Alternatives for Scripting in the AMPL Modeling Language

Robert Fourer

AMPL Optimization
www.ampl.com — 773-336-AMPL

Industrial Engineering & Management Sciences, Northwestern University

4th INFORMS Optimization Society Conference
Coral Gables, Florida — 24-26 February, 2012
Session SB01, Software for Optimization Modeling
Topics

1: Parametric analysis
2: Solution generation
   a: via cuts
   b: via solver
3: Heuristic optimization
4: Pattern generation
5: Decomposition

Scripts in practice . . .
Prospective improvements . . .
1: Parametric Analysis

Multi-period production

- Max total profit
- Production, inventory, sales variables
- Limited production time per period

Try different time availabilities in period 3

- Step availability and re-solve
  - until dual is zero (constraint is slack)
- Record results
  - distinct dual values
  - corresponding objective values
Parametric

Model (sets, parameters, variables)

```plaintext
set PROD;       # products
param T > 0;    # number of weeks
param rate {PROD} > 0;          # tons per hour produced
param inv0 {PROD} >= 0;        # initial inventory
param avail {1..T} >= 0;        # hours available in week
param market {PROD,1..T} >= 0;  # limit on tons sold in week
param prodcost {PROD} >= 0;    # cost per ton produced
param invcost {PROD} >= 0;      # carrying cost/ton of inventory
param revenue {PROD,1..T} >= 0;   # revenue per ton sold

var Make {PROD,1..T} >= 0;      # tons produced
var Inv {PROD,0..T} >= 0;       # tons inventoried
var Sell {p in PROD, t in 1..T} >= 0, <= market[p,t]; # tons sold
```
Parametric

Model (objective, constraints)

```
maximize Total_Profit:
    sum {p in PROD, t in 1..T} (revenue[p,t]*Sell[p,t] -
    prodcost[p]*Make[p,t] - invcost[p]*Inv[p,t]);

subject to Time {t in 1..T}:
    sum {p in PROD} (1/rate[p]) * Make[p,t] <= avail[t];

subject to Init_Inv {p in PROD}:
    Inv[p,0] = inv0[p];

subject to Balance {p in PROD, t in 1..T}:
    Make[p,t] + Inv[p,t-1] = Sell[p,t] + Inv[p,t];
```
Parametric

Data

```
param T := 4;
set PROD := bands coils;
param avail := 1 40 2 40 3 32 4 40 ;
param rate := bands 200 coils 140 ;
param inv0 := bands 10 coils 0 ;
param prodcost := bands 10 coils 11 ;
param invcost := bands 2.5 coils 3 ;
param revenue: 1 2 3 4 :=
   bands 25 26 27 27
   coils 30 35 37 39 ;
param market: 1 2 3 4 :=
   bands 6000 6000 4000 6500
   coils 4000 2500 3500 4200 ;
```
Parametric

Script

```AMPL
set AVAIL default {};  
param avail_obj {AVAIL};  
param avail_dual {AVAIL};

let avail[3] := 1;  
param avail_step = 1;  
param previous_dual default Infinity;

repeat while previous_dual > 0 {
    solve;
    if time[3].dual < previous_dual then {
        let AVAIL := AVAIL union {avail[3]};  
        let avail_obj[avail[3]] := total_profit;  
        let avail_dual[avail[3]] := time[3].dual;  
        let previous_dual := time[3].dual;
    }
}
```
Parametric

Results

```ampl
ampl: include steelTparam.run;

ampl: display avail_obj, avail_dual;

:   avail_obj   avail_dual  :=
1   404616      3620
23  484233      3500
26  494633      3400
45  559233      2980
68  626283      0
;
```
Parametric: Observations

Parameters are true objects

- Assign new value to param avail[3]
- Problem instance changes accordingly

Sets are true data

- Assign new value to set AVAIL
  - let AVAIL := AVAIL union {avail[3]};
- All indexed entities change accordingly
2a: Solution Generation via Cuts

Workforce planning

- Cover demands for workers
  - Each “shift” requires a certain number of employees
  - Each employee works a certain “schedule” of shifts

- Satisfy scheduling rules
  - Only “valid” schedules from given list may be used
  - Each schedule that is used at all must be worked by at least ?? employees

- Minimize total workers needed
  - Which schedules should be used?
  - How many employees should work each schedule?

