

Implementation of some practical scheduling models Tales from the trenches

Erwin Kalvelagen
erwin@amsterdamoptimization.com

Amsterdam Optimization Modeling Group



Unrelated Parallel Machine Scheduling with Sequence Dependent Set-up Times

Application: scheduling jobs for
producing colored plastic pellets on
extruder lines

Plastic Pellet Production



GE Plastics, now SABIC

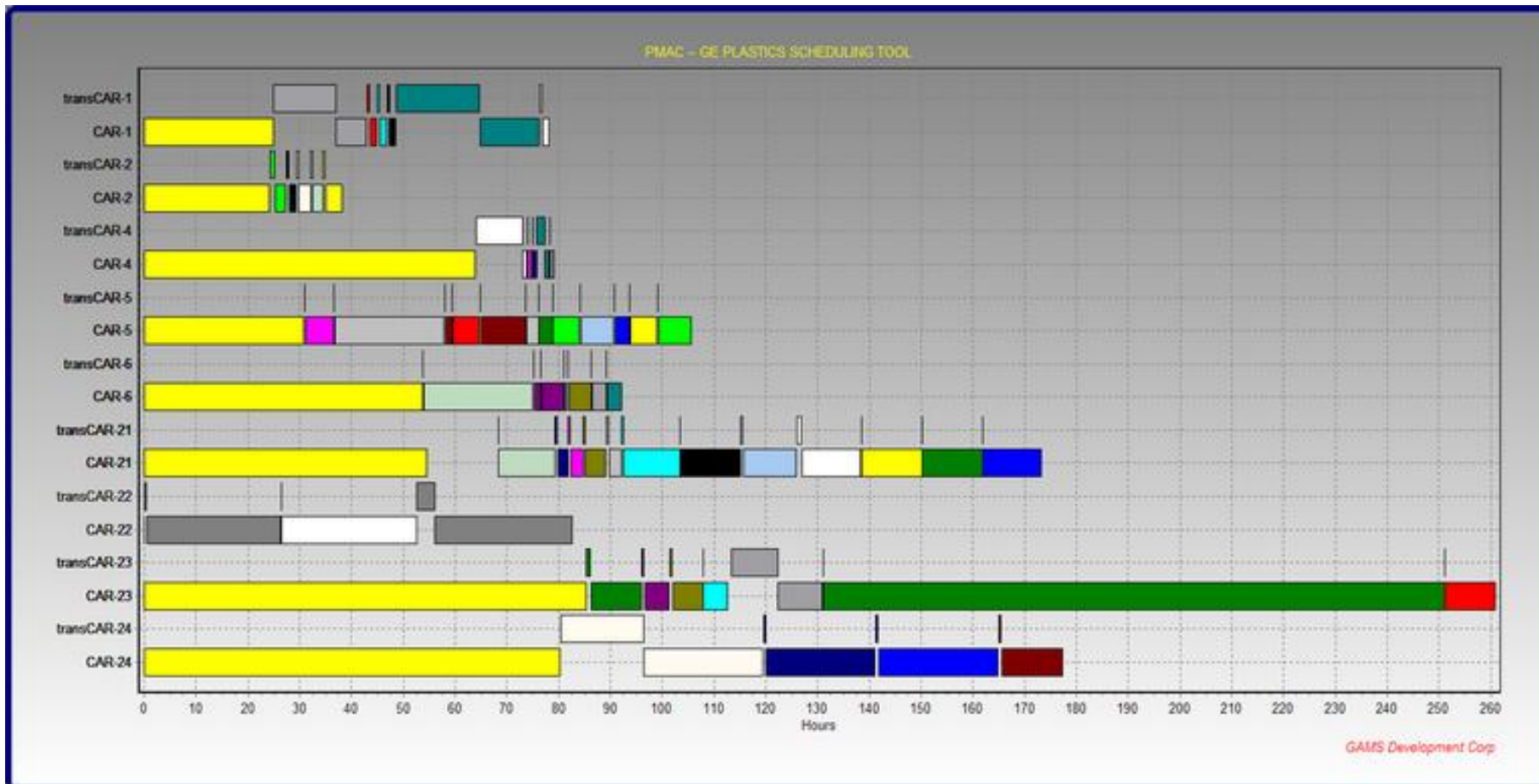
Carthagena (Spain) Plant



Similar Application

- Printing of colored paper
- Color difference determines setup time
 - Close colors are cheap
 - White → anything is cheap
 - Black → anything (except black) is expensive
 - Longer cleaning

Example



Decision variables

$$x_{i,j,k}^{order} = \begin{cases} 1 & \text{if job } i \text{ is immediately before job } j \text{ on machine } k \\ 0 & \text{otherwise} \end{cases}$$

$$x_{i,k}^{first} = \begin{cases} 1 & \text{if job } i \text{ is first job on machine } k \\ 0 & \text{otherwise} \end{cases}$$

If a job i cannot execute on machine k , we impose
 $x_{order}(i,j,k)=x_{order}(j,i,k)=x_{first}(i,k)=0$

Parallel Machine Scheduling Model

Machine assignment $x_{j,k}^{mach} = x_{j,k}^{first} + \sum_i x_{i,j,k}^{order} \quad 0 \leq x_{j,k}^{mach} \leq 1$

At most one successor $\sum_{j,k} x_{i,j,k}^{order} \leq 1$ x_{mach} is automatically integer

