Integer programming for the MAP problem in Markov random fields

James Cussens, University of York

HIIT, 2015-04-17

A Markov random field (MRF)

A	B	
-	-	—
0	0	0.10
0	1	0.20
1	0	0.30
1	1	0.20

$\times$

A	D	
-	-	—
0	0	0.90
0	1	0.20
1	0	0.70
1	1	0.10

$\times$

B	C	
-	-	—
0	0	0.40
0	1	0.70
1	0	0.30
1	1	0.10

$\times$

C	D	
-	-	—
0	0	0.50
0	1	0.20
1	0	0.40
1	1	0.10

A Markov random field (MRF)

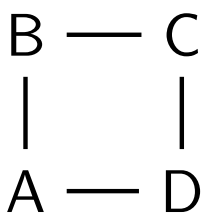
A	B			A	D			B	C			C	D	
-	-	—		-	-	—		-	-	—		-	-	—
0	0	0.10		0	0	0.90		0	0	0.40		0	0	0.50
0	1	0.20	×	0	1	0.20	×	0	1	0.70	×	0	1	0.20
1	0	0.30		1	0	0.70		1	0	0.30		1	0	0.40
1	1	0.20		1	1	0.10		1	1	0.10		1	1	0.10

- An MRF defines a joint probability distribution over its variables:
- $P(A = 0, B = 1, C = 1, D = 0) \propto 0.2 \times 0.9 \times 0.1 \times 0.4$

A Markov random field (MRF)

A	B			A	D			B	C			C	D	
-	-	—		-	-	—		-	-	—		-	-	—
0	0	0.10		0	0	0.90		0	0	0.40		0	0	0.50
0	1	0.20	×	0	1	0.20	×	0	1	0.70	×	0	1	0.20
1	0	0.30		1	0	0.70		1	0	0.30		1	0	0.40
1	1	0.20		1	1	0.10		1	1	0.10		1	1	0.10

- Its associated graph can be used to 'read off' conditional independence relations:



MAP for MRFs

- The MAP problem for MRF is to find an instantiation of the variables with maximal probability.
- This is an NP-hard problem.
- $A = 1, B = 0, C = 1, D = 0$ is a MAP solution for our example MRF:

A	B			A	D			B	C			C	D	
-	-	—		-	-	—		-	-	—		-	-	—
0	0	0.10		0	0	0.90		0	0	0.40		0	0	0.50
0	1	0.20	×	0	1	0.20	×	0	1	0.70	×	0	1	0.20
1	0	0.30		1	0	0.70		1	0	0.30		1	0	0.40
1	1	0.20		1	1	0.10		1	1	0.10		1	1	0.10

MIP for MRF MAP

- It is easy to encode an MRF MAP problem as an integer program with only binary variables.
- This is a special case of a *mixed integer program (MIP)* where there are no real-valued variables, so not properly 'mixed'.
- But we'll call these MIPs nonetheless due to the nice alliteration.
- Cue demo . . .

Solving without thinking

- 458 MRF MAP instances from the UAI 2011 PIC challenge were given to CPLEX 12.6 using the MIP encoding I have just presented.
- Machine: 4-core 3.2GHz with 7.8Gb RAM
- CPLEX solved 313 (to certified optimality) in under 2000 seconds.
- Half of all instances were solved in under 6 seconds.
- The `fileforGal_350markers` instance had had 7,871,997 MIP variables (after presolving)and 312,420 linear constraints. It was solved in 870.97 seconds. Solving its initial linear relaxation took 458.09 seconds.