Generate alternative optimal solutions

- Save & display each shift schedule
Solutions \textit{via Cuts}

\textit{Model (sets, parameters)}

\begin{verbatim}
set SHIFTS;               # shifts
param Nsched;             # number of schedules;
set SCHEDS = 1..Nsched;   # set of schedules
set SHIFT_LIST {SCHEDS} within SHIFTS;
param rate {SCHEDS} >= 0; # pay rates
param required {SHIFTS} >= 0; # staffing requirements
param least_assign >= 0;   # min workers on any schedule used
\end{verbatim}
Solutions via Cuts

Model (variables, objective, constraints)

var Work {SCHEDS} >= 0 integer;
var Use {SCHEDS} >= 0 binary;

minimize Total_Cost:
    sum {j in SCHEDS} rate[j] * Work[j];

subject to Shift_Needs {i in SHIFTS}:
    sum {j in SCHEDS: i in SHIFT_LIST[j]} Work[j] >= required[i];

subject to Least_Use1 {j in SCHEDS}:
    least_assign * Use[j] <= Work[j];

subject to Least_Use2 {j in SCHEDS}:
    Work[j] <= (max {i in SHIFT_LIST[j]} required[i]) * Use[j];
Solutions via Cuts

Data

```ampl
set SHIFTS := Mon1 Tue1 Wed1 Thu1 Fri1 Sat1
             Mon2 Tue2 Wed2 Thu2 Fri2 Sat2
             Mon3 Tue3 Wed3 Thu3 Fri3 ;

param Nsched := 126 ;
set SHIFT_LIST[1] := Mon1 Tue1 Wed1 Thu1 Fri1 ;
set SHIFT_LIST[2] := Mon1 Tue1 Wed1 Thu1 Fri2 ;
set SHIFT_LIST[3] := Mon1 Tue1 Wed1 Thu1 Fri3 ;
set SHIFT_LIST[4] := Mon1 Tue1 Wed1 Thu1 Sat1 ;
set SHIFT_LIST[5] := Mon1 Tue1 Wed1 Thu1 Sat2 ; .......

param required :=
                 Mon1 100  Mon2 78  Mon3 52
                 Tue1 100  Tue2 78  Tue3 52
                 Wed1 100  Wed2 78  Wed3 52
                 Thu1 100  Thu2 78  Thu3 52
                 Fri1 100  Fri2 78  Fri3 52
                 Sat1 100  Sat2 78 ;
```
Solutions via Cuts

Script

```plaintext
param nSols default 0;
param maxSols = 20;

set USED {1..nSols} within SCHEDS;

subject to exclude {k in 1..nSols}:
    sum {j in USED[k]} (1-Use[j]) +
    sum {j in SCHEDS diff USED[k]} Use[j] >= 1;

repeat {
    solve;
    display Work;
    let nSols := nSols + 1;
    let USED[nSols] := {j in SCHEDS: Use[j] > .5};
} until nSols = maxSols;
```
Solutions \textit{via Cuts}

\textbf{Results}

\begin{verbatim}
AMPL: include scheds.run

Gurobi 4.0.1: optimal solution; \textbf{objective 266}
857 simplex iterations
29 branch-and-cut nodes

Work [*] :=
1 21  21 36  52  7  89 29  94  7  109 16  124 36
3  7  37 29  71 13  91 16  95 13  116 36 ;

Gurobi 4.0.1: optimal solution; \textbf{objective 266}
1368 simplex iterations
59 branch-and-cut nodes

Work [*] :=
1  9  17  9  38  7  59 21  75 36  94  7  114  8  124 35
4 20  33 27  56  7  71 27  86  8  107  9  116 36 ;
\end{verbatim}
Solutions via Cuts

Results (continued)

Gurobi 4.0.1: optimal solution; objective 266
982 simplex iterations
57 branch-and-cut nodes

Work [*] :=
2 28 16 8 38 18 75 34 86 8 108 8 115 16 121 36
7 18 28 10 70 18 85 18 97 18 109 10 116 18 ;