Schedule job once $\sum_k x_{i,k}^{mach} = 1$

Machine can start with only 1 job $\sum_i x_{i,k}^{first} \leq 1$

Proper schedule: $x_{i,k}^{mach} \geq x_{i,j,k}^{order}$ If $x_{order}(i,j,k)=1$ then $x_{mach}(i,k)=1$

Model (2)

Completion times:

$$S_j = \sum_k \left[(F_k + \Delta_{j,k}^{init} + P_{j,k}) x_{j,k}^{first} + \sum_i (S_i + \Delta_{i,j,k} + P_{i,k}) x_{i,j,k}^{order} \right]$$

But this is nonlinear: multiplication of $S(i) * x_{order}(i,j,k)$

Bounds:

$$S_j^{early} \leq S_j \leq S_j^{duedate}$$

Later on in the project this was relaxed to become a soft constraint with penalties. We use combination of MIN SUM and MIN NUM Deviations (if you don't know make it an option).

Linearization

$$S'_{i,j,k} \leq Mx_{i,j,k}^{order}$$

$$S'_{i,j,k} \leq S_i + \Delta_{i,j,k} + P_{j,k}$$

$$S'_{i,j,k} \geq (S_i + \Delta_{i,j,k} + P_{j,k}) - M(1 - x_{i,j,k}^{order})$$

$$S''_{j,k} = (F_k + \Delta_{j,k}^{init} + I_{j,k} + P_{j,k})x_{j,k}^{first} + \sum_i S'_{i,j,k}$$

$$S_j = \sum_k S''_{j,k}$$

Requires additional continuous variables and equations, but we still can use a MIP solver instead of an MINLP solver.

Objective

Minimize total transition time:

$$\textit{Min} \sum_{j,k} \left[\left(\Delta_{j,k}^{init} + I_{j,k} \right) x_{j,k}^{init} + \sum_i \Delta_{i,j,k} x_{i,j,k}^{order} \right]$$

GAMS/Cplex

- Complex MIP models benefit enormously from use of a modeling language
 - Quick implementation of ideas
 - Allows for rapid implementation of alternative formulations and experiments
 - Allows for quick implementation of heuristics/tricks
 - Very difficult to do in normal programming language (say C) + solver API
 - Code is “carved in stone” too quickly

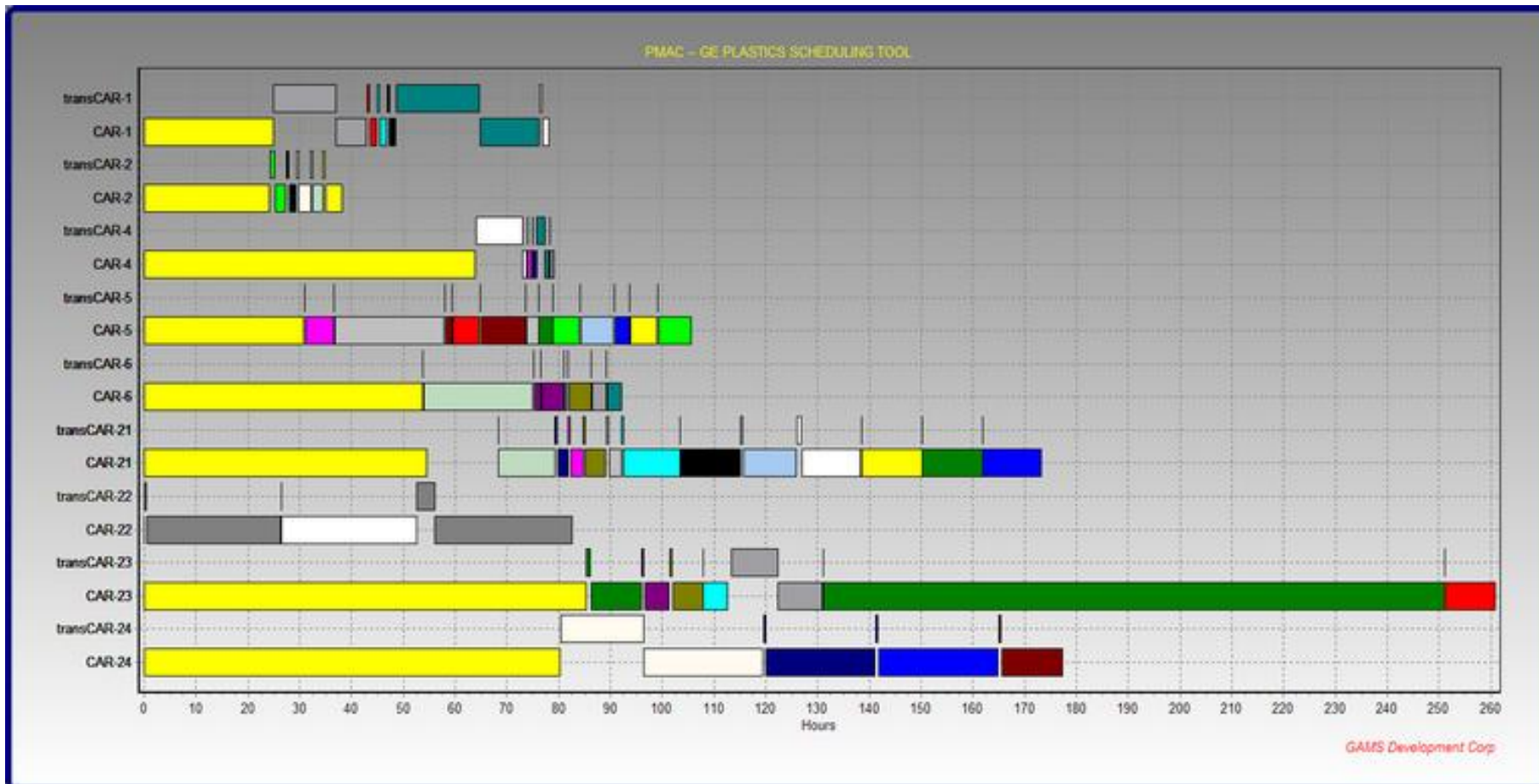
All Sub-Models Improve existing solutions

