Modeling and Solving Nontraditional Optimization Problems

Session 3b: Discrete Solver Support

Robert Fourer

Industrial Engineering & Management Sciences
Northwestern University

AMPL Optimization LLC

4er@northwestern.edu — 4er@ampl.com

Chiang Mai University International Conference
Workshop
Chiang Mai, Thailand — 4-5 January 2011

Session 3b: Discrete Solver Support

Focus

- Constraint programming
  as an alternative solver approach for discrete optimization

Topics

- Traditional branch-and-bound
- Alternative constraint programming approach
  - Example
  - Principles
  - Practical issues
  - Trends

Solvers for Discrete Optimization

MIP: branch-and-bound approach

- Build a search tree (“branching”)
- Solve linear programs at tree nodes (“bounding”)

CP: constraint programming approach

- Build a search tree
- Reduce search space at tree nodes
  through alternative methods

Local-search metaheuristics

- Progressively improve the solution
  - simulated annealing, tabu search, evolutionary algorithms,
    scatter search, ant colony opt, particle swarm opt, . . .
- Mostly special purpose
  - but used in a general way within tree-search methods

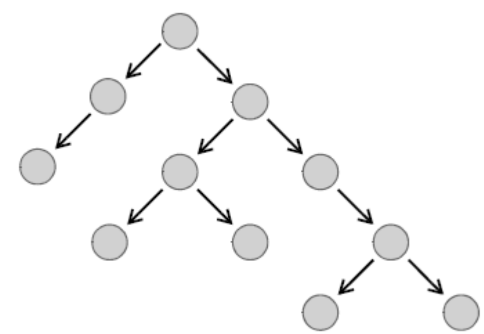
Branch-and-Bound

Root node

- ❖ Analyze (“presolve”) to reduce problem size
- ❖ Solve LP relaxation for fractional solution (lower bound)
- ❖ Apply heuristics to seek integer solution (upper bound)
- ❖ Generate constraints (“cuts”) to improve the LP

. . . ***repeat while progress is made***

Child nodes



- ❖ Split a fractional variable into two cases
  - ∗ for binary variables, zero or one
- ❖ Repeat as above for each child problem
- ❖ Stop branching at node (“fathom”) if . . .
  - ∗ all variables of LP relaxation are integral
  - ∗ lower bound is too high

Branch-and-Bound *(cont'd)*

Termination

- No nodes left to consider
- Lower bound close enough to upper bound
- Current best integer solution seems good enough

Computational cost

- Tree grows exponentially in worst case
- Often reasonably efficient in practice

*. . . **but not always!***

Branch-and-Bound *(cont'd)*

Log from Gurobi run

```
Optimize a model with 1358 Rows, 2204 Columns and 7649 NonZeros
Presolve removed 463 rows and 563 columns
Presolve time: 0.06s
Presolved: 895 Rows, 1641 Columns, 5766 Nonzeros

Root relaxation: objective 3.164090e+06, 489 iterations, 0.00 seconds

.......
```

Branch-and-Bound *(cont'd)*

Log from Gurobi run (cont'd)

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 3164089.95    0   36          - 3164089.95      -      -    0s
     0     0 3198679.75    0   21          - 3198679.75      -      -    0s
     0     0 3203107.26    0    9          - 3203107.26      -      -    0s
     0     0 3204224.70    0   11          - 3204224.70      -      -    0s
     0     0 3204433.46    0    8          - 3204433.46      -      -    0s
     0     0 3204487.19    0   11          - 3204487.19      -      -    0s
     0     0 3204487.19    0   11          - 3204487.19      -      -    0s
H    0     0                    4735286.40 3204487.19  32.3%     -    0s
     0     2 3204487.19    0   12 4735286.40 3204487.19  32.3%     -    0s
H   33    28                    3220327.86 3205716.57  0.45%   5.0    0s
*  388   275              41    3220213.49 3205792.16  0.45%   5.4    0s
*  815   351              77    3214965.89 3205792.16  0.29%   5.3    0s
*  852    98              76    3209168.61 3205792.16  0.11%   5.3    0s
   952    21 3208517.98    9    2 3209168.61 3208314.79  0.03%   5.2    1s

Explored 1266 nodes (6891 simplex iterations) in 1.12 seconds
Thread count was 8 (of 8 available processors)