Gurobi 4.0.1: optimal solution; objective 266
144 simplex iterations

Work [*] :=
2 29 16 7 76 36 88 29 106 16 116 7 123 7
7 36 70 28 85 7 97 7 109 29 121 21 126 7 ;

Gurobi 4.0.1: optimal solution; objective 266
122 simplex iterations

Work [*] :=
2 15 16 20 70 15 85 21 106 16 116 21 123 21
7 36 53 14 76 36 97 21 109 15 121 8 126 7 ;
Solutions via Cuts: Observations

Same expressions describe sets and indexing

- Index a summation
  * ... sum {j in SCHEDS diff USED[k]} Use[j] >= 1;
- Assign a value to a set
  * let USED[nSols] := {j in SCHEDS: Use[j] > .5};

New cuts defined automatically

- Index cuts over a set
  * subject to exclude {k in 1..nSols}: ...
- Add a cut by expanding the set
  * let nSols := nSols + 1;
2b: Solution Generation via Solver

Same model

Ask solver to return multiple solutions

- Set options
- Get all results from one “solve”
Solutions via Solver

Script

```plaintext
option solver cplex;

option cplex_options "poolstub=sched poolcapacity=20 \ poolintensity=4 poolgap=0"

solve;

for {i in 1..Current.npool} {
    solution ("sched" & i & ".sol");
    display Work;
}
```
Solutions via Solver

Results

ampl: include schedsPool.run;

CPLEX 12.2.0.2: poolstub=sched
poolcapacity=20
populate=1
poolintensity=4
poolgap=0

CPLEX 12.2.0.2: optimal integer solution; objective 266
464 MIP simplex iterations
26 branch-and-bound nodes

Wrote 20 solutions in solution pool
to files sched1.sol ... sched20.sol.

Solution pool member 1 (of 20); objective 266

Work [*] :=
    1 15    7 14    27 7    70 29    78 29   103 7   115 14
   5 21   11 7    51 7    71 21   87 21   106 38   121 36 ;
Solutions via Solver

Results (continued)

Solution pool member 2 (of 20); objective 266

Work [*] :=
1 7 5 8 18 7 70 29 78 36 87 14 115 14 121 36
2 28 7 14 65 7 72 7 83 21 106 31 116 7 ;

Solution pool member 3 (of 20); objective 266

Work [*] :=
5 21 29 13 51 7 71 34 98 7 115 13
7 15 35 8 64 8 78 16 101 13 116 15
21 7 40 13 70 8 83 8 106 24 121 36 ;

Solution pool member 4 (of 20); objective 266

Work [*] :=
2 7 11 7 40 7 71 29 87 15 106 31 121 28
5 22 23 8 64 7 78 13 101 8 115 14 126 7
7 14 29 14 70 14 83 7 102 7 116 7 ;
Solutions via Solver: Observations

Filenames can be formed dynamically

- Write a (string expression)
- Numbers are automatically converted
  * solution ("sched" & i & ".sol");
3: Heuristic Optimization

Same model

Difficult instances

- Set `least_assign` to a “hard” value
- Get a very good solution quickly
Heuristic

*Hard case:* \texttt{least_assign = 19}

```ampl
ampl: model sched1.mod;
ampl: data sched.dat;
ampl: let least_assign := 19;
ampl: option solver cplex;
ampl: solve;

CPLEX 12.2.0.2: optimal integer solution; objective 269
635574195 MIP simplex iterations
86400919 branch-and-bound nodes

ampl: option omit_zero_rows 1, display_1col 0;
ampl: display Work;

Work [*] :=
    4 22   16 39   55 39   78 39   101 39   106 52   122 39
;

... 94.8 minutes
```
Heuristic

*Alternative, indirect approach*

- Step 1: Relax integrality of Work variables
  Solve for zero-one Use variables
- Step 2: Fix Use variables
  Solve for integer Work variables

... not necessarily optimal, but ...
Heuristic

Script

```AMPL
model sched1.mod;
data sched.dat;
let least_assign := 19;

let {j in SCHEDS} Work[j].relax := 1;
solve;

fix {j in SCHEDS} Use[j];
let {j in SCHEDS} Work[j].relax := 0;
solve;
```
Heuristic

