Explaining Propagators for Edge-valued Decision Diagrams

Graeme Gange[1] Peter J. Stuckey[1,2] Pascal Van Hentenryck[1,2]

National ICT Australia, Victoria Laboratory

Department of Computer Science and Software Engineering
The University of Melbourne, Vic. 3010, Australia
ggange@csse.unimelb.edu.au
{peter.stuckey,pvh}@nicta.com.au

September 25, 2013

Outline

Finite-domain constraint satisfaction

Given a set of variables $\mathcal{V}$ with domains $\mathcal{D}$, and a set of constraints C over those variables, find an assignment to all variables in $\mathcal{V}$ such that all constraints in C are satisfied.

CP solvers interleave propagation (removing inconsistent values from domains) with backtracking search.

Lazy clause generation

A finite-domain CP engine with SAT-style conflict-directed learning.

- Lazily constructs a partial SAT model of the problem.

Complication:

- Propagators must be able to *explain* any inferences they make.

Motivation: Scheduling

We want to schedule n people among k tasks over m shifts.

- Each task requires a minimum coverage for each shift.
- There is a constraint on the set of legal shifts for each person.
- Each task has an associated cost.

The set of legal schedules can often be encoded as some form of automaton.

Scheduling constraints

$x_{ij} = d$ means person i does task d in shift j

$$
\begin{array}{lll}
\textbf{min} & cost & \\
\text{s.t.} & & \\
 & cost = \sum_{i,j} \text{taskcost}[x_{ij}] & \\
 & \texttt{regular}([x_{i1}, \ldots, x_{im}]) & \forall\ i \\
 & \sum_{i} (x_{ij} = d) \geq \text{cover}_{jd} & \forall\ j,\ d \\
 & x_{ij} \in \{1, \ldots, k\} & \forall\ i,\ j
\end{array}
$$

Unfortunately, this has extremely weak propagation between the schedules and the corresponding costs.

Basic model

$$
\begin{array}{ll}
\textbf{min} & cost \\
\text{s.t.} & \\
 & cost = \sum_{i,j} \text{taskcost}[x_{ij}] \\
 & \texttt{regular}([x_{i1}, \ldots, x_{im}]) \quad \forall\, i \\
 & \sum_{i} (x_{ij} = d) \geq \text{cover}_{jd} \quad \forall\, j,\ d \\
 & x_{ij} \in \{1, \ldots, k\} \quad \forall\, i,\ j
\end{array}
$$

Unfortunately, this has extremely weak propagation between the schedules and the corresponding costs.

Basic model

$$
\begin{array}{ll}
\textbf{min} & cost \\
\text{s.t.} & \\
& cost = \sum_{i,j} \text{taskcost}[x_{ij}] \\
& \texttt{regular}([x_{i1}, \ldots, x_{im}]) \quad \forall\, i \\
& \sum_{i} (x_{ij} = d) \geq \text{cover}_{jd} \quad \forall\, j,\ d \\
& x_{ij} \in \{1, \ldots, k\} \quad \forall\, i,\ j
\end{array}
$$

Unfortunately, this has extremely weak propagation between the schedules and the corresponding costs.

Instead, we construct a combined scheduling-and-cost constraint.

Basic model

$$
\begin{array}{ll}
\textbf{min} & \mathit{cost} \\
\text{s.t.} & \\
& \mathit{cost} = \sum_i c_i \\
& \texttt{cost\_regular}([x_{i1}, \ldots, x_{im}], c_i) \quad \forall\, i \\
& \sum_i (x_{ij} = d) \geq \text{cover}_{jd} \quad \forall\, j,\ d \\
& x_{ij} \in \{1, \ldots, k\} \quad \forall\, i,\ j
\end{array}
$$

Unfortunately, this has extremely weak propagation between the schedules and the corresponding costs.

Instead, we construct a combined scheduling-and-cost constraint.

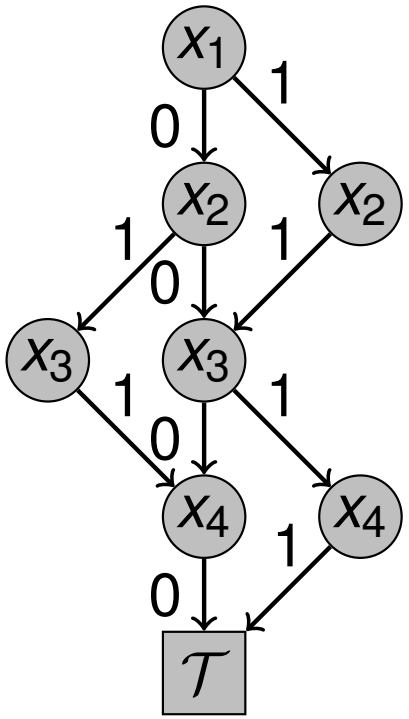
Multi-valued Decision Diagrams

- For a node $n = \langle x, [(v_1, n_1), \ldots, (v_k, n_k)] \rangle$:

  $\langle\langle n \rangle\rangle = (x = v_1 \wedge \langle\langle n_1 \rangle\rangle) \vee \ldots \vee (x = v_k \wedge \langle\langle n_k \rangle\rangle)$

- A path from the root to $\mathcal{T}$ represents a satisfying assignment.

How do we incorporate costs?