Best objective 3.2091686060e+06, best bound 3.2090006626e+06, gap 0.0052%
```

Constraint Programming

Similarities

- Builds and prunes search tree
- May solve efficiently in practice

Differences

- No linear programs solved
- Aggressive reduction of variable domains at each node

. . . ***different approach to pruning the tree***

Constraint Programming *(cont'd)*

Example

- Solving an assignment problem
- Summary of propagation rules used

Principles

- Search for a solution
- Optimization of an objective

Practice

- Constraint propagation
- Search strategies
- Formulation guidelines

Trends in CP for discrete optimization . . .

Discrete Solvers

CP Example

Assign professors to offices

```
enum FACULTY = ... ;
enum OFFICES = ... ;

int+ pref[FACULTY,OFFICES] = ... ;
int+ cutoff = ... ;

var OFFICES Assign[FACULTY];

minimize

   sum(j in FACULTY) pref[j,Assign[j]]

subject to {

   alldifferent(Assign);

   forall(j in FACULTY) pref[j,Assign[j]] < cutoff;

};
```

Example

Data

Data for 6 professors, 6 offices

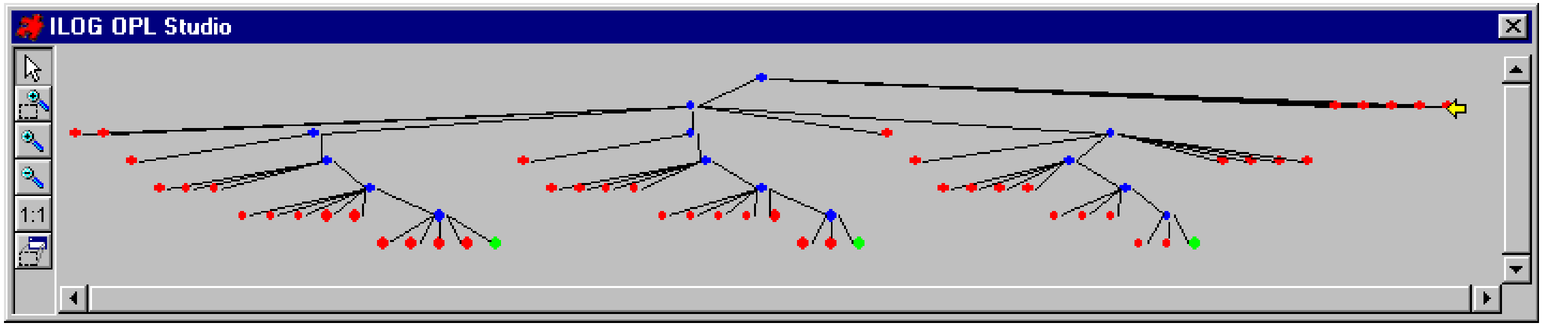
```
FACULTY = {Birge Coullard Daskin Fourer Munson Nocedal};

OFFICES = {C24 C34 C42 D16 D19 D23};

pref = [ [1,2,3,4,5,6],
         [2,5,4,3,6,1],
         [3,2,1,4,6,5],
         [6,4,5,2,1,3],
         [4,3,6,2,4,1],
         [3,4,2,5,1,6] ];