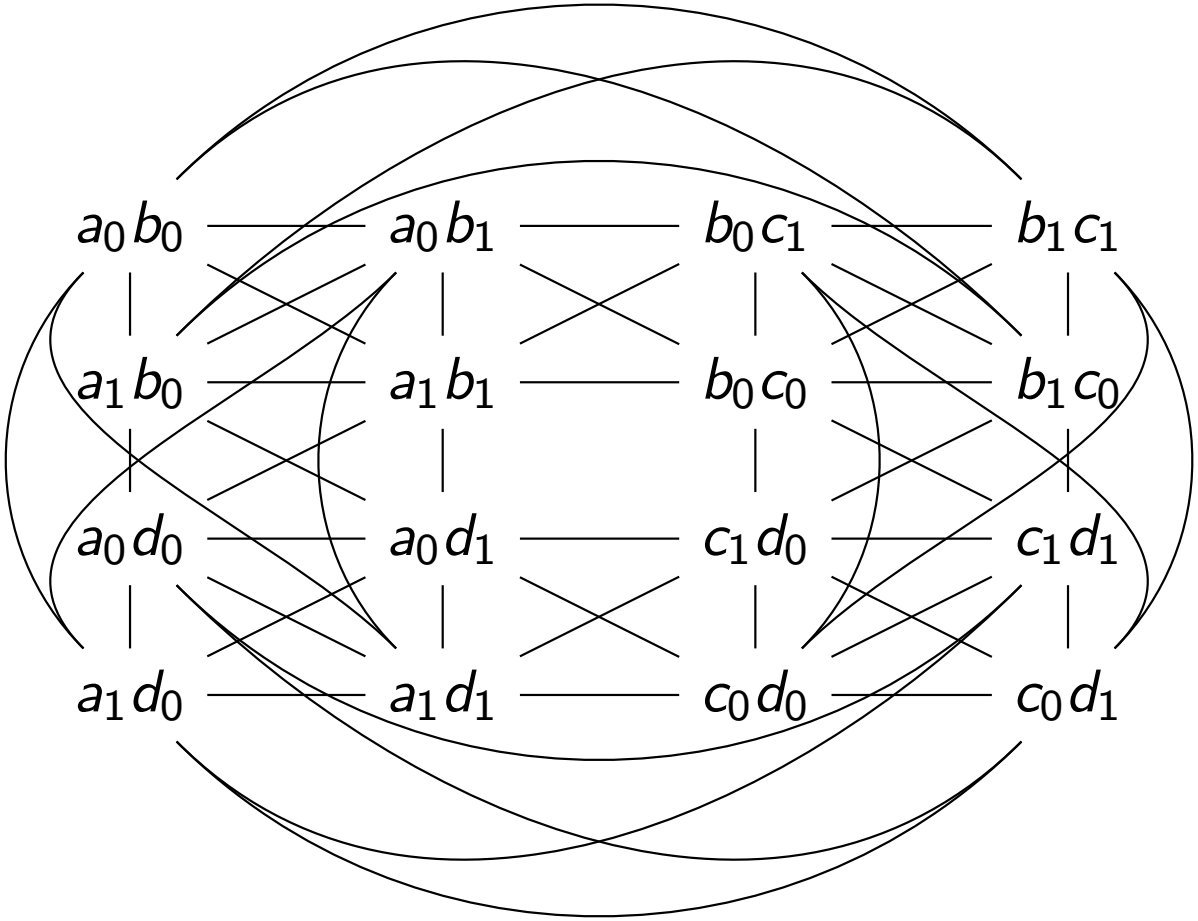
Solving with added thinking

- A *set partitioning constraint* states that exactly one of a given set of binary variables has value 1.
- For example, $a_0 b_0 + a_0 b_1 + a_1 b_0 + a_1 b_1 = 1$.
- A *set partitioning problem (SPP)* is a MIP with only binary variables where all constraints are set partitioning constraints.
- It is easy to see that any MRF MAP instance can presented as an SPP instance.
- Cue demo . . .

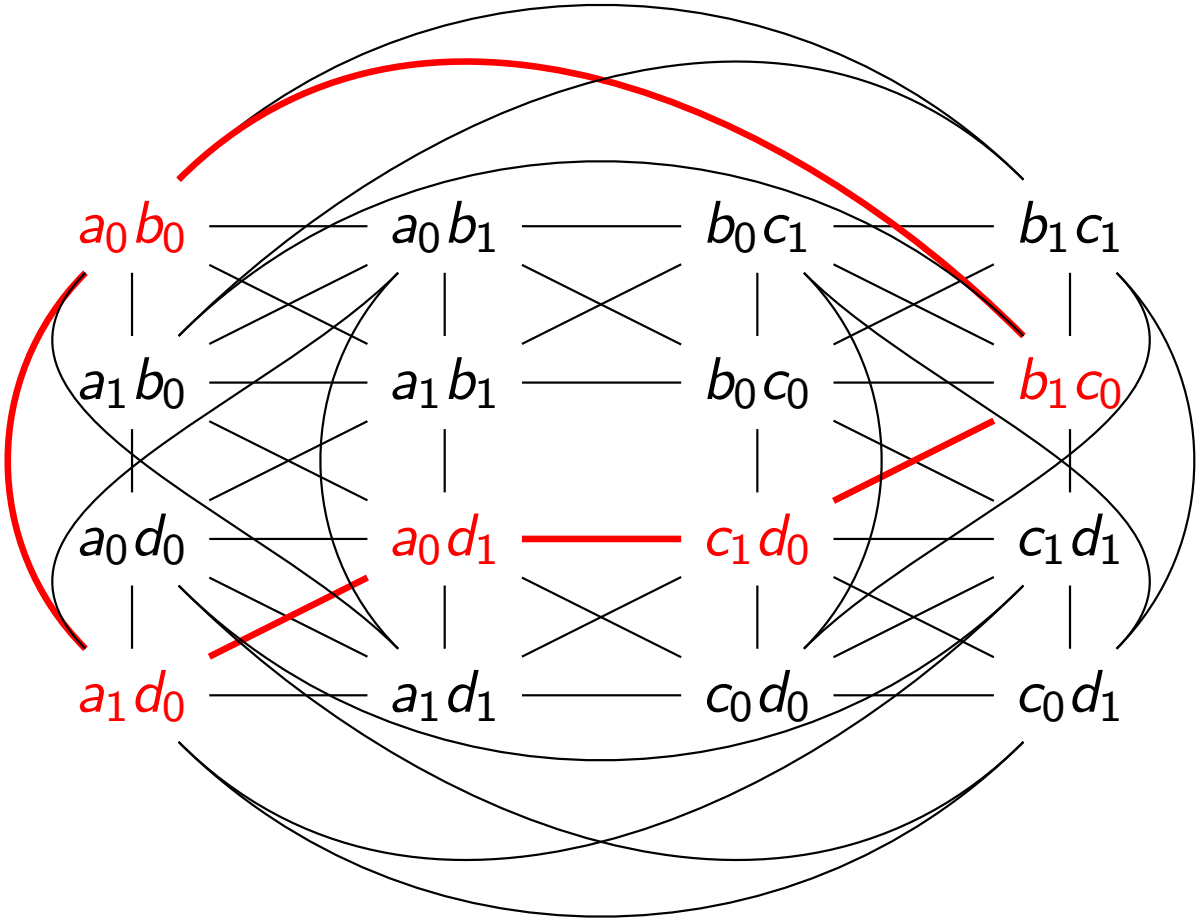
Set partitioning: why do we care?

- Set partitioning (SPP), and the closely related problems of *set packing (SP)* (replace $=$ with $\leq$) and *set covering (SC)* (replace $=$ with $\geq$), have been extensively studied in the mathematical programming community for decades [BP76].
- For example, we can convert any SPP into either an SP or an SC instance.
- We can convert any SP instance into a *node packing* (aka *vertex covering, maximum independent set, etc*) instance, although this is typically not a good idea, since the linear relaxation is weaker.
- There are also many results on how good various approximate algorithms are [Pas97].