Results

```
ampl: include sched1-fix.run;

CPLEX 12.2.0.2: optimal integer solution; objective 268.5
32630436 MIP simplex iterations
2199508 branch-and-bound nodes

Work [*] :=
  1 24    32 19    80 19.5   107 33    126 19.5
  3 19    66 19    90 19.5   109 19
  10 19   72 19.5   105 19.5   121 19;

CPLEX 12.2.0.2: optimal integer solution; objective 269
2 MIP simplex iterations
0 branch-and-bound nodes

Work [*] :=
  1 24    10 19    66 19    80 19    105 20   109 19    126 20
  3 19    32 19    72 19    90 20    107 33   121 19;

... 2.85 minutes
```
Heuristic: Observations

Models can be changed dynamically

- Retain model-like syntax
- Execute model-related commands
  ```
  fix {j in SCHEDS} Use[j];
  ```
- Assign values to properties of model components
  ```
  let {j in SCHEDS} Work[j].relax := 1;
  ```
4: Pattern Generation

Roll cutting
- Min rolls cut (or material wasted)
- Decide number of each pattern to cut
- Meet demands for each ordered width

Generate cutting patterns
- Read general model
- Read data: demands, raw width
- Compute data: all usable patterns
- Solve problem instance
Pattern Generation

Model

```AMPL
param roll_width > 0;
set WIDTHS ordered by reversed Reals;
param orders {WIDTHS} > 0;
param maxPAT integer >= 0;
param nPAT integer >= 0, <= maxPAT;
param nbr {WIDTHS,1..maxPAT} integer >= 0;
var Cut {1..nPAT} integer >= 0;
minimize Number:
    sum {j in 1..nPAT} Cut[j];
subj to Fulfill {i in WIDTHS}:
    sum {j in 1..nPAT} nbr[i,j] * Cut[j] >= orders[i];
```
Pattern Generation

Data

\[
\begin{align*}
\text{param roll_width := 90 ;} \\
\text{param: WIDTHS: orders :=} \\
60 & \quad 3 \\
30 & \quad 21 \\
25.5 & \quad 94 \\
20 & \quad 50 \\
17.25 & \quad 288 \\
15 & \quad 178 \\
12.75 & \quad 112 \\
10 & \quad 144 \\
\end{align*}
\]
Pattern Generation

Script (initialize)

```ampl
model cutPAT.mod;
data ChvatalD.dat;
model;
param curr_sum >= 0;
param curr_width > 0;
param pattern {WIDTHS} integer >= 0;
let maxPAT := 100000000;
let nPAT := 0;
let curr_sum := 0;
let curr_width := first(WIDTHS);
let {w in WIDTHS} pattern[w] := 0;
```
Pattern Generation

Script (loop)

repeat {
    if curr_sum + curr_width <= roll_width then {
        let pattern[curr_width] := floor((roll_width-curr_sum)/curr_width);
        let curr_sum := curr_sum + pattern[curr_width] * curr_width;
    }
    if curr_width != last(WIDTHS) then
        let curr_width := next(curr_width,WIDTHS);
    else {
        let nPAT := nPAT + 1;
        let {w in WIDTHS} nbr[w,nPAT] := pattern[w];
        let curr_sum := curr_sum - pattern[last(WIDTHS)] * last(WIDTHS);
        let pattern[last(WIDTHS)] := 0;
        let curr_width := min {w in WIDTHS: pattern[w] > 0} w;
        if curr_width < Infinity then {
            let curr_sum := curr_sum - curr_width;
            let pattern[curr_width] := pattern[curr_width] - 1;
            let curr_width := next(curr_width,WIDTHS);
        }
        else break;
    }
}
Pattern Generation

Script \textit{(solve, report)}

\begin{verbatim}
option solver gurobi;
solve;
printf "\n%5i patterns, %3i rolls", nPAT, sum {j in 1..nPAT} Cut[j];
printf "\n\n Cut \n"
printf {j in 1..nPAT: Cut[j] > 0}: "%3i", Cut[j];
printf "\n\n"
for {i in WIDTHS} {
    printf "%7.2f \n", i;
    printf {j in 1..nPAT: Cut[j] > 0}: "%3i", nbr[i,j];
    printf "\n";
}
printf \nWASTE = %5.2f\%\n\n",
100 * (1 - (sum {i in WIDTHS} i * orders[i]) / (roll_width * Number));
\end{verbatim}
### Pattern Generation