1. Check: Use schedule in input data
 - Fix variables xfirst, xorder, according to the data and solve quickly
 2. Line by line model: Improve solution by solving model per machine
 - Unfix only for within machine (k solves)
 3. Big one: Improve solution by solving the complete model
 - Unfix all variables (this model is the bottleneck)
 4. Clean up: Improve solution by solving model per machine
 - Unfix only for within machine (k solves)
- All steps use the same model, just fix/unfix vars.
 - Let presolver kill unneeded equ's.
 - GAMS generates these models fast enough.

Cplex

- Heavily use of advanced Cplex option: MIPSTART
 - Advantage: if a submodel fails, it should not destroy the whole run, just not improve the current solution
- Additional polishing step fits nicely in this scheme and often produced good improvements

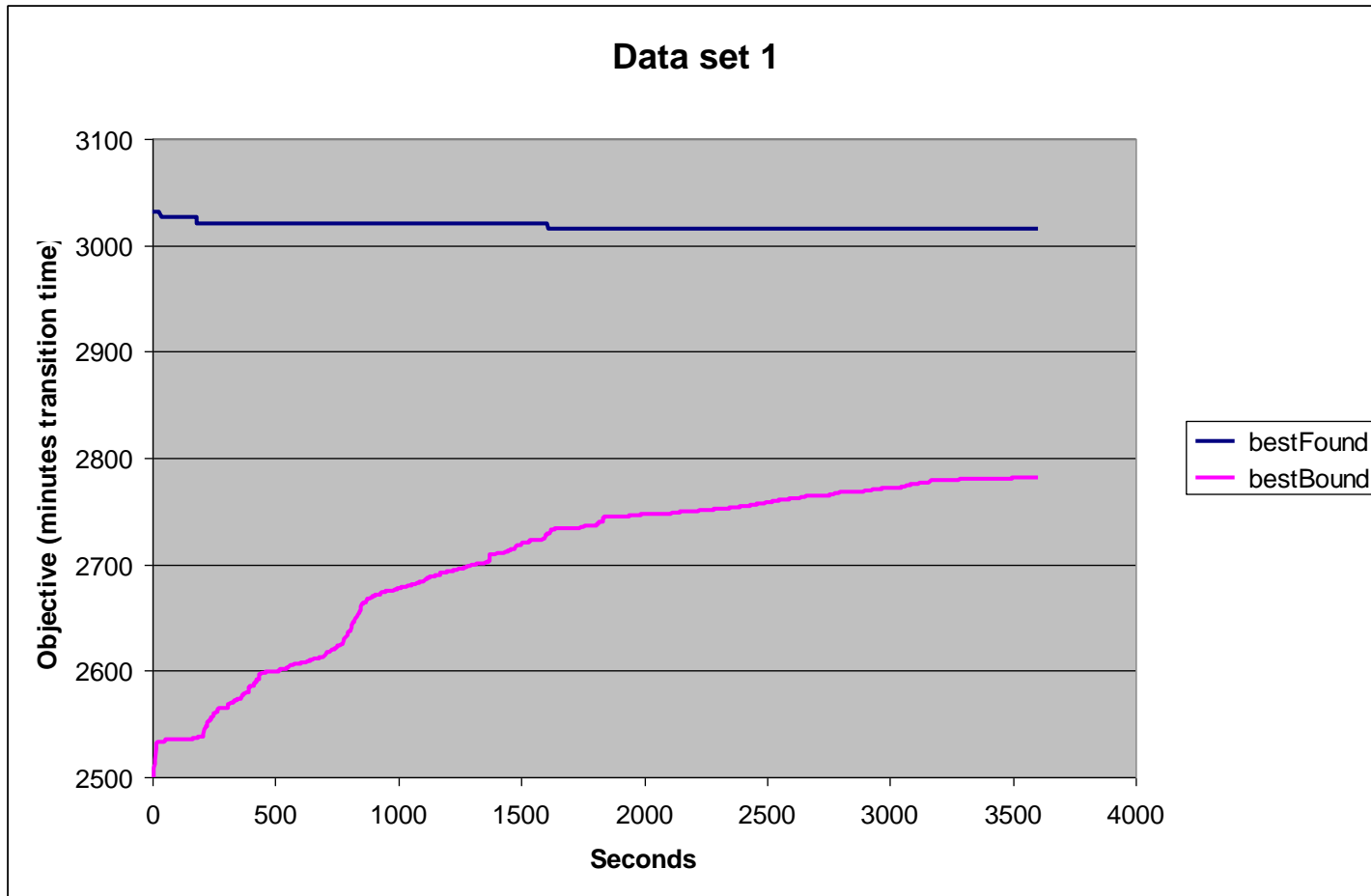
Example



Data set 1, 47 jobs

- Fixed model: obj=3723,time=0.1 sec
- Solve for each line, keep rest fixed (optcr=2%)
 - Obj=3555, time=0.6 sec
 - Obj=3448, time=0.3 sec
 - Obj=3448, time=0.3 sec
 - Obj=3032, time=0.3 sec
 - Obj=3032, time=0.3 sec
 - Obj=3032, time=0.3 sec
- Solve complete model: obj=3015, time=3600 sec, gap=7.7%
- No improvement afterward

