to be separated from the data
  - This is IMPORTANT!!!
- Data
  - Sets: lists of products, materials, etc
  - Parameters: numerical inputs such as costs, etc
- Model
  - Variables: The values to be decided upon
  - Objective Function
  - Constraints

Fickle Management

- Management now has decided that it wants to build *five* new products.

	Product 1	Product 2	Product 3	Product 4	Product 5
Frame Parts	1	3	2	3	1
Elec. Comp.	2	2	2	1	3
Profit	1	2	1.4	1.8	1.7
Prod. Limit	∞	60	80	50	66

The Generalized WorldLight Problem – In AMPL

```
set PROD;
param profit {PROD};
param frame_req {PROD};
param elec_req {PROD};
param max_production {PROD};

var x{PROD} >= 0;

maximize total_profit:
sum {i in PROD} profit[i] * x[i];
```


GWP, Cont.

```
subject to frame_parts:
sum {i in PROD} frame_req[i] * x[i] <= 200;

subject to electrial_components:
sum {i in PROD} elec_req[i] * x[i] <= 300;

subject to production_limits {i in PROD}:
x[i] <= max_production[i];
```

New World Light Data File wl-1.dat

```
set PROD := p1 p2 p3 p4 p5;

param: profit frame_req elec_req max_production :=
p1 1 2 1 Infinity
p2 3  2 2 60
p3 2 2 1.4 80
p4 3 1 1.8 50
p5 1 3 1.7 66 ;
```


Solving the Big WorldLight Problem

```
ampl: model wl.mod;
ampl: data wl-1.dat;
ampl: data wl-1.dat;
ampl: solve;
CPLEX 7.1.0: optimal solution; objective 360
3 simplex iterations (0 in phase I)
ampl: display x;
x [*] :=
p1   0
p2  60
p3  15
p4  50
p5   0  ;
```

Important AMPL Notes

- The # character starts a comment
- Variables are declared using the **var** keyword.
- All statements must end in a semi-colon;
- Names must be unique!
  - A variable and a constraint cannot have the same name
- AMPL is case sensitive. Keywords must be in lower case.

Getting AMPL

- AMPL is available in COR@L (`/usr/local/bin/ampl`)
- Student versions at `http://www.ampl.com`
  - Limited to 300 variables and 300 constraints.
  - You will also want to get the AMPL/CPLEX Solver
- There are "full fledged" versions of solvers you can use with AMPL on NEOS.
- What is NEOS?
  - Shame on you if you haven't ever used it.
  - `http://www.mcs.anl.gov/neos`

Fun, Interactive Portion of Class

- Let's solve a TSP!
- How to deal with those pesky "subtour eliminations?"
- Let's solve the problem without them first...

The Separaration Problem

Given $\hat{x} \in \mathbb{R}^{|E|}$, does $\exists S \subseteq V$ such that

$$\sum_{e \in \delta(S)} \hat{x}_e < 1?$$

- $\delta(S) = \{e = (i,j) \in E \mid i \in S, j \notin S\}$
- Does this problem look familiar?
  - min $s - t$ cut!
- Is the problem easier if $x \in \mathbb{B}^{|E|}$?

Our TSP

- Through 10 cities in the United States.

```
param c : Atlanta Chicago Denver Houston LosAngeles Miami NewYork SanFrancisco Seattle W
ashingtonDC :=
Atlanta        0  587 1212  701 1936  604  748 2139 2182  543
Chicago      587    0  920  940 1745 1188  713 1858 1737  597
Denver      1212  920    0  879  831 1726 1631  949 1021 1494
Houston      701  940  879    0 1372  968 1420 1645 1891 1220
LosAngeles  1936 1745  831 1374    0 2339 2451  347  959 2300
Miami        604 1188 1726  968 2339    0 1092 2594 2734  923
NewYork      748  713 1631 1420 2451 1092    0 2571 2408  205
SanFrancisco 2139 1858  949 1645  347 2594 2571    0  678 2442
Seattle      2182 1737 1021 1891  959 2734 2408  678    0 2329
WashingtonDC  543  597 1494 1220 2300  923  205 2442 2329    0
;
```

Mosel

- A modeling language from Dash Optimization that uses the Xpress-MP optimizer
- On shark
  - In /usr/local/shark
  - file:///usr/local/xpress/docs/mosel/mosel_ug/dhtml/moselug.html
  - Software: /home/jeff/share/xpress
- For you Windoze lovers, there exists an XPRESS-IVE pun intended, I believe which allows you to model, debug, run, analyze, etc...
- You want it?