#### Results

```ampl
ampl: include cutPatEnum.run

Gurobi 4.6.1: optimal solution; objective 164
15 simplex iterations

290 patterns, 164 rolls

<table>
<thead>
<tr>
<th>Cut</th>
<th>3</th>
<th>7</th>
<th>50</th>
<th>44</th>
<th>17</th>
<th>25</th>
<th>2</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>60.00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30.00</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>25.50</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>20.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>17.25</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15.00</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12.75</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>10.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

WASTE = 0.32%
```
Pattern Generation

Data 2

```
param roll_width := 349 ;
param: WIDTHS: orders :=
    28.75    7
    33.75    23
    34.75    23
    37.75    31
    38.75    10
    39.75    39
    40.75    58
    41.75    47
    42.25    19
    44.75    13
    45.75    26 ;
```
Pattern Generation

Results 2

```
AMPL: include cutPatEnum.run

Gurobi 4.6.1: optimal solution; objective 34
291 simplex iterations

54508 patterns, 34 rolls

Cut     8 1 1 1 3 1 1 1 1 2 7 2 3 1 1
45.75   3 2 0 0 0 0 0 0 0 0 0 0 0 0 0
44.75   1 2 2 1 0 0 0 0 0 0 0 0 0 0 0
42.25   0 2 0 0 4 2 2 1 0 0 0 0 0 0 0
41.75   4 2 0 2 0 0 0 0 2 1 1 0 0 0 0
40.75   0 0 4 4 1 4 3 0 2 3 1 6 3 2 2
39.75   0 0 0 0 0 0 0 0 2 0 0 5 0 0 2 0
38.75   0 0 1 0 0 0 0 0 4 0 0 0 0 2 3
37.75   0 0 0 0 0 0 0 1 0 0 4 0 0 6 2 4
34.75   0 0 0 0 4 0 3 1 0 0 0 3 0 1 0
33.75   0 0 0 0 0 3 0 4 0 1 2 0 0 0 0
28.75   0 0 2 2 0 0 0 2 1 0 0 0 0 0 0

WASTE = 0.69%```
Pattern Generation

Data 3