Data set 1, Bounds



Data set 1, Size of model

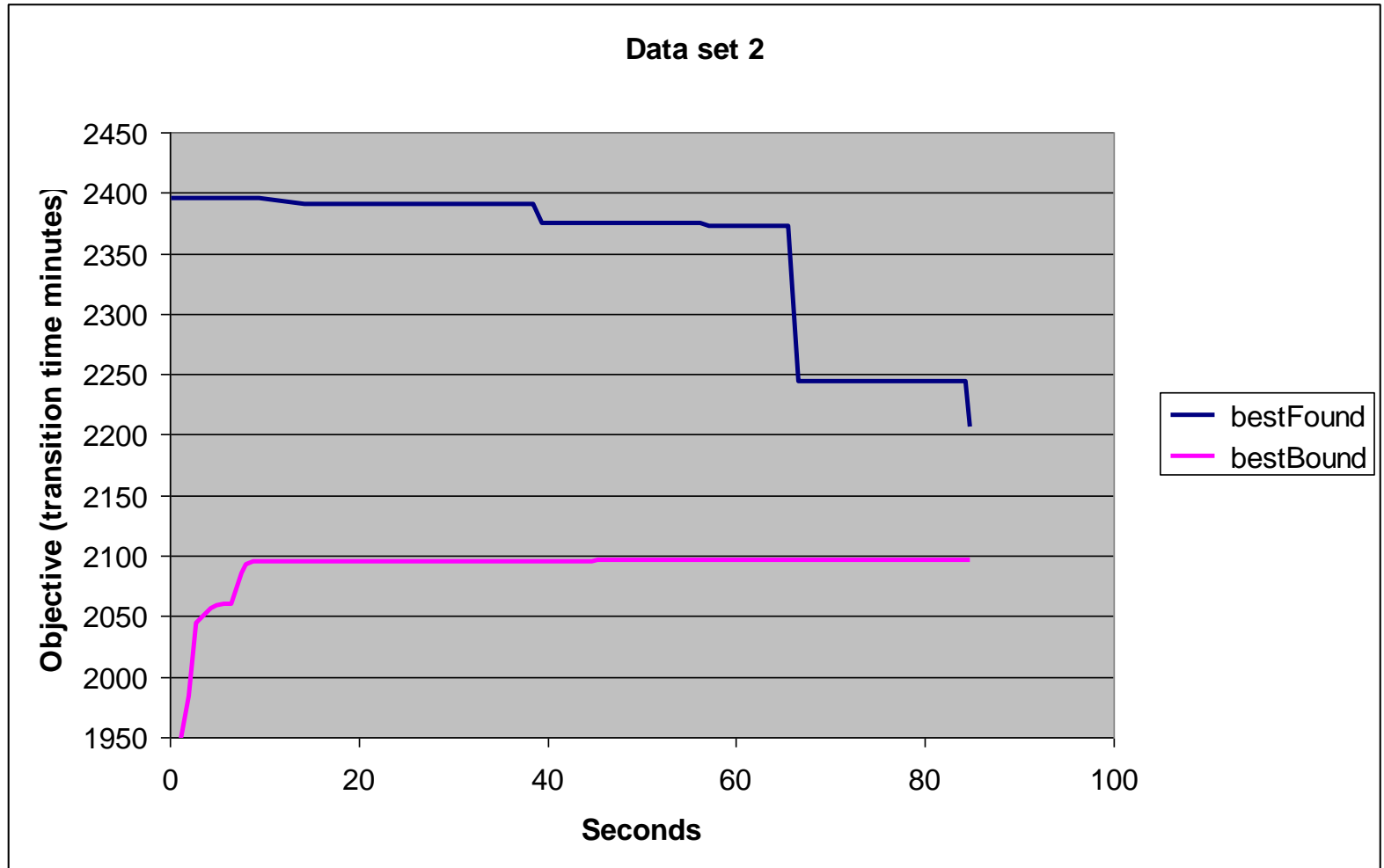
MODEL STATISTICS

BLOCKS OF EQUATIONS	14	SINGLE EQUATIONS	16,956
BLOCKS OF VARIABLES	10	SINGLE VARIABLES	11,448
NON ZERO ELEMENTS	60,102	DISCRETE VARIABLES	4,092

Data set 2, 41 jobs

- Fixed model: obj=2867,time=0.1 sec
- Solve for each line, keep rest fixed (optcr=2%)
 - Obj=2688, time=0.3 sec
 - Obj=2396, time=0.5 sec
 - Obj=2396, time=0.2 sec
 - Obj=2396, time=0.1 sec
 - Obj=2396, time=0.1 sec
- Solve complete model: obj=2207, time=85 sec, gap=5%
- No improvement afterward

Data set 2, Bounds



Data set 2, Size of model

MODEL STATISTICS

BLOCKS OF EQUATIONS	14	SINGLE EQUATIONS	7,178
BLOCKS OF VARIABLES	10	SINGLE VARIABLES	5,660
NON ZERO ELEMENTS	26,790	DISCRETE VARIABLES	1,695