cutoff = 5;
```

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

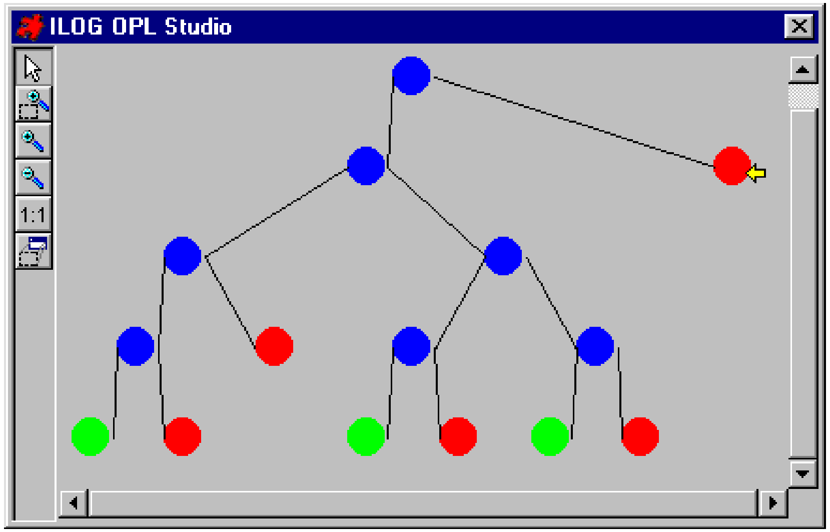
Example

Search Trees

Without constraint propagation (domain filtering)



With constraint propagation



Example

Search Details

Initialize domains

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate cutoff constraints

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Branch on Birge = C24

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate all-diff constraint

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Example

Search Details *(cont'd)*

Branch on Coullard = C42

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate all-diff constraint

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Branch on Daskin = C34

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate all-diff constraint

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Example

Search Details *(cont'd)*

Nocedal = D19 is forced

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate all-diff constraint

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Branch on Fourer = D16

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate & force Munson = D23

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

. . . add objective ≤ 10 to constraints

Example

Search Details *(cont'd)*

Backtrack to Fourer = D23

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate objective: fail

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Backtrack to Daskin = D16

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate alldiff & objective: fail

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Example

Search Details *(cont'd)*

Backtrack to Coullard = D16

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate alldiff constraint

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate obj: Fourer ≤ 3, . . .

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Munson ≤ 3, Nocedal ≤ 3

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Example

Search Details *(cont'd)*

Branch on Daskin = C34

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate alldiff & fix

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

. . . add objective ≤ 9 to constraints

Backtrack to Daskin = C42

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate alldiff & fix: fail

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Example

Search Details *(cont'd)*

Branch on Coullard = D23

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate alldiff & objective

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Branch on Daskin = C34

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

Propagate alldiff & objective

	C24	C34	C42	D16	D19	D23
Birge	1	2	3	4	5	6
Coullard	2	5	4	3	6	1
Daskin	3	2	1	4	6	5
Fourer	6	4	5	2	1	3
Munson	4	3	6	2	4	1
Nocedal	3	4	2	5	1	6

. . . add objective ≤ 8 to constraints

possibilities for further improvement quickly eliminated

Example

Constraint Propagation Rules

alldifferent(Assign)

if `Assign[p]` fixed at `k`,
remove `k` from domain of all `Assign[j]`, `j` ≠ `p`

Score[j] = pref[j,Assign[j]]

dom-min(`Score[j]`) =
the smallest `pref[j,k]` over all `k` in domain of `Assign[j]`

dom-max(`Score[j]`) =
the largest `pref[j,k]` over all `k` in domain of `Assign[j]`

Score[j] < cutoff

dom-max(`Score[j]`) ≤ `cutoff - 1`

sum(j in FACULTY) Score[j] ≤ best-so-far - 1

for each `p` in `FACULTY`, *dom-max* (`Score[p]`) ≤
(*best-so-far* `- 1`) `- sum(j in FACULTY: j` ≠ `p)` *dom-min*(`Score[j]`)

if `sum(j in FACULTY)` *dom-min*(`Score[j]`) ≥ *best-so-far*, *fail*

Search for a Solution (Depth-First)

Initialize

Set *domains* of all variables

Call SeekSol(domains)

SeekSol

If all variables fixed,

Stop with solution found

Choose a variable not yet fixed

Repeat for each value in the chosen variable's domain:

Fix the variable at the value

Reduce all other variables' domains accordingly

If all domains remain non-empty,

Call SeekSol(domains)

. . . reduction in domains is called
constraint propagation or domain filtering

Search for a Solution (General)

Initialize

Define one node, the *root*

Set *domains* of all variables

Repeat until solution found

Choose:

a *node* where all variables' domains are non-empty
a *variable* not already fixed at that node
a *value* in the domain of that variable

Create a child node:

fix the chosen variable at the chosen value
reduce variables' domains at the child node accordingly

Repeat while there's an empty domain at any node:

delete the node
reduce variables' domains at the parent node accordingly

. . . to find multiple solutions, don't stop after the first

Optimization of an Objective

Find a first solution

Apply previously described search

Repeat until no solution found

Add constraint:

objective function $\leq$ *best objective value so far* `- 1`

Seek another solution:

apply previously described search

. . . optionally re-start each search at root

Constraint Propagation

Principles

Every variable has a current domain

Using a constraint and current domains of its variables,
infer tighter domains on its variables

Constraints interact only through effects on domains

"Good" constraints permit
propagation from any one variable to all others

Properties

Any conditions (however complex) can serve as constraints
if good *fast* domain filtering routines are available

Specialized sequencing constraints work well

Expressions can be given domains
by adding constraints equating them to new variables

Constraint Propagation *(cont'd)*

Linear (in)equalities

Deduce tighter upper bound on one variable
from lower bounds on the other variables

All-different

Reduce domains by solving a *matching problem*

Variables in subscripts

Create an *element constraint*
by equating the subscripted entity to a new variable

Propagation both ways