```plaintext
param roll_width := 172 ;
param: WIDTHS: orders :=
  25.000  5
  24.750  73
  18.000  14
  17.500  4
  15.500  23
  15.375  5
  13.875  29
  12.500  87
  12.250  9
  12.000  31
  10.250  6
  10.125  14
  10.000  43
  8.750   15
  8.500   21
  7.750   5 ;
```
Pattern Generation

Results 3 (using a subset of patterns)

ampl: include cutPatEnum.run

Gurobi 4.6.1: optimal solution; objective 33
722 simplex iterations
40 branch-and-cut nodes
273380 patterns, 33 rolls

<table>
<thead>
<tr>
<th>Cut</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>4</th>
<th>4</th>
<th>4</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>25.00</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>24.75</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>17.50</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>.......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10.12</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10.00</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>8.75</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>8.50</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7.75</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

WASTE = 0.62%
Pattern Generation: Observations

Parameters can serve as script variables

- Declare as in model
  ```
  * param pattern {WIDTHS} integer >= 0;
  ```
- Use in algorithm
  ```
  * let pattern[curr_width] := pattern[curr_width] - 1;
  ```
- Assign to model parameters
  ```
  * let {w in WIDTHS} nbr[w,nPAT] := pattern[w];
  ```

Scripts are easy to modify

- Store only every 100th pattern found
  ```
  * if nPAT mod 100 = 0 then
    let {w in WIDTHS} nbr[w,nPAT/100] := pattern[w];
  ```
5: Decomposition

**Stochastic nonlinear location-transportation**

- Min expected total cost
  - Nonlinear construction costs at origins
  - Linear transportation costs from origins to destinations
- Stochastic demands with recourse
  - Decide what to build
  - Observe demands and decide what to ship

**Solve by Benders decomposition**

- Nonlinear master problem
- Linear subproblem for each scenario
## Decomposition

### Original model (sets, parameters, variables)

<table>
<thead>
<tr>
<th>Set Description</th>
<th>Declaration</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHSE</td>
<td>set WHSE; # shipment origins (warehouses)</td>
<td></td>
</tr>
<tr>
<td>STOR</td>
<td>set STOR; # shipment destinations (stores)</td>
<td></td>
</tr>
<tr>
<td>Build cost</td>
<td>param build_cost {WHSE} &gt; 0; # costs per unit to build warehouse</td>
<td></td>
</tr>
<tr>
<td>Build limit</td>
<td>param build_limit {WHSE} &gt; 0; # limits on units shipped</td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td>var Build {i in WHSE} &gt;= 0, &lt;= .9999 * build_limit[i]; # capacities of warehouses to build</td>
<td></td>
</tr>
<tr>
<td>SCEN</td>
<td>set SCEN; # demand scenarios</td>
<td></td>
</tr>
<tr>
<td>Probabilities</td>
<td>param prob {SCEN} &gt;= 0, &lt;= 1; # probabilities of scenarios</td>
<td></td>
</tr>
<tr>
<td>Demand</td>
<td>param demand {STOR,SCEN} &gt;= 0; # amounts required at stores</td>
<td></td>
</tr>
<tr>
<td>Ship costs</td>
<td>param ship_cost {WHSE,STOR} &gt;= 0; # shipment costs per unit</td>
<td></td>
</tr>
<tr>
<td>Ship</td>
<td>var Ship {WHSE,STOR,SCEN} &gt;= 0; # amounts to be shipped</td>
<td></td>
</tr>
</tbody>
</table>
Decomposition

Original model (objective, constraints)

\[
\begin{align*}
\text{minimize Total Cost:} & \\
& \sum \{i \in \text{WHSE}\} \text{ build_cost}[i] \times \text{Build}[i] / (1 - \text{Build}[i]/\text{build_limit}[i]) + \\
& \sum \{s \in \text{SCEN}\} \text{ prob}[s] \times \\
& \quad \sum \{i \in \text{WHSE}, j \in \text{STOR}\} \text{ ship_cost}[i,j] \times \text{Ship}[i,j,s]; \\
\text{subj to Supply \{i \in WHSE, s in SCEN\}:} & \\
& \sum \{j \in \text{STOR}\} \text{ Ship}[i,j,s] \leq \text{Build}[i]; \\
\text{subj to Demand \{j \in STOR, s in SCEN\}:} & \\
& \sum \{i \in \text{WHSE}\} \text{ Ship}[i,j,s] = \text{demand}[j,s];
\end{align*}
\]
Decomposition

Sub model (sets, parameters, variables)

set WHSE;  # shipment origins (warehouses)
set STOR;  # shipment destinations (stores)
param build {i in WHSE} >= 0, <= .9999 * build_limit[i];
            # capacities of warehouses built
set SCEN;  # demand scenarios
param prob {SCEN} >= 0, <= 1;    # probabilities of scenarios
param demand {STOR,SCEN} >= 0;  # amounts required at stores
param ship_cost {WHSE,STOR} >= 0;  # shipment costs per unit
var Ship {WHSE,STOR,SCEN} >= 0;  # amounts to be shipped
Decomposition

Sub model (objective, constraints)

```AMPL
param S symbolic in SCEN;

minimize Scen_Ship_Cost:
    prob[S] * sum {i in WHSE, j in STOR} ship_cost[i,j] * Ship[i,j];

subj to Supply {i in WHSE}:
    sum {j in STOR} Ship[i,j] <= build[i];

subj to Demand {j in STOR}:
    sum {i in WHSE} Ship[i,j] = demand[j,S];
```
Decomposition

Master model (sets, parameters, variables)