Important Resources

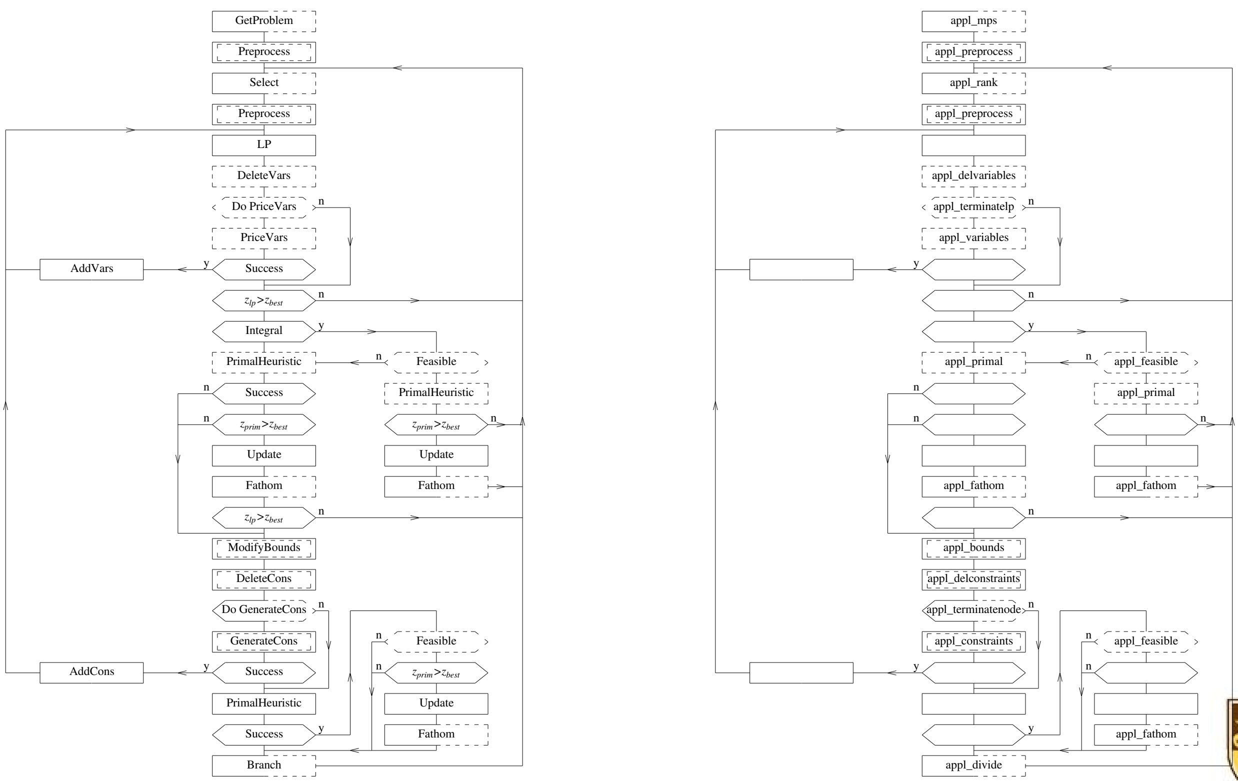
Prof. Linderoth has created a Wiki at http://coral.ie.lehigh.edu/cgi-bin/wiki.pl

CPLEX and XPRESS

- CPLEX is installed in /usr/local/cplex
- XPRESS is installed in /usr/local/xpress
- We showed these off a bit this morning—See the Wiki and the documentation if you need more details.

MINTO

- MINTO is a flexible (relatively) powerful solver for general mixed integer programs.
- `minto [-xo<.>m<.>t<.>be<.>E<.>p<.>hcikgfrRB<.>sn<.>a] <name>`
- The "power" of MINTO lies in the (relative) ease with with the branch-and-{bound, cut, price} algorithm can be customized
- Installed in COR@L in `/usr/local/minto31-linux-*`



MINTO options

option	effect
x	assume maximization problem
o $< 0, 1, 2, 3 >$	level of output
m $< \ldots >$	maximum number of nodes to be evaluated
t $< \ldots >$	maximum cpu time in seconds
b	deactivate bound improvement
e $< 0, 1, 2, 3, 4, 5 >$	type of branching
E $< 0, 1, 2, 3, 4 >$	type of node selection
p $< 0, 1, 2, 3 >$	level of preprocessing and probing
h	deactivate primal heuristic
c	deactivate clique generation
i	deactivate implication generation
k	deactivate knapsack cover generation
g	deactivate GUB cover generation
f	deactivate flow cover generation
r	deactivate row management
R	deactivate restarts
B	$< 0, 1, 2 >$ type of forced branching
s	deactivate all system functions
n $< 1, 2, 3 >$	activate a names mode
a	activate use of advance basis

Branching and Node Selection

- $e < 0, 1, 2, 3, 4, 5 >$
  - maximum infeasibility (0),
  - penalty based (1),
  - strong branching (2),
  - pseudocost based (3),
  - adaptive (4),
  - SOS branching (5).
- $E < 0, 1, 2, 3, 4 >$
  - best bound (0),
  - depth first (1),
  - best projection (2),
  - best estimate (3), and
  - adaptive (4).