Assumptions

- Proposed schedule is feasible, unscheduled jobs are marked
 - Initial fixed model and line by line optimization will fail otherwise, putting too much burden on integrated model
 - We can make the line by line models smarter to deal with infeasibilities
- Planning horizon starts with completion time of last fixed job
 - No fixed jobs within planning horizon
- No idle time between jobs
 - Exception: we can start with idle time in case first job cannot start immediately because of max earliness
 - Formulation allowing idle time everywhere is developed but solves slower

Production version (discussion)

- Handle problematic inputs
 - Data checks
 - Some of them are already in prototype but production system needs more systematic approach
 - Provide feedback on nature of problem
 - Infeasible schedules (no solution exists)
 - Detect and handle
 - Possible strategies:
 - » Extra dummy machine for overflow
 - » Relax due dates
 - » ...
 - Proposed schedule is not correct (e.g. not feasible)
 - Try to repair in line-by-line optimization
 - Try to repair in integrated model
- Handle failures
 - If sub-model fails we should recover
- Line balancing
 - Try to minimize number of jobs allocated to line k
 - Which line to select to try this (recognize lightly loaded lines)
 - How to evaluate
 - Can be done afterwards:
 - » provide alternative schedules

Allow downtime for repair

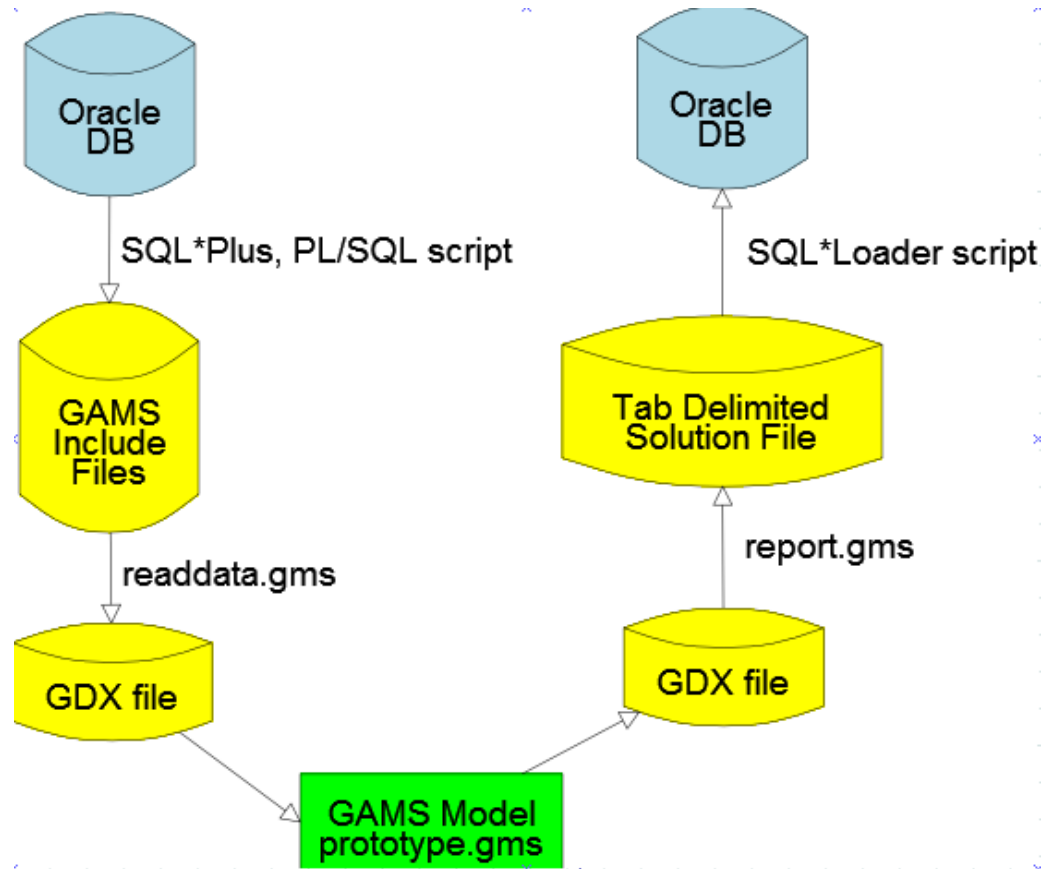
Completion times:

$$S_j = \sum_k \left[(F_k + \Delta_{j,k}^{init} + P_{j,k}) x_{j,k}^{first} + \sum_i (S_i + \Delta_{i,j,k} + P_{i,k}) x_{i,j,k}^{order} \right]$$