```
alldifferent(Assign)
Score[j] = pref[j,Assign[j]]
sum(j in FACULTY) Score[j] ≤ best-so-far - 1
```

Practice

Search Strategies: Standard

Branch from which node?

Depth-first: from the most recently visited active node

Best-first, limited-discrepancy, interleaved depth-first, etc.

Fix which variable?

First-fail: one with the smallest current domain size

Smallest current domain minimum, best objective value, etc.

Lexicographic: Smallest domain size,
breaking ties by smallest domain min, etc.

Fix it to which value?

Smallest in current domain, etc.

Dichotomize: choose “half” of domain rather than one value

Search Strategies: Priority

Fix which variable?

Priority: highest modeler-specified priority in current domain

Highest priority, breaking ties by smallest domain

Smallest domain, breaking ties by highest priority

Which variables need to be fixed?

Search on "actual" variables, not "defined" variables

```
var JobForSlot {1..nSlots} in JOBS;
var ComplTime {1..nJobs} integer > 0;

subj to ComplTimeDefn {k in 1..nSlots}:
   ComplTime[JobForSlot[k]] =
      min( dueTime[JobForSlot[k]],
           ComplTime[JobForSlot[k+1]]
           - procTime[JobForSlot[k+1]]
           - setupTime[JobForSlot[k],JobForSlot[k+1]] )
```

Practice

Search Strategies: Specialized

Simplistic strategy: 152 nodes *(from OPL's search language)*

```
search {
   forall(j in FACULTY)
      tryall(k in OFFICES)
         Assign[j] = k; }
```

Standard strategy: 38 nodes

```
search {
   forall(j in FACULTY ordered by increasing dsize(Assign[j]))
      tryall(k in OFFICES: isInDomain(Assign[j],k))
         Assign[j] = k; }
```

Specialized strategy: 27 nodes

```
search {
   forall(j in FACULTY ordered by increasing dsize(Assign[j]))
      tryall(k in OFFICES: isInDomain(Assign[j],k)
            ordered by increasing pref[j,k])
         Assign[j] = k; }
```

Search Strategies: Complex

Search directives from two OPL models

```
search {
  forall(i in Domain
      ordered by increasing <dsize(queen[col,i]),abs(n/2-i)>)
    tryall(v in Domain ordered by increasing dsize(queen[row,i]))
      queen[col,i] = v;
};
```

```
SearchProcedure label(int day) {
  select(s in Scene: not bound(shoot[s]) ordered by decreasing
      <dsize(shoot[s]),sum(a in appears[s]) pay[a]>)
    tryall(d in Day: d <= day + 1 & isInDomain(shoot[s],d)) {
      shoot[s] = d;
      if d = day + 1 then label(d) else label(day) endif;
    }
}
search label(0);
```

Practice

Formulation Guidelines

Define fewer model components

Use fewer variables with larger domains

Use structure constraints (like `alldifferent`) rather than large numbers of simple constraints

Remove symmetries

Index variables over types rather than individuals

Introduce ordering among variables

Use set variables

Add redundant constraints

Provide more opportunities for domain filtering

Trends in CP for OR

Technological advances

Better selection of standard search strategies

Better domain filtering for specialized constraints

Cooperation with linear programming

LP formulation added to provide redundant constraints

LP treated as a structure constraint

LP generated from CP constraints,
updated each time CP domains are tightened

Integration with integer programming

New variable types, constraint types, branching rules
in IP branch-and-bound codes

More branching options, bounding computations
in CP solvers

Stronger modeling language support
for combinatorial optimization via CP or IP

To learn more . . .

Special Issue on
The Merging of Mathematical Programming
and Constraint Programming

INFORMS Journal on Computing

Volume 14, Number 4 (Fall 2002)

Irvin J. Lustig and Jean Francois Puget,
"Constraint Programming and its Relationship to
Mathematical Programming"

Interfaces

Volume 31, Number 6 (Nov/Dec 2001) 29–53