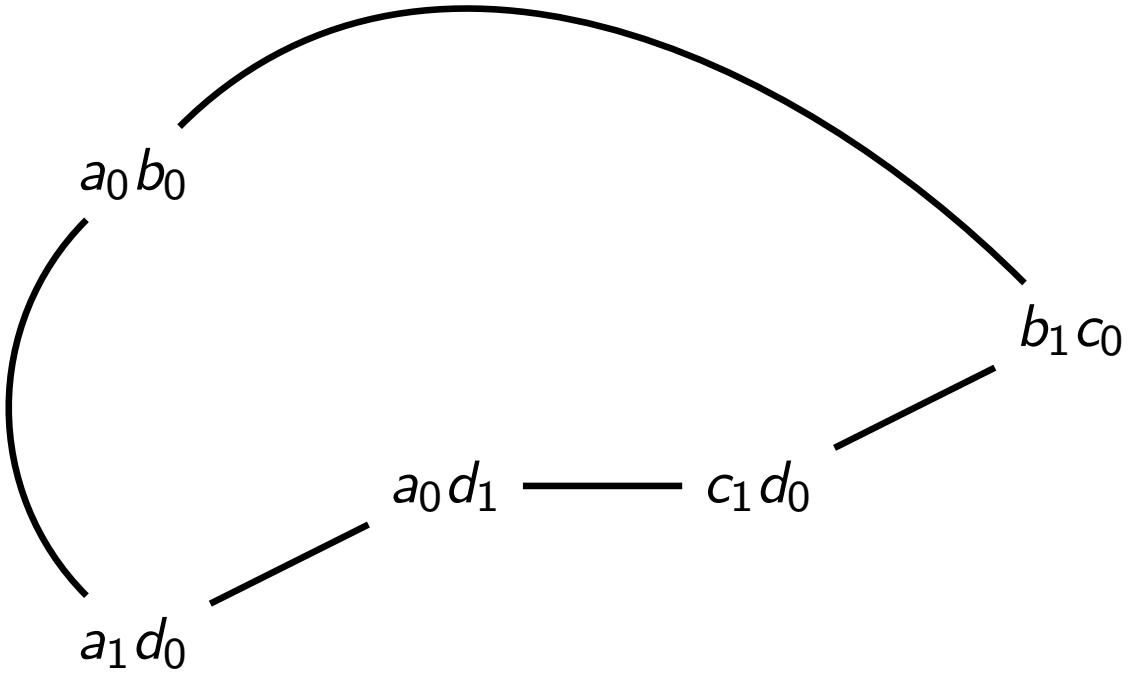
Intersection graph



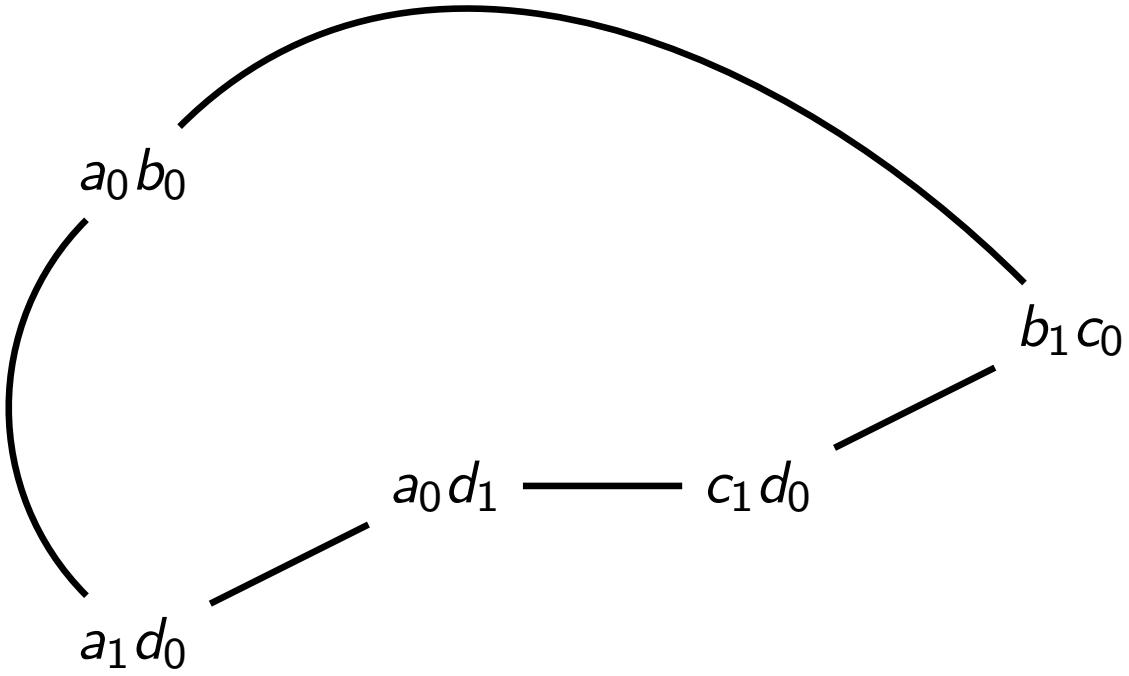
An 'odd hole' in the intersection graph



An 'odd hole' in the intersection graph

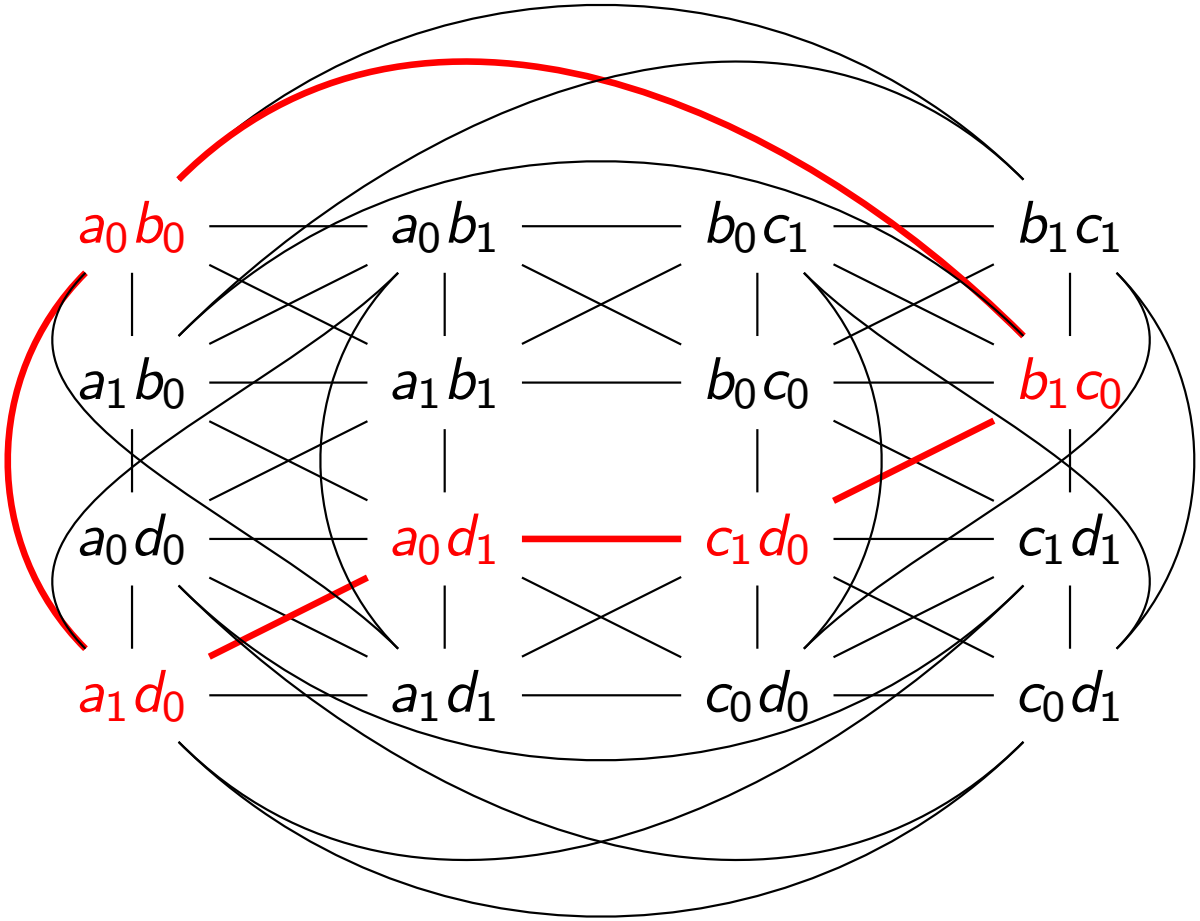


An 'odd hole' in the intersection graph

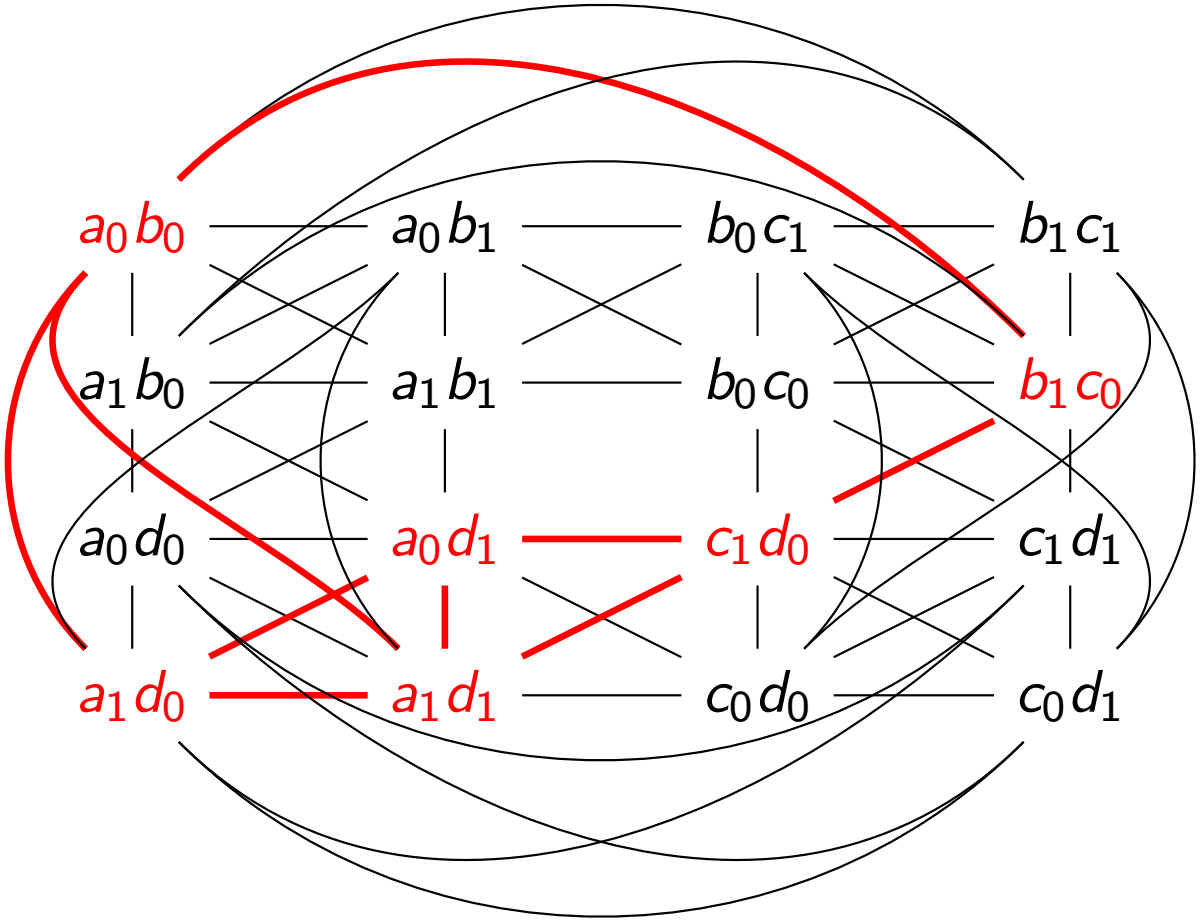


$$a_0\,b_0 + b_1\,c_0 + c_1\,d_0 + a_0\,d_1 + a_1\,d_0 \leq 2$$

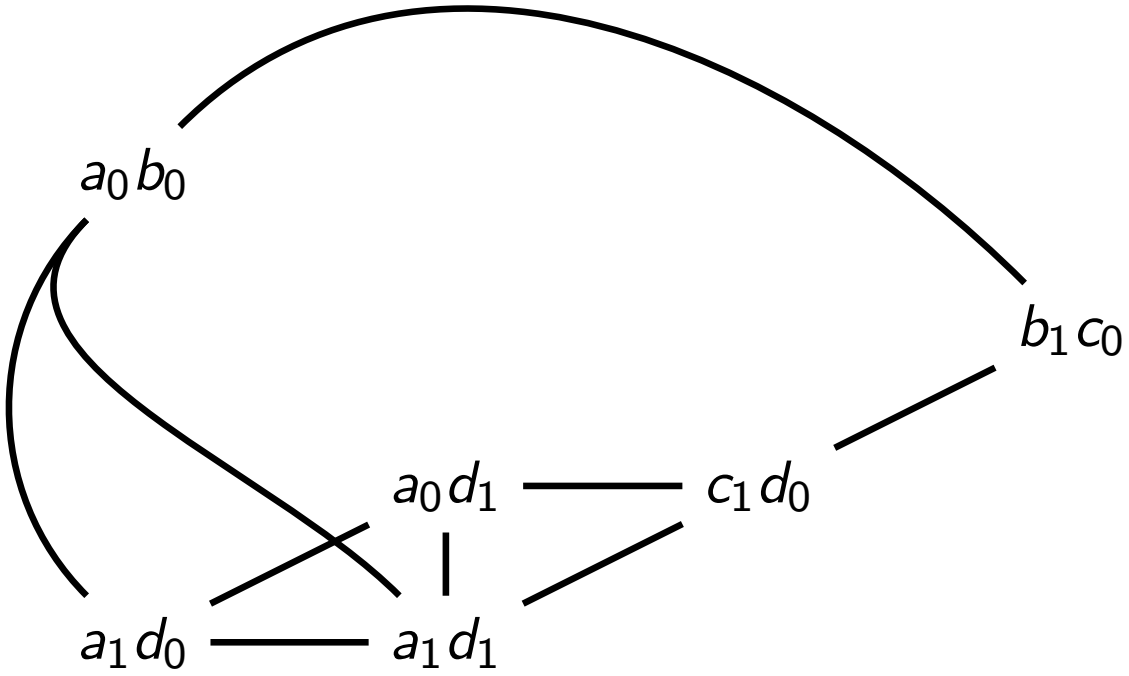
Lifting odd holes



Lifting odd holes