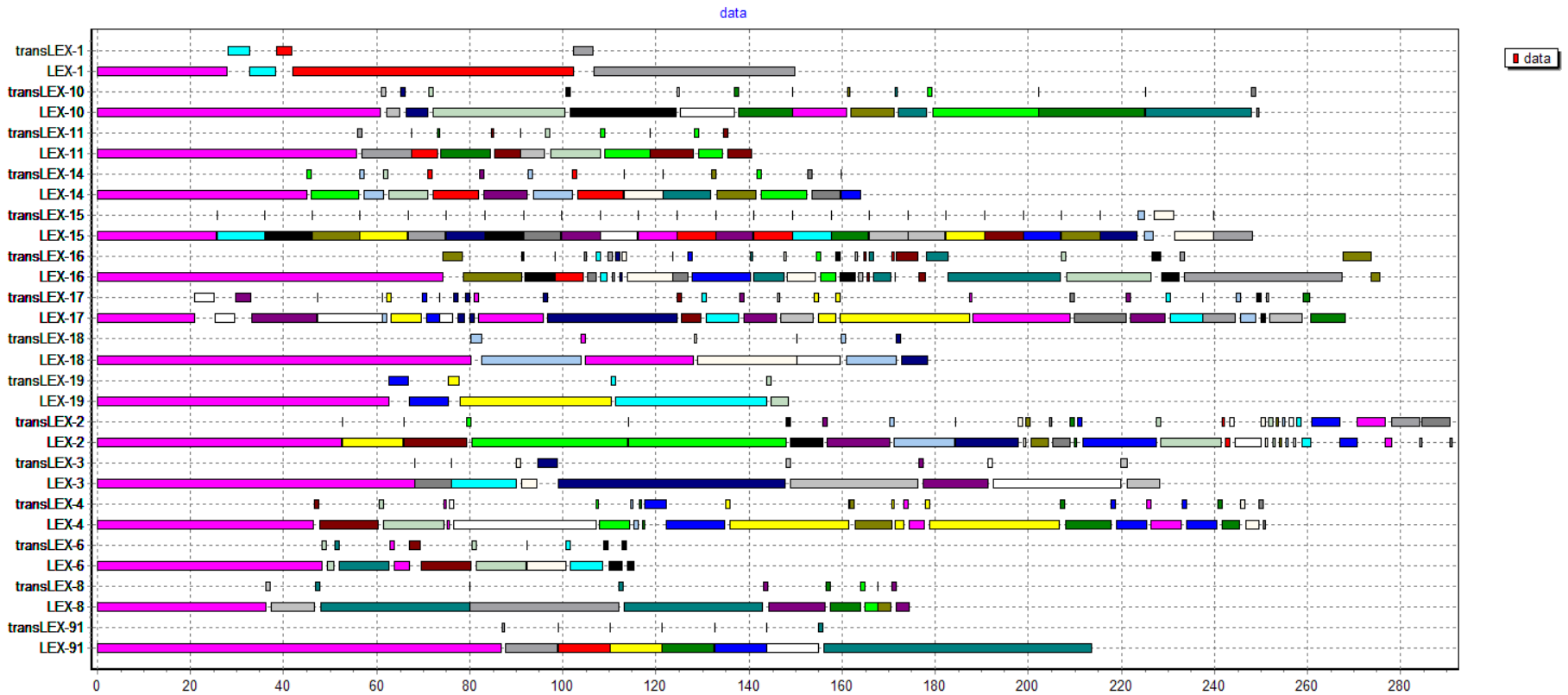
- Scheduled down time is just another job with fixed completion time $S(i)$
- Replace = in above equation to an \geq to allow for idle time on a machine

Unfortunately, this made models much more difficult to solve

Oracle Integration



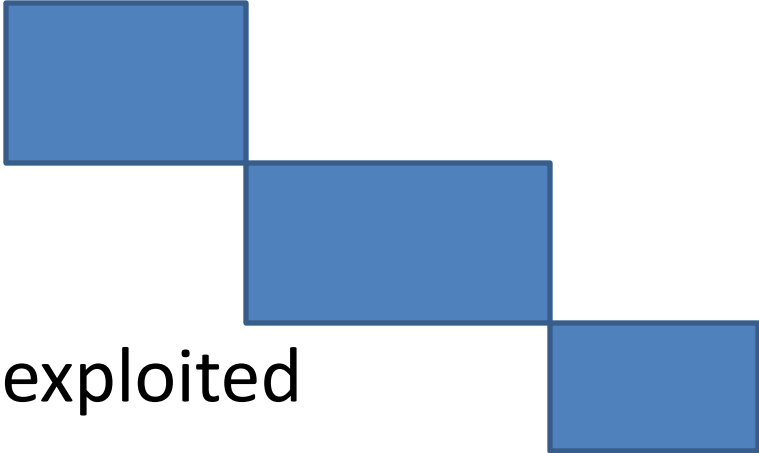
A bigger instance



Problems

- Model with scheduled maintenance (pink orders) was difficult compared to prototype model
- Problems with bug in GAMS/Cplex link
 - Returned sometimes wrong solution
- Much effort to do input checks
 - Input would need more redundancy to give better error messages
- Some additional constraints were only formulated after they saw solutions
 - Eg. Keep natural order if not detrimental to overall switch-over time.
 - All feedback was about single line scheduling
 - schedulers have good intuition on per line schedule, not on overall schedule

Additional Feature

- Some jobs can only be executed on some machines
- Sometimes matrix of allowable jobs-machine assignments has block-diagonal structure
- I.e. after reordering:


The diagram illustrates a block-diagonal matrix structure. It consists of three blue rectangular blocks arranged along the main diagonal of an implied square matrix. The first block is in the top-left position, the second is in the middle-right position, and the third is in the bottom-right position. The blocks are separated by white space, indicating that the matrix is sparse and has a specific structure where only certain jobs are assigned to certain machines.
- We can solve smaller problems if this can be exploited
- Coded this algorithm in GAMS

Of course in practice:

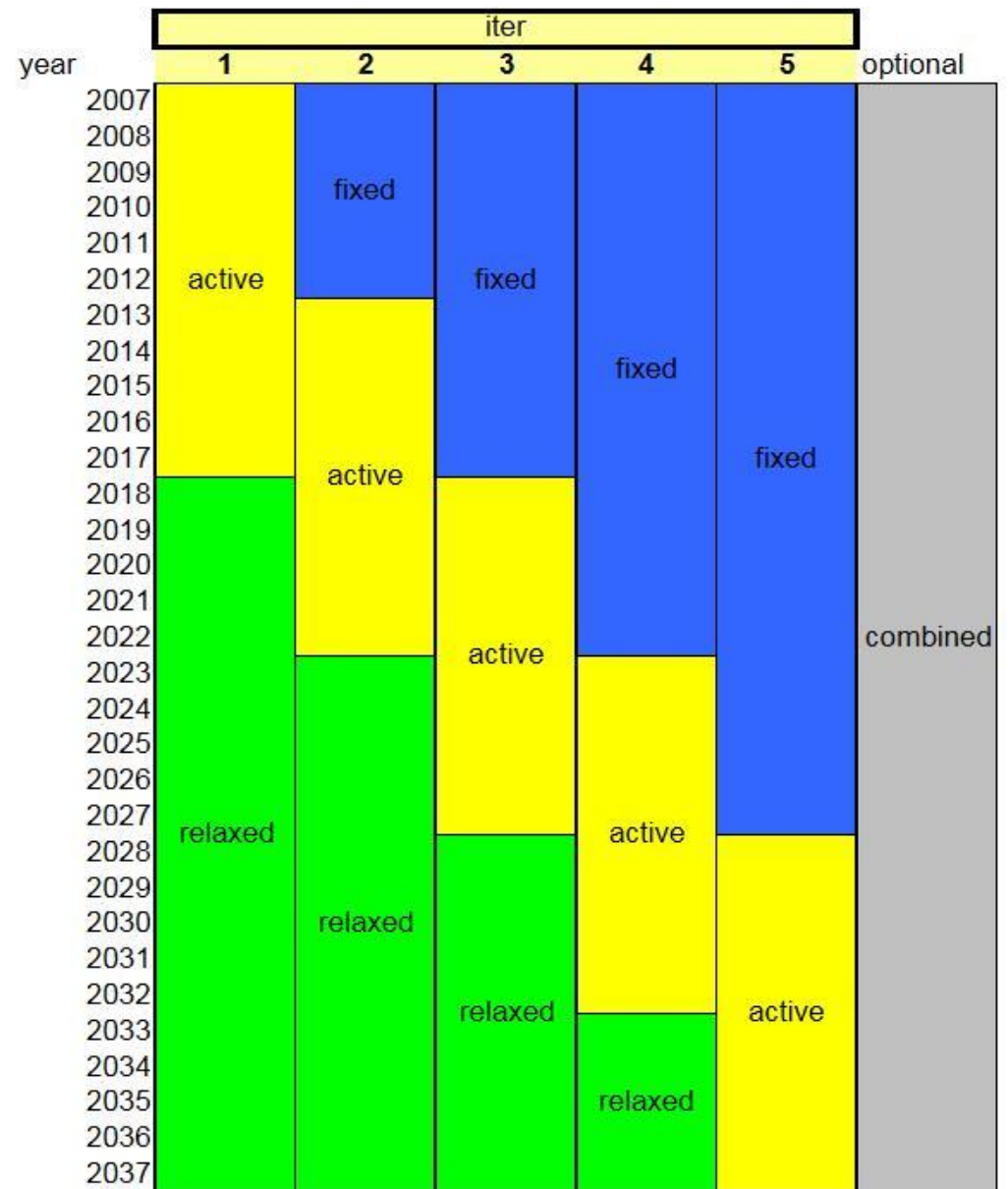


Writing Algorithms

- A system like GAMS will allow you to implement “mini” algorithms quickly. This can increase the range of models that can be solved.
- Sometimes this is very easy in GAMS
- Example: rolling horizon algorithm in power planning model (investment in generators on New Zealand grid)

Rolling Horizon

- Split whole model in pieces wrt integer variables
- But use overlap to mitigate end-of-horizon effects
- Optional: solve big one at end (using MIPSTART)



obj	12202.14	12220.56	12226.11	12239.86	12247.47	12247.47
relgap	0%	0%	0%	0%	0%	0.24%
absgap	0	0	0	0	0	
time	125.859	801.906	293.343	352.734	142.312	3002.078

```
sets
subiter 'rolling horizon iteration'
/iter1*iter5/
relaxed(subiter,yr) /
iter1.(2018*2037)
iter2.(2023*2037)
iter3.(2028*2037)
iter4.(2033*2037)
/
fixed(subiter,yr) /
iter2.(2007*2012)
iter3.(2007*2017)
iter4.(2007*2022)
iter5.(2007*2027)
/
;
```

```
* Solve GEM:
gem.optfile=1;
gem.reslim=1000;
gem.optcr=0;
gem.optca=0;
```

```
loop(subiter,
    GENBLDINT.prior(s,yr) = 1;
    loop(relaxed(subiter,yr),
        GENBLDINT.prior(s,yr) = INF;
    );
    loop(fixed(subiter,yr),
        GENBLDINT.fx(s,yr) = GENBLDINT.L(s,yr);
    );
    SOLVE GEM USING MIP MINIMIZING TOTALCOST ;
    gem.optfile=2;
);

gem.optfile=3;
gem.reslim=10000;
GENBLDINT.prior(s,yr) = 1;
GENBLDINT.lo(s,yr) = 0;
GENBLDINT.up(s,yr) = 1;
SOLVE GEM USING MIP MINIMIZING TOTALCOST ;
```