Building MINTO

- There are "two" MINTOs in COR@L.
  1. One uses CPLEX to solve the LP relaxation
  2. One uses COIN-OR (Clp) to solve the LP relaxation
- We'll use the (Clp) version for now

---

1. `cp -r /usr/local/minto31-linux-osiclp/APPL .`
2. `cd APPL`
3. `make`
4. `ls -l minto`

What the !@#!@#!@#** is `make`

- `make` is a command for making something :-)
- In this case, we are making the minto executable
- If you wish to modify the behavior of minto through the use of the `appl_` functions, you simply write the C code in the functions, and type `make` again.
- If you don't know C, you will not be able to use MINTO.
- Need some pointers on learning C?
  - google learning C
  - Buy a book
  - Stop by my office and ask for help...
- Demonstration...

inq_form()

- A call to `inq_form()` initializes the variable *info_form* that has the following structure:

```
typedef struct info_form {
    int form_vcnt;        /* number of variables in the formulation */
    int form_ccnt;        /* number of constraints in the formulation */
} INFO_FORM;
```


inq_form() example

```
/*
 * E_SIZE.C
 */

#include <stdio.h>
#include "minto.h"

/*
 * WriteSize
 */

void
WriteSize ()
{
    inq_form ();
    printf ("Number of variables:   %d\n", info_form.form_vcnt);
    printf ("Number of constraints: %d\n", info_form.form_ccnt);
}
```

inq_var()

```
typedef struct info_var {
    char     *var_name;     /* name, if any */
    char     var_class;     /* class: CONTINUOUS, INTEGER, or BINARY */
    double   var_obj;       /* objective function coefficient */
    int      var_nz;        /* number of constraints with nonzero coefficients *
    int      *var_ind;      /* indices of constraints with nonzero coefficients
    double   *var_coef;     /* actual coefficients */
    int      var_status;    /* ACTIVE, INACTIVE, or DELETED */
    double   var_lb;        /* lower bound */
    double   var_ub;        /* upper bound */
    VLB      *var_vlb;      /* associated variable lower bound */
    VUB      *var_vub;      /* associated variable upper bound */
    int      var_lb_info;   /* ORIGINAL, MODIFIED_BY_MINTO,
                               MODIFIED_BY_BRANCHING, or MODIFIED_BY_APPL */
    int      var_ub_info;   /* ORIGINAL, MODIFIED_BY_MINTO,
                               MODIFIED_BY_BRANCHING, or MODIFIED_BY_APPL */
} INFO_VAR;
```


inq_var() Cont.

- If $y_j \leq u_j x_j, (x_j \in \{0, 1\})$, y_j is said to have a *variable upper bound*.
- These are used to generate some classes of strong valid inequalities

```
typedef struct {
    int    vlb_var;       /* index of associated 0-1 variable */
    double vlb_val;       /* value of associated bound */
} VLB;

typedef struct {
    int    vub_var;       /* index of associated 0-1 variable */
    double vub_val;       /* value of associated bound */
} VUB;
```

Example of inq_var()

```
/*
 * E_FIXED.C
 */

#include <stdio.h>
#include "minto.h"

/*
 * WriteFixed
 */

void
WriteFixed ()
{
    int j;
    int nvar;

    inq_form();
    nvar = info_form.form_vcnt;
    for (j = 0; j < nvar; j++) {
      inq_var (j, NO);
      if (info_var.var_lb > info_var.var_ub - 1.0e-6) {
        printf ("Variable %d is fixed at %f\n", j, info_var.var_lb);
      }
    }
}
```


inq_constr

```
typedef struct info_constr {
    char     *constr_name;   /* name, if any */
    int      constr_class;   /* classification: ... */
    int      constr_nz;      /* number of variables with nonzero coefficients */
    int      *constr_ind;    /* indices of variables with nonzero coefficients *
    double   *constr_coef;   /* actual coefficients */
    char     constr_sense;   /* sense */
    double   constr_rhs;     /* right hand side */
    int      constr_status;  /* ACTIVE, INACTIVE, or DELETED */
    int      constr_type;    /* LOCAL or GLOBAL */
    int      constr_info;    /* ORIGINAL, GENERATED_BY_MINTO,
                                GENERATED_BY_BRANCHING, or GENERATED_BY_APPL */
} INFO_CONSTR;
```

inq_constr() Example

```
/*
 * E_TYPE.C
 */

#include <stdio.h>
#include "minto.h"

/*
 * WriteType
 */

void
WriteType ()
{
    int i;

    for (inq_form (), i = 0; i < info_form.form_ccnt; i++) {
        inq_constr (i);
        printf ("Constraint %d is of type %s\n",
            i, info_constr.constr_type == GLOBAL ? "GLOBAL" : "LOCAL");
    }
}
```


