IE418: Integer Programming

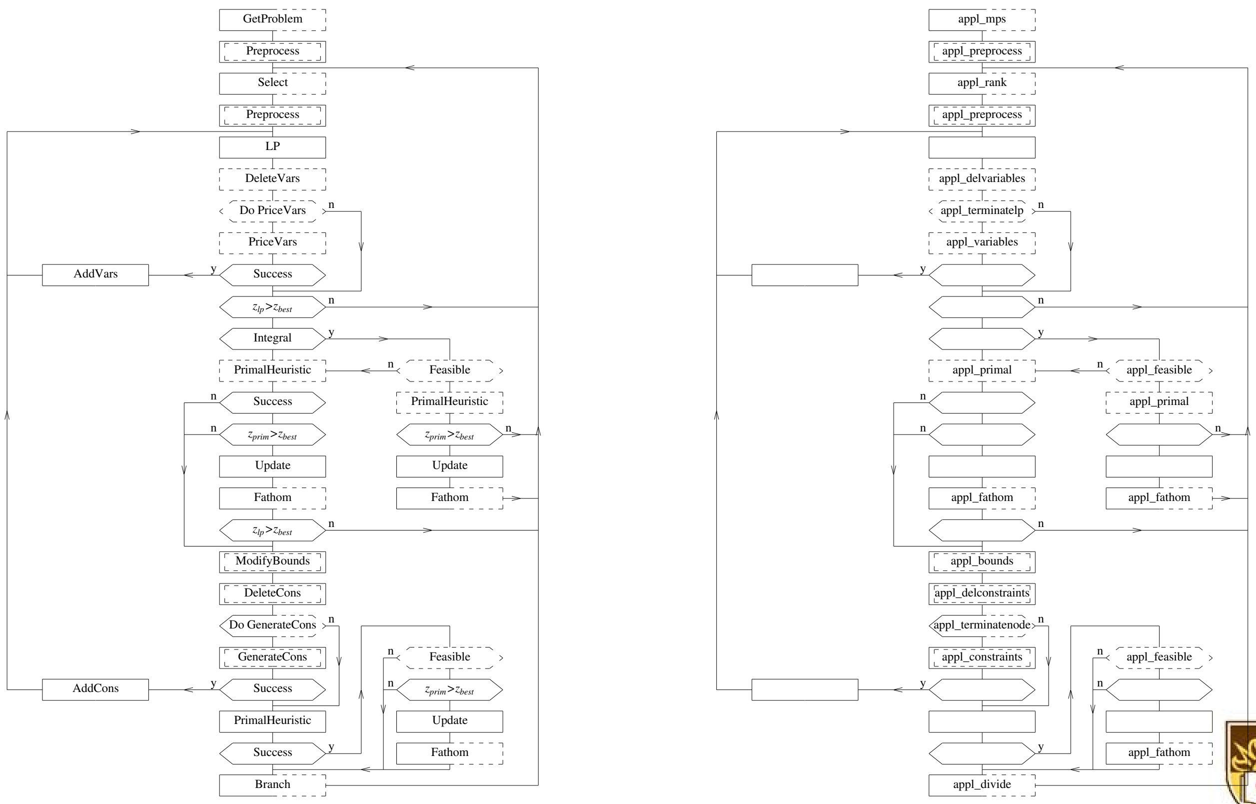
Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

14th February 2005

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);
```

Separated at Birth?

MINTO

$\neq$

SYMPHONY

SYMPHONY

- SYMPHONY is another wonderful framework for solving MIPs.
- MINTO is better
  - As a "black box" solver
  - For generating columns (branch-and-price)
- SYMPHONY is better...