Lifting odd holes



$$a_0\,b_0 + b_1\,c_0 + c_1\,d_0 + a_0\,d_1 + a_1\,d_0 + a_1\,d_1 \leq 2$$

Odd cycle cuts

- So how do we (quickly) find these cuts?
- We could use the MIP solver SCIP and change this default setting:
  - separating/oddcycle/freq = -1
  - separating/oddcycle/liftoddcycles = FALSE

  to, say, this:
  - separating/oddcycle/freq = 1
  - separating/oddcycle/liftoddcycles = TRUE
- Some (not very thorough) testing indicates that this is helpful.
- But we can do better.

Zero-half cuts

Take 5 simple 'edge' inequalities:

$$a_0 b_0 + b_1 c_0 \leq 1$$
$$b_1 c_0 + c_1 d_0 \leq 1$$
$$a_0 d_1 + c_1 d_0 \leq 1$$
$$a_0 d_1 + a_1 d_0 \leq 1$$
$$a_1 d_0 + a_0 b_0 \leq 1$$

multiply each by half and add to derive:

$$a_0 b_0 + b_1 c_0 + c_1 d_0 + a_0 d_1 + a_1 d_0 \leq 5/2$$

Since the LHS in integer we can tighten to get:

$$a_0 b_0 + b_1 c_0 + c_1 d_0 + a_0 d_1 + a_1 d_0 \leq 2$$

the odd hole cut shown earlier.