Constraint Classes in MINTO

class	constraint
MIXUB	$\sum_{j \in B} a_j x_j + \sum_{j \in I \cup C} a_j y_j \leq a_0$
MIXEQ	$\sum_{j \in B} a_j x_j + \sum_{j \in I \cup C} a_j y_j = a_0$
NOBINUB	$\sum_{j \in I \cup C} a_j y_j \leq a_0$
NOBINEQ	$\sum_{j \in I \cup C} a_j y_j = a_0$
ALLBINUB	$\sum_{j \in B} a_j x_j \leq a_0$
ALLBINEQ	$\sum_{j \in B} a_j x_j = a_0$
SUMVARUB	$\sum_{j \in I^+ \cup C^+} a_j y_j - a_k x_k \leq 0$
SUMVAREQ	$\sum_{j \in I^+ \cup C^+} a_j y_j - a_k x_k = 0$
VARUB	$a_j y_j - a_k x_k \leq 0$
VAREQ	$a_j y_j - a_k x_k = 0$
VARLB	$a_j y_j - a_k x_k \geq 0$
BINSUMVARUB	$\sum_{j \in B \setminus \{k\}} a_j x_j - a_k x_k \leq 0$
BINSUMVAREQ	$\sum_{j \in B \setminus \{k\}} a_j x_j - a_k x_k = 0$
BINSUM1VARUB	$\sum_{j \in B \setminus \{k\}} x_j - a_k x_k \leq 0$
BINSUM1VAREQ	$\sum_{j \in B \setminus \{k\}} x_j - a_k x_k = 0$
BINSUM1UB	$\sum_{j \in B} x_j \leq 1$
BINSUM1EQ	$\sum_{j \in B} x_j = 1$

Adapting MINTO. appl_constraints()

```
unsigned
appl_constraints (id, zlp, xlp, zprimal, xprimal, nzcnt, ccnt, cfirst,
                  cind, ccoef, csense, crhs, ctype, cname, sdim, ldim)
int id;              /* identification of active minto */
double zlp;          /* value of the LP solution */
double *xlp;         /* values of the variables */
double zprimal;      /* value of the primal solution */
double *xprimal;     /* values of the variables */
int *nzcnt;          /* variable for number of nonzero coefficients */
int *ccnt;           /* variable for number of constraints */
int *cfirst;         /* array for positions of first nonzero coefficients */
int *cind;           /* array for indices of nonzero coefficients */
double *ccoef;       /* array for values of nonzero coefficients */
char *csense;        /* array for senses */
double *crhs;        /* array for right hand sides */
int *ctype;          /* array for the constraint types: LOCAL or GLOBAL */
int **cname;         /* array for the names */
int sdim;            /* length of small arrays */
int ldim;            /* length of large arrays */
{
}
```


Using appl_constraints()

- Suppose after some processing, I realize that I would like to add three cutting planes to the global formulation of my IP instance.

$$\begin{aligned} x_1 + 2x_2 &\leq 7 \\ x_1 + x_2 - x_3 &\leq 2 \\ -7x_1 + x_4 &\geq 0 \end{aligned}$$

C Code Example in appl_constraints()

```
/* Number of constraints */
*ccnt = 3;

/* Number of nonzeroes */
*nzcnt = 7;

cfirst[0] = 0;
cfirst[1] = 2;
cfirst[2] = 5;
cfirst[3] = 7;

cind[0] = 0;
cind[1] = 1;
cind[2] = 0;
cind[3] = 1;
cind[4] = 2;
cind[5] = 0;
cind[6] = 3;

ccoef[0] = 1.0;
ccoef[1] = 2.0;
ccoef[2] = 1.0;
ccoef[3] = 1.0;
ccoef[4] = -1.0;
ccoef[5] = -7.0;
ccoef[6] = 1.0;

csense[0] = 'L';
csense[1] = 'L';
csense[2] = 'G';

crhs[0] = 7.0;
crhs[1] = 2.0;
crhs[2] = 0.0;

ctype[0] = GLOBAL;
ctype[1] = GLOBAL;
ctype[2] = GLOBAL;

cname[0] = '\0';
cname[1] = '\0';
cname[2] = '\0';
return(SUCCESS);
```


Class Stuff

- Homework #2 — Due on 2/21.
  - It will be posted sometime tomorrow.
  - I will also post partial[2] homework #1 answers sometime this weekend.
  - Start getting familiar with the software now!
- Midterm Exam — Will be on 3/2 (Last class before Spring Break)

[2] Read: The ones I typed up