Reconnect '04
Using PICO

Cynthia Phillips, Sandia National Labs

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

Sandia National Laboratories

AMPL Model for 1| prec | $\sum w_j C_j$

• Declare/compute parameters, sets

```
param n > 0 integer;                # Number of job
set Jobs := 0..n-1;
param p {Jobs} >= 0 integer;        # Job Duration p(j)
param w {Jobs} >= 0 integer;        # Job weight (for objective) w(j)
# (predecessor, successor) pairs
set Precedence within Jobs cross Jobs;
# The makespan of the schedule (and latest finish time)
param T := sum{j in Jobs} p[j];
```


Sandia National Laboratories

AMPL Model for 1| prec | $\sum w_j C_j$

• Declare variables

```
var x {j in Jobs, t in p[j]..T} binary;
```


Sandia National Laboratories

AMPL Model for 1| prec | $\sum w_j C_j$

```
minimize WCT:
      sum{j in Jobs, t in p[j]..T} w[j] * t * x[j,t];

• subject to TaskDone {j in Jobs}:
   sum{t in p[j]..T} x[j,t] = 1;
• subject to UnitWork {tu in 1..T}:
      sum{j in Jobs,  t in tu..tu+p[j]-1 inter p[j]..T} x[j,t] <= 1;
• subject to PrecConstraint{j in Jobs,k in Jobs,
                              tu in p[j]..T-p[k]: (j,k) in Precedence}:
      sum{t in p[j]..tu}x[j,t] >=  sum{t in p[j]+p[k]..tu + p[k]} x[k,t];
```


Sandia National Laboratories

AMPL Model for 1| prec | $\sum w_j C_j$

```
minimize WCT:
      sum{j in Jobs, t in p[j]..T} w[j] * t * x[j,t];
```

Corresponds to:

$$\min \sum_{j=1}^{n} \sum_{t=p_j}^{T} w_j t x_{jt}$$

```
subject to TaskDone {j in Jobs}:
    sum{t in p[j]..T} x[j,t] = 1;
```

Corresponds to:

$$\sum_{t=p_j}^{T} x_{jt} = 1 \qquad \forall j$$

AMPL Model for 1| prec | $\sum w_j C_j$

```
subject to UnitWork {tu in 1..T}:
      sum{j in Jobs,  t in tu..tu+p[j]-1 inter p[j]..T} x[j,t] <= 1;
```

Corresponds to:

$$\sum_{j=1}^{n} \sum_{u=t}^{t+p_j-1} x_{ju} \le 1 \qquad t = 1,...,T = \sum_{j=1}^{n} p_j$$

```
subject to PrecConstraint{j in Jobs,k in Jobs,
                          tu in p[j]..T-p[k]: (j,k) in Precedence}:
      sum{t in p[j]..tu}x[j,t] >=  sum{t in p[j]+p[k]..tu + p[k]} x[k,t];
```

Corresponds to:

$$\sum_{u=1}^{t} x_{ju} - \sum_{u=1}^{t+p_k} x_{ku} \ge 0 \qquad \forall J_i \prec J_k,\ t = 1,...,T - p_k$$

More Complex Sets to Reduce IP size

```
# Transitive closure of precedence relations.
set step{s in 1..ceil((n-1)/2)} dimen 2 :=
  if s==1
    then Precedence
    else step[s-1] union setof {k in Jobs, (i,k) in step[s-1], (k,j) in step[s-1]} (i,j);
set TransPrec := step[ceil((n-1)/2)];
# Earliest finish time
param EFT{j in Jobs} := p[j] + sum{k in Jobs: (k,j) in TransPrec} p[k];
# Latest finish time
param LFT{j in Jobs} := T - sum{k in Jobs: (j,k) in TransPrec} p[k];
# Possible finish times
set FinishWindow{j in Jobs} := EFT[j]..LFT[j];
```


Smaller, simpler formulation

```
var x {j in Jobs, t in FinishWindow[j]} binary;
minimize WCT:
      sum{j in Jobs, t in FinishWindow[j]} w[j] * t * x[j,t];

subject to TaskDone {j in Jobs}:
    sum{t in FinishWindow[j]} x[j,t] = 1;

subject to UnitWork {tu in 1..T}:
      sum{j in Jobs,  t in tu..tu+p[j]-1 inter FinishWindow[j]} x[j,t] <= 1;
```