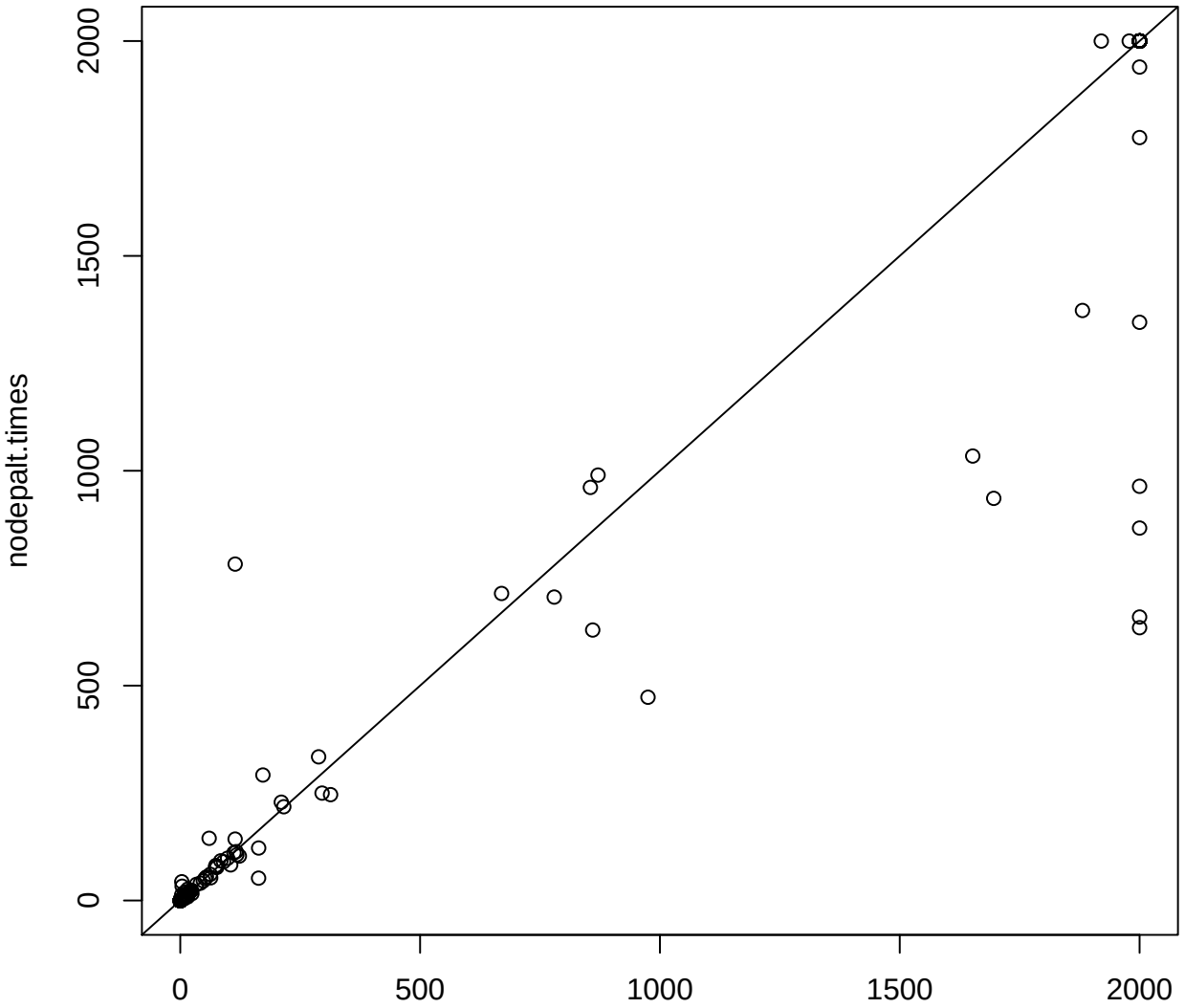
Zero-half cuts in CPLEX

- A cut generated by (i) multiplying some existing inequalities by 1/2, (ii) adding and (iii) rounding is a *zero-half cut*.
- Any (lifted) odd hole cut can be generated as a zero-half cut.
- (As can *odd anti-hole cuts* for which we have no time.)

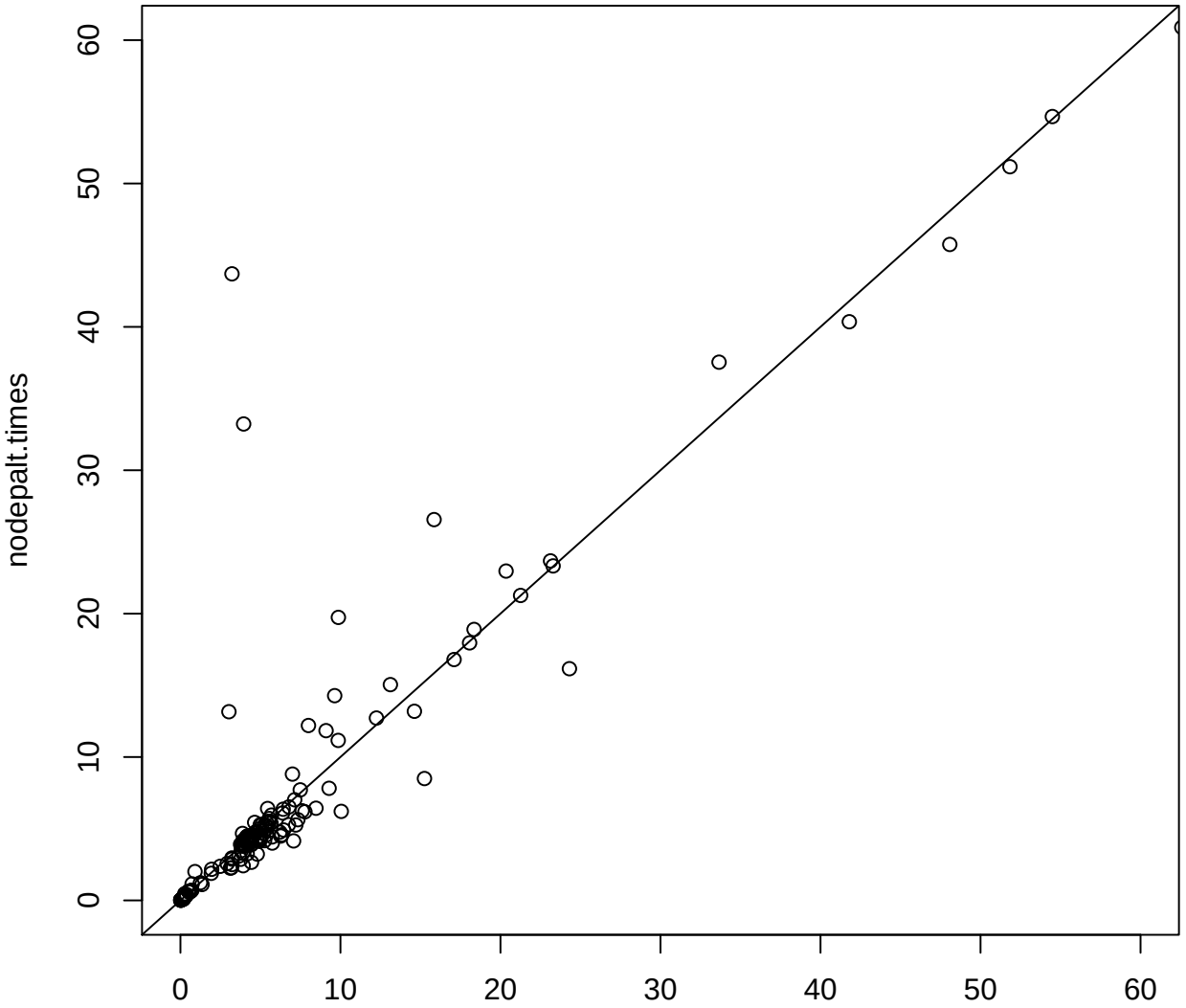
Zero-half cuts in CPLEX

- A cut generated by (i) multiplying some existing inequalities by 1/2, (ii) adding and (iii) rounding is a *zero-half cut*.
- Any (lifted) odd hole cut can be generated as a zero-half cut.
- (As can *odd anti-hole cuts* for which we have no time.)
- So put CPLEX's zero-half cut generator into overdrive!
- mip.cuts.zerohalfcut.set(2)
- mip.limits.cutpasses.set(9999999)
- mip.limits.cutsfactor.set(90)