Also for large difficult NLPs



International Food Policy Research Institute

sustainable solutions for ending hunger and poverty

- Very large convex linearly constrained NLP (spatial land allocation model, part of economic analysis):
 - Rows:28792
 - Cols:546866
 - Nz:2771109
 - Nlnz:543391
- They stopped Conopt with obj= 202.0981 after 9455 seconds
- Mosek found obj= 101.7951 in 582 seconds but “NEAR OPTIMAL” (4 cores)

Mini “SQP”

- Solve QP a few times until convergence
 - $\min g(x) = f(x^0) + \nabla f(x^0) (x - x^0) + 0.5 (x - x^0)^T \nabla^2 f(x^0) (x - x^0)$
s.t. $Ax = b$

```
MODEL mapprox /LANDTOT, SUBCROP, SUMONE, IRRLIMIT, RDEF5/;  
option qcp=mosek;
```

```
set iter /1*100/;
```

```
parameter objApprox(iter);
```

```
scalar done /0/;
```

```
loop(iter$(not done),
```

```
  solve mapprox using qcp minimizing entropy;
```

```
  objApprox(iter) = entropy.l; display objApprox;
```

```
  done$(abs(objApprox(iter)-objApprox(iter-1)) <= 1.0e-5) = 1;
```

```
);
```

Results

model	time	obj
nlp	582.648	101.7951
qp1	73.003	31.3121
qp2	70.156	27.508
qp3	65.358	27.0025
qp4	64.346	26.8975
qp5	67.953	26.8712
qp6	69.314	26.8353
qp7	70.216	26.7588
qp8	72.137	26.6631
qp9	72.846	26.6035
qp10	94.495	26.5873
qp11	98.648	26.5851
qp12	92.613	26.585
qp13	93.819	26.585
conopt	538.999	26.585

Total turnaround of the model went down from 6 hours to 30-40 minutes with much better objective value.

Test with CONOPT is optional (just verifying the Solution)

New Developments

- Some new solvers are entering the market



G u r o b i
Optimization



Microsoft®
Solver Foundation

Example MS Solver Foundation

Model Import Export Binding Check Solve Next Stop Summary

Model Solve Analyze

A1 fx

A B C D E F G H I J K L M N O P Q R

1

2 **Sudoku**

3

4 Solve the 9x9 sudoku problem as a CSP problem.

5 <http://en.wikipedia.org/wiki/Sudoku>

6

7

8

9 **Input**

	c1	c2	c3	c4	c5	c6	c7	c8	c9
r1	5	3			7				
r2	6			1	9	5			
r3		9	8					6	
r4	8				6				3
r5	4			8	3				1
r6					2				6
r7		6					2	8	
r8				4	1	9			5
r9					8			7	9

Specify numbers between 0 and 9
0 means: let solver decide.

Amsterdam Optimization Modeling Group LLC
<http://www.amsterdamoptimizati>

10

11

12

13

14

15

16

17

18

19

20

21 **Output**

	c1	c2	c3	c4	c5	c6	c7	c8	c9
r1	5	3	4	6	7	8	9	1	2
r2	6	7	2	1	9	5	3	4	8
r3	1	9	8	3	4	2	5	6	7
r4	8	5	9	7	6	1	4	2	3
r5	4	2	6	8	5	3	7	9	1
r6	7	1	3	9	2	4	8	5	6
r7	9	6	1	5	3	7	2	8	4

Modeling Pane

Microsoft® Solver Foundation

Data Binding

d<="Main!\$C\$11:\$K\$19"

Model

```

1
2 Model[
3
4 Parameters[Sets,I,J],
5 Parameters[Integers,d[I,J]],
6
7 Decisions[Integers[1,9],x[I,J]],
8
9 Constraints[
10 //FilteredForeach[{i,I},{j,J},d[i,j]>0,x[i,j]==d[i,j]],
11 Foreach[{i,I},{j,J},x[i,j]==d[i,j] | d[i,j]==0],
12 Foreach[{i,I},Unequal[Foreach[{j,J},x[i,j]]]],
13 Foreach[{j,J},Unequal[Foreach[{i,I},x[i,j]]]],
14 Foreach[{ib,3},

```

Model Validation

Model syntax passed

Main Solver Foundation Results

Modeling pane open

90%

Just a bit too large for standard edition

The screenshot displays the Microsoft Excel Solver Foundation interface. The main window shows a grayscale image of the Mona Lisa. The Solver Foundation Modeling Pane is open on the right, displaying the following configuration:

- Data Binding:** $c \leq \text{"Data!\$B\$2:\$I\$361"}$
- Model:**

```
1 Model[
2
3 Parameters[Sets,I,J],
4 Parameters[Integers,c[I,J]],
5
6 Decisions[Reals[0,1],x[I,J]],
7
8 Constraints[
9   Foreach[{i,I}, Sum[{j,J}, x[i,j]] <= 1],
10  Foreach[{j,J}, Sum[{i,I}, x[i,j]] == 1]
11 ],
12
13 Goals[
14   Minimize[Sum[{i,I},{j,J}, c[i,j]*x[i,j]]]
15 ]
16 ]
17 ]
```
- Model Validation:** Error: Model size limit has been exceeded for this version of the product. Please contact Microsoft Corporation for licensing options.

The status bar at the bottom indicates a "Solve Error" and shows the Solver Foundation Results tab is active.