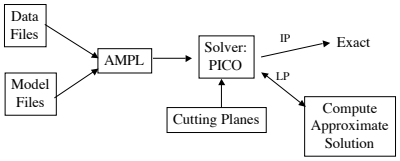
[Similar changes for precedence constraints]

AMPL Data Files - Straightforward

```
param n := 6;
param p :=
0 8
1 2
2 1
3 3
4 12
5 1;
set Precedence :=
0 2
1 2
2 3
4 5;
```


Sandia National Laboratories

Use Sandia National Labs IP Solver: PICO

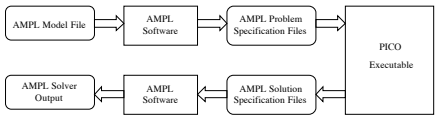


- Write cutting-plane and approximate-solution code using AMPL variables
- Mapping transparent

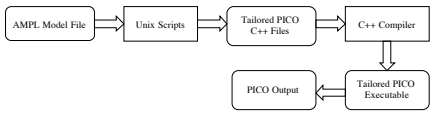
Sandia National Laboratories

AMPL-PICO Interface

Standard AMPL interfaces



Customized PICO Interface



Sandia National Laboratories

Caveats: Status as of June 22, 2004

- The derived classes are for branch and bound only (no cuts yet; not stable; but they'll be there "soon")
- Only the serial version works as of this morning
  - This could change by tomorrow

Sandia National Laboratories

Customized PICO Application

Goal: transparent access to AMPL variables from within problem-specific derivatives of core PICO application classes:

- MILP — The "master" branching class
- MILPNode — The "worker" subproblem class
- MILPProblem — The problem definition class

- Creates a new namespace, so (as far as you're concerned), these names are the same.
- Maps multidimensional ampl variables to PICO's linear variable space
  - Foo(1,2)
  - Foo(bar(stuff),callFunc(moreStuff))

Sandia National Laboratories

How PICO Learns AMPL Names

- Variable names, constraint names
  - Automatic (when the customized code is set up)
- Other parameters: list explicitly
  - You may never use some helper parameters from the ampl file

PICO SYMBOL: p n w EFT LFT T

Sandia National Laboratories

Example: Incumbent Heuristic for 1|prec| $\sum w_j C_j$

Computing midpoints by stepping through valid x in lexicographic order

```
DoubleVector frac(n());
IntVector time(n());
int curr_job = -1;
x_const_iterator curr = x_valid().begin();
x_const_iterator last = x_valid().end();
while (curr != last) {
  frac[curr->val1] += solution[x(*curr)];
  if (frac[curr->val1] > 0.5) {
    time[curr->val1] = curr->val2;
    //
    // Step past remaining jobs
    //
    curr_job = curr->val1;
    while (curr->val1 == curr_job)
      curr++;
    }
  else
    curr++;
}
```


Sandia National Laboratories

Alternative Iteration

Computing midpoints

```
DoubleVector frac(n());
IntVector time(n());
for (int i=0;i< n(); i++){
  for (int j=EFT(i) ; j<LFT(i) ; j++){
    if (x_valid.isMember(i,j)){
      frac[i] += solution[x(i,j)];
      if (frac[i] > 0.5) {
        time[i] = j;
        continue;
        }
      }
    }
```


Sandia National Laboratories

Building a custom indexing scheme

- Collect all the valid tuples with even indices into a list:

```
List<Tuple2<int,int>> my_index
x_const_iterator curr = x_valid().begin()
x_const_iterator end = x_valid().end()
While (curr != end){
  if (cur->val1 % 2 == 0) && (curr->val2 %2 ==0))
   my_index.push_back(*curr);
  curr++;
}
```

- There's no built-in iterator for this list (just an STL list)

Another Example: computing objective

```
DoubleVector soln(n());
IntVector index(n());
order(time,index); // sorting
soln [0] = p(index[0]);
value = p(index[0]) * w(index[0]);
for (int i=1; i< n(); i++) {
 soln[index[i]] = soln[index[i-1]] + p(index[i-1]);
 value += soln[index[i]] * w(index[i]);
 }
```


Incumbent Heuristics

- These code fragments would be part of the incumbentHeuristic() method in MILPNode.
- If you find an incumbent, this method must call
  globalPtr →updateIncumbent(solution, solution value)

This is a PICO (1D) solution.