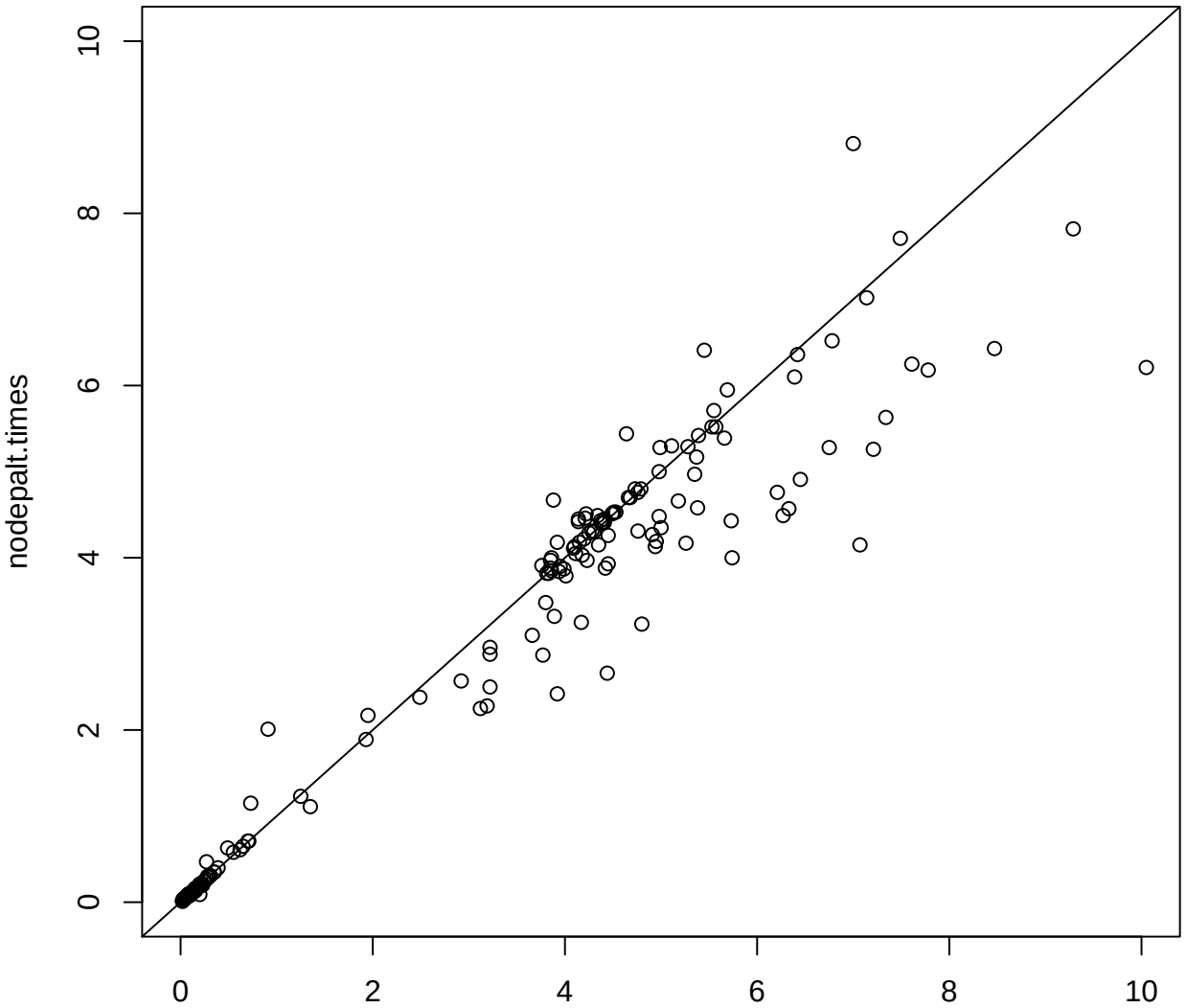
Comparing the times: all problems (5 more solved!)