```plaintext
param build_cost {WHSE} > 0;  # costs per unit to build warehouse
param build_limit {WHSE} > 0;  # limits on units shipped
var Build {i in WHSE} >= 0, <= 0.9999 * build_limit[i];  
   # capacities of warehouses to build

param nCUT >= 0 integer;

param cut_type {SCEN,1..nCUT} symbolic
   within {"feas","infeas","none"};

param supply_price {WHSE,SCEN,1..nCUT} <= 0.000001;
param demand_price {STOR,SCEN,1..nCUT};

var Max_Exp_Ship_Cost {SCEN} >= 0;
```
Decomposition

Master model (objective, constraints)

```
minimize Expected_Total_Cost:
    sum {i in WHSE}
    build_cost[i] * Build[i] / (1 - Build[i]/build_limit[i]) +
    sum {s in SCEN} Max_Exp_Ship_Cost[s];

subj to Cut_Defn {s in SCEN, k in 1..nCUT: cut_type[s,k] != "none"}:
    if cut_type[s,k] = "feas" then Max_Exp_Ship_Cost[s] else 0 >=
    sum {i in WHSE} supply_price[i,s,k] * Build[i] +
    sum {j in STOR} demand_price[j,s,k] * demand[j,s];
```
Decomposition

Script (initialization)

```AMPL
model stbenders.mod;
data stnlntrnloc.dat;
suffix dunbdd;
option presolve 0;

problem Sub: Ship, Scen_Ship_Cost, Supply, Demand;
   option solver cplex;
   option cplex_options 'primal presolve 0';

problem Master: Build, Max_Exp_Ship_Cost, Exp_Total_Cost, Cut_Defn;
   option solver minos;

let nCUT := 0;

param GAP default Infinity;
param RELGAP default Infinity;
param Exp_Ship_Cost;
```
Decomposition

**Script (iteration)**

```plaintext
repeat {
    solve Master;
    let {i in WHSE} build[i] := Build[i];
    let Exp_Ship_Cost := 0;
    let nCUT := nCUT + 1;

    for {s in SCEN} {
        let S := s;
        solve Sub;

        ... generate a cut ...
    }

    if forall {s in SCEN} cut_type[s,nCUT] != "infeas" then {
        let GAP := min (GAP,
                      Exp_Ship_Cost - sum {s in SCEN} Max_Exp_Ship_Cost[s]);
        let RELGAP := 100 * GAP / Expected_Total_Cost;
    }
}

} until RELGAP <= .000001;
```
Decomposition

Script (cut generation)

```plaintext
for {s in SCEN} {
    let S := s;
    solve Sub;
    if Sub.result = "solved" then {
        let Exp_Ship_Cost := Exp_Ship_Cost + Scen_Ship_Cost;
        if Scen_Ship_Cost > Max_Exp_Ship_Cost[s] + 0.00001 then {
            let cut_type[s,nCUT] := "feas";
            let {i in WHSE} supply_price[i,s,nCUT] := Supply[i].dual;
            let {j in STOR} demand_price[j,s,nCUT] := Demand[j].dual;
        }
        else let cut_type[s,nCUT] := "none";
    }
    else if Sub.result = "infeasible" then {
        let cut_type[s,nCUT] := "infeas";
        let {i in WHSE} supply_price[i,s,nCUT] := Supply[i].dunbdd;
        let {j in STOR} demand_price[j,s,nCUT] := Demand[j].dunbdd;
    }
}
```
Decomposition

Results

```plaintext
ampl: include stbenders.run;