A Customized PICO Application: A Simple Example

Command line:

gen_milp_app test.mod test.dat

Generates:

test.col	Column labels
test.row	Row labels
test.val	Values specified as PICO SYMBOLs
test.mps	The MPS file for this application
test.map	A data file that describes the sets and symbols defined by test.mod

A Customized PICO Application: A Simple Example

Command line:

gen_milp_app test.mod test.dat

Generates:

Makefile-test	generates executable
Makefile	symbolic link to Makefile-test
test_info.{h,cpp}	data structures to map ampl variables
test_milp.{h,cpp}	derived classes for branching, nodes, etc.
test_extras.{h,cpp}	for your custom methods (never deleted)

Where name.{h,cpp} means name.h and name.cpp

Staged generation

- Stage 1 runs ampl to get row, column, val files, etc
- Stage 2 generates the C++ files with derived milp classes

You can break this up:

gen_milp_app -stage 1 test.mod test.dat

Syntactic Components in test.map

The AMPL/PICO interface dynamically identifies:

- Sets: a collection a n-tuples.
  - Sets of string literals: { aa, bb, cc, dd }
  - Sets of integers: {1, 2, 3, 5, 7}
  - Sets of n-tuples defined from other sets: {(aa,1), (bb,2), (cc,3), (dd,5)}
- Symbols: any row or column label

Additional sets and data values are exported from AMPL by adding to test.mod:

PICO SYMBOLS: <symbol1> <symbol2> …

Editing the map file

- Index sets are read from ampl row/col files, not model files
  - Cannot connect similar index sets

Set0(INT);
Set1(INT);
….
Set6(INT);
TaskDone[Set0];
UnitWork[Set1]
PrecConstraint[Set2, Set3, Set4];
X[Set5, Set6];

Editing the map file

• Clean up to reflect our knowledge

```
Jobs(INT);
Time(INT);
TaskDone[Jobs];
UnitWork[Time]
PrecConstraint[Jobs, Jobs, Time];
X[Jobs, Time];
```


Protected Symbolic Access

Problem: the AMPL symbols may conflict with symbols in a the PICO classes.

Solution:

• Access AMPL symbols *only* through the `info()` method
  Use info→p(j) instead of p(j)

• Enable protection by
  – Compiling with the `-DPROTECTED_APPINFO` flag
  – Running gen_milp_app with the `-protected-vars` flag

Using the Customized PICO Classes

By default `gen_milp_app` generates a generic `main()` function (parallel if you have configured with mpi, serial otherwise)

The following commands generate, compile and execute the PICO classes

```
gen_milp_app test.mod test.dat
make all
test_milp test.mps
```


Overriding Parameters

All parameters have default values.

```
test_milp --help
```

Should output information about parameters you can override

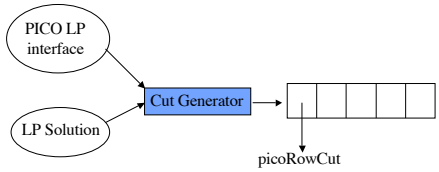
```
test_milp --debug=50 --useSets --preprocessIP=0 test.mps
```

If you don't specify a value during the override, PICO will assume 1

• Fine for binary parameters

• Parameters can have any type (probably effectively any built-in type)

Generating Cutting Planes



- Conforms to Open Source Initiative (OSI) conventions
- Uses (and will contribute to) Cut Generation Library (CGL)
- Cuts are sparse (you have to build them)

Sandia National Laboratories

ACRO

- acro-pico gives most of the pieces you need: pico, utilib, soplex, COIN
- Configuration facility
- For this workshop, you can check out from a local cvs repository
- Example configuration

```
Configure --site=lafayette --without-mpi --with-clp --without-cplex --with-debugging
```

Sets up makfiles that link properly to local MPI libraries, cplex, etc

Sandia National Laboratories

Comparison with ABACUS

A Branch And CUt System
Variables and Constraints are C++ classes.
Variables must implement

```
double coeff(CONSTRAINT *c);
```

Constraints must implement

```
double coeff(VARIABLE *v);
```

Flexible, reasonably quick development, but

- User manages mapping of individual variables to linear ordering
- Cutting planes are accessed densely
- Have to compute the coefficients of all constraints, not just cuts

Sandia National Laboratories