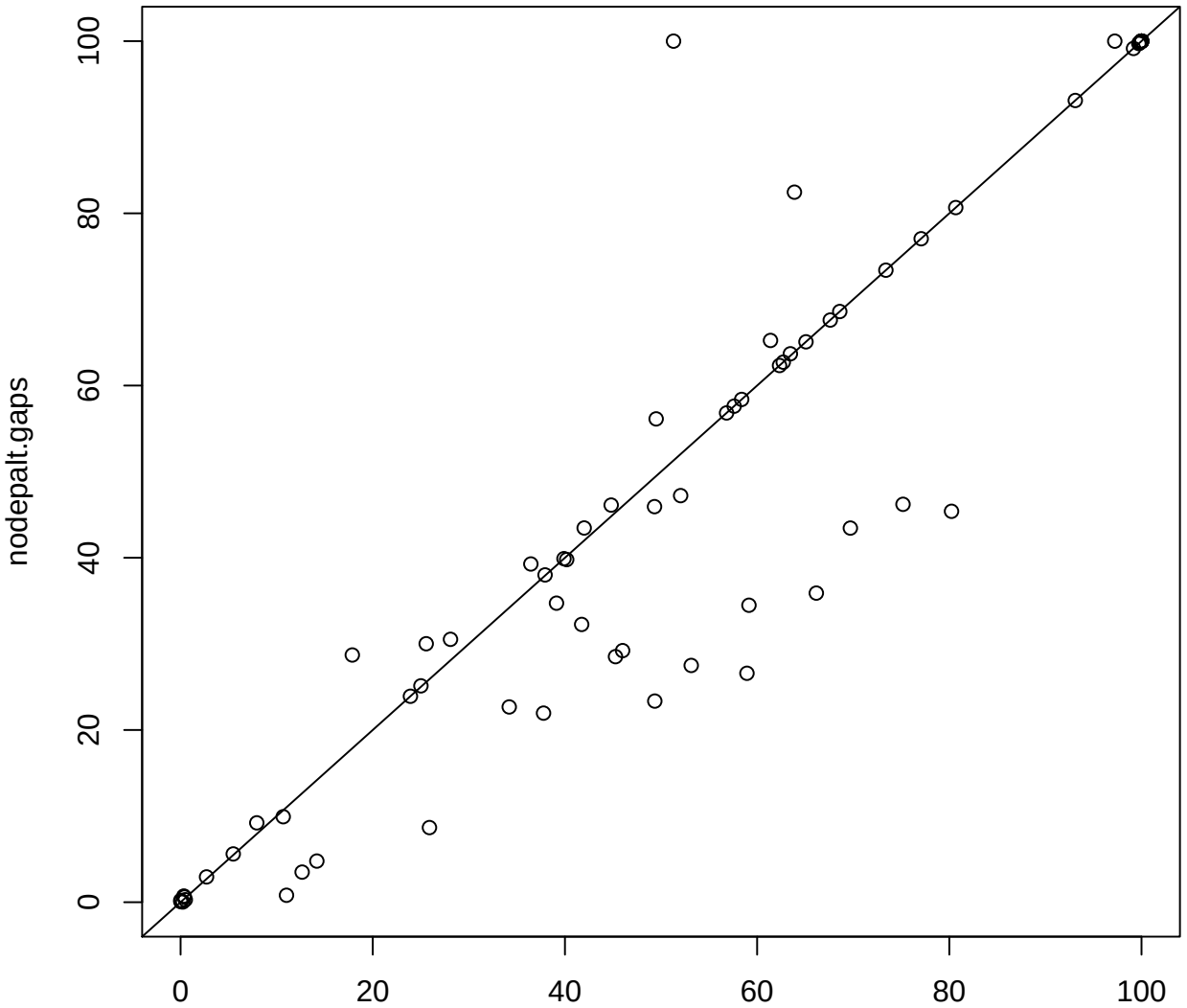
Comparing the times: easy problems



Comparing the times: very easy problems



Comparing gaps: unsolved problems



Markov blanket constraints

A	B			A	D			B	C			C	D	
-	-	—		-	-	—		-	-	—		-	-	—
0	0	0.10		0	0	0.90		0	0	0.40		0	0	0.50
0	1	0.20	×	0	1	0.20	×	0	1	0.70	×	0	1	0.20
1	0	0.30		1	0	0.70		1	0	0.30		1	0	0.40
1	1	0.20		1	1	0.10		1	1	0.10		1	1	0.10

- The *Markov blanket* for D is $\{A, C\}$,
- so for any given joint instantiation of $\{A, C\}$, we can compute the optimal value of D.
- For $A = 0, C = 0$ the value for $D = 0$ is 0.9×0.5 and for $D = 1$ is 0.2×0.2,
- so we have the constraint $a_0 \wedge c_0 \rightarrow d_0$
- we add this as the linear constraint $(1 - a_0) + (1 - c_0) + d_0 \geq 1$

Markov blanket constraints (ctd)

In this example, it so happens that we have:

$$a_0 \wedge c_0 \rightarrow d_0$$
$$a_0 \wedge \neg c_0 \rightarrow d_0$$
$$\neg a_0 \wedge c_0 \rightarrow d_0$$
$$\neg a_0 \wedge \neg c_0 \rightarrow d_0$$

so we can fix d_0 to 1 immediately.

Markov blanket problems

- In the 16 'deer' instances each of the 60 variables has all the other 59 variables in its Markov blanket (every pair of variables is a hyperedge).
- Naively constructing Markov blanket constraints for such examples had predictably dire consequences!

Markov blanket problems

- In the 16 'deer' instances each of the 60 variables has all the other 59 variables in its Markov blanket (every pair of variables is a hyperedge).
- Naively constructing Markov blanket constraints for such examples had predictably dire consequences!
- In other cases, adding Markov blanket constraints makes solving a bit (technical term!) slower.

Markov blanket problems

- In the 16 'deer' instances each of the 60 variables has all the other 59 variables in its Markov blanket (every pair of variables is a hyperedge).
- Naively constructing Markov blanket constraints for such examples had predictably dire consequences!
- In other cases, adding Markov blanket constraints makes solving a bit (technical term!) slower.
- Current implementation is, however, very poor.
- We're mostly cutting not propagating

Markov blanket problems

- In the 16 'deer' instances each of the 60 variables has all the other 59 variables in its Markov blanket (every pair of variables is a hyperedge).
- Naively constructing Markov blanket constraints for such examples had predictably dire consequences!
- In other cases, adding Markov blanket constraints makes solving a bit (technical term!) slower.
- Current implementation is, however, very poor.
- We're mostly cutting not propagating
- Further research needed ...

Markov blanket problems

- In the 16 'deer' instances each of the 60 variables has all the other 59 variables in its Markov blanket (every pair of variables is a hyperedge).
- Naively constructing Markov blanket constraints for such examples had predictably dire consequences!
- In other cases, adding Markov blanket constraints makes solving a bit (technical term!) slower.
- Current implementation is, however, very poor.
- We're mostly cutting not propagating
- Further research needed . . . as always . . .

Acknowledgement: Thanks to COIN for supporting this research visit.


Egon Balas and Manfred W. Padberg.
Set partitioning: A survey.
SIAM Review, 18(4):710–760, October 1976.


Vangelis Th. Paschos.
A survey of approximately optimal solutions to some covering and packing problems.
ACM Comput. Surv., 29(2):171–209, 1997.