MASTER PROBLEM 1: 0.000000
SUB-PROBLEM 1  low: infeasible
SUB-PROBLEM 1  mid: infeasible
SUB-PROBLEM 1 high: infeasible
MASTER PROBLEM 2: 267806.267806
SUB-PROBLEM 2  low: 1235839.514234
SUB-PROBLEM 2  mid: 1030969.048921
SUB-PROBLEM 2 high: infeasible
MASTER PROBLEM 3: 718918.236014
SUB-PROBLEM 3  low: 1019699.661119
SUB-PROBLEM 3  mid: 802846.293052
SUB-PROBLEM 3 high: 695402.974379
GAP = 2517948.928551, RELGAP = 350.241349%
```
Decomposition

Results (continued)

<table>
<thead>
<tr>
<th>MASTER PROBLEM 4: 2606868.719958</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB-PROBLEM 4 low: 1044931.784272</td>
</tr>
<tr>
<td>SUB-PROBLEM 4 mid: 885980.640150</td>
</tr>
<tr>
<td>SUB-PROBLEM 4 high: 944581.118758</td>
</tr>
<tr>
<td>GAP = 749765.716399, RELGAP = 28.761161%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MASTER PROBLEM 5: 2685773.838398</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB-PROBLEM 5 low: 1028785.052062</td>
</tr>
<tr>
<td>SUB-PROBLEM 5 mid: 815428.531237</td>
</tr>
<tr>
<td>SUB-PROBLEM 5 high: 753627.189086</td>
</tr>
<tr>
<td>GAP = 394642.837091, RELGAP = 14.693822%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MASTER PROBLEM 6: 2743483.001029</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB-PROBLEM 6 low: 1000336.408156</td>
</tr>
<tr>
<td>SUB-PROBLEM 6 mid: 785602.983289</td>
</tr>
<tr>
<td>SUB-PROBLEM 6 high: 725635.817601</td>
</tr>
<tr>
<td>GAP = 222288.965560, RELGAP = 8.102436%</td>
</tr>
</tbody>
</table>
Decomposition

Results (continued)

<table>
<thead>
<tr>
<th>MASTER PROBLEM 7: 2776187.713412</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB-PROBLEM 7 low: 986337.500000</td>
</tr>
<tr>
<td>SUB-PROBLEM 7 mid: 777708.466300</td>
</tr>
<tr>
<td>SUB-PROBLEM 7 high: 693342.659287</td>
</tr>
<tr>
<td>GAP = 59240.084058, RELGAP = 2.133864%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MASTER PROBLEM 8: 2799319.395374</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB-PROBLEM 8 low: 991426.284976</td>
</tr>
<tr>
<td>SUB-PROBLEM 8 mid: 777146.351060</td>
</tr>
<tr>
<td>SUB-PROBLEM 8 high: 704353.854398</td>
</tr>
<tr>
<td>GAP = 38198.286498, RELGAP = 1.364556%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MASTER PROBLEM 9: 2814772.778136</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUB-PROBLEM 9 low: 987556.309573</td>
</tr>
<tr>
<td>SUB-PROBLEM 9 mid: 772147.258329</td>
</tr>
<tr>
<td>SUB-PROBLEM 9 high: 696060.666966</td>
</tr>
<tr>
<td>GAP = 17658.226624, RELGAP = 0.627341%</td>
</tr>
</tbody>
</table>
Decomposition

Results (continued)

MASTER PROBLEM 10: 2818991.649514
SUB-PROBLEM 10 mid: 771853.500000
SUB-PROBLEM 10 high: 689709.131427
GAP = 2361.940101, RELGAP = 0.083787%

MASTER PROBLEM 11: 2819338.502316
SUB-PROBLEM 11 high: 692406.351318
GAP = 2361.940101, RELGAP = 0.083776%

MASTER PROBLEM 12: 2819524.204253
SUB-PROBLEM 12 high: 690478.286312
GAP = 541.528304, RELGAP = 0.019206%

MASTER PROBLEM 13: 2819736.994159
GAP = -0.000000, RELGAP = -0.000000%

OPTIMAL SOLUTION FOUND

Expected Cost = 2819736.994159
Decomposition: **Observations**

*Loops can iterate over sets*

- Solve a subproblem for each scenario
  - \* for \{s in SCEN\} \{ ... \\

*One model can represent all subproblems*

- Assign loop index \(s\) to set \(S\), then solve
  - \* let \(S := s\);
    solve Sub;

*Results of solve can be tested*

- Check whether optimization was successful
  - \* if Sub.result = "solved" then \{ ... \\
  - \* else if Sub.result = "infeasible" then \{ ...
Concluding Observations

Scripts in practice

- Large and complicated
  - Multiple files
  - Hundreds of statements
  - Millions of statements executed
- Run within broader applications

Prospective improvements

- Faster loops
- True script functions
  - Arguments and return values
  - Local sets & parameters
- More database connections
- IDE for debugging
- APIs for popular languages (C++, Java, C#, VB, Python)