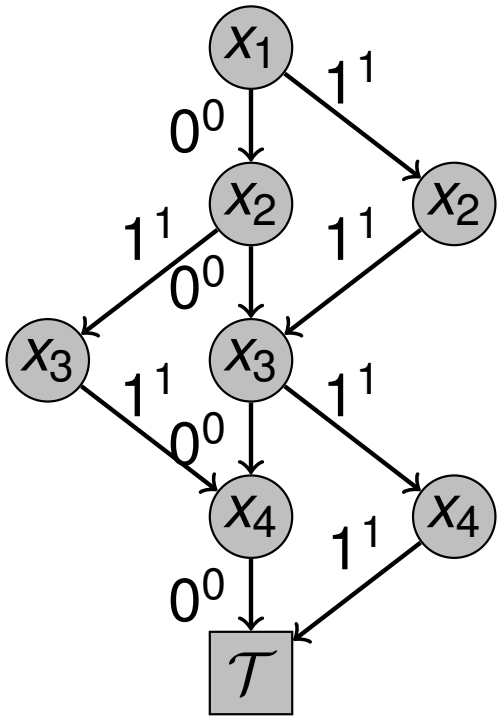


Edge-valued Decision Diagrams

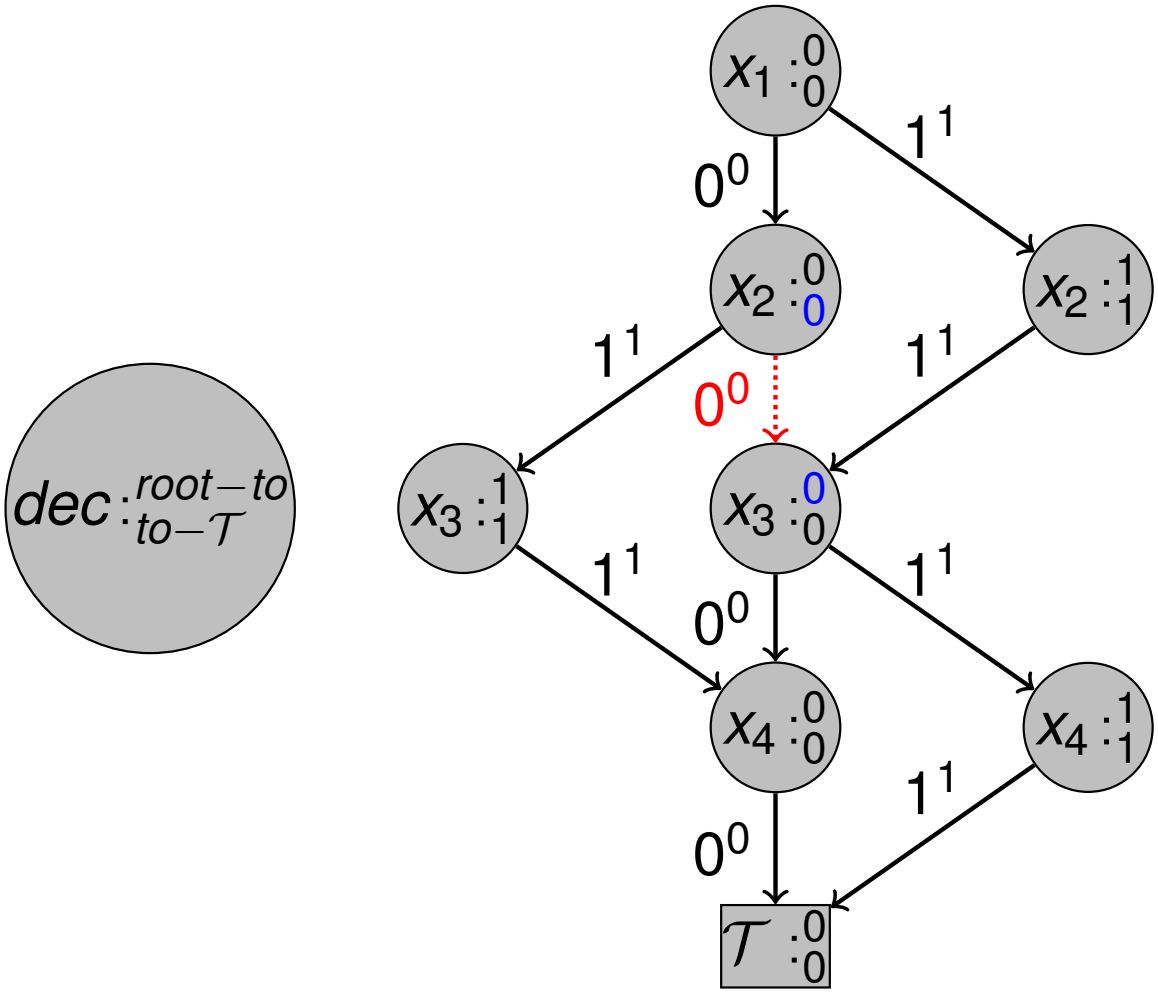
- Just like an MDD, but each edge has a corresponding cost w.
- For a node $n = \langle x, [(v_1, w_1, n_1), \ldots, (v_k, w_k, n_k)] \rangle$:

$$\langle\langle n \rangle\rangle = \min\{w_j + \langle\langle n_j \rangle\rangle \mid j \in 1..k, x_i = v_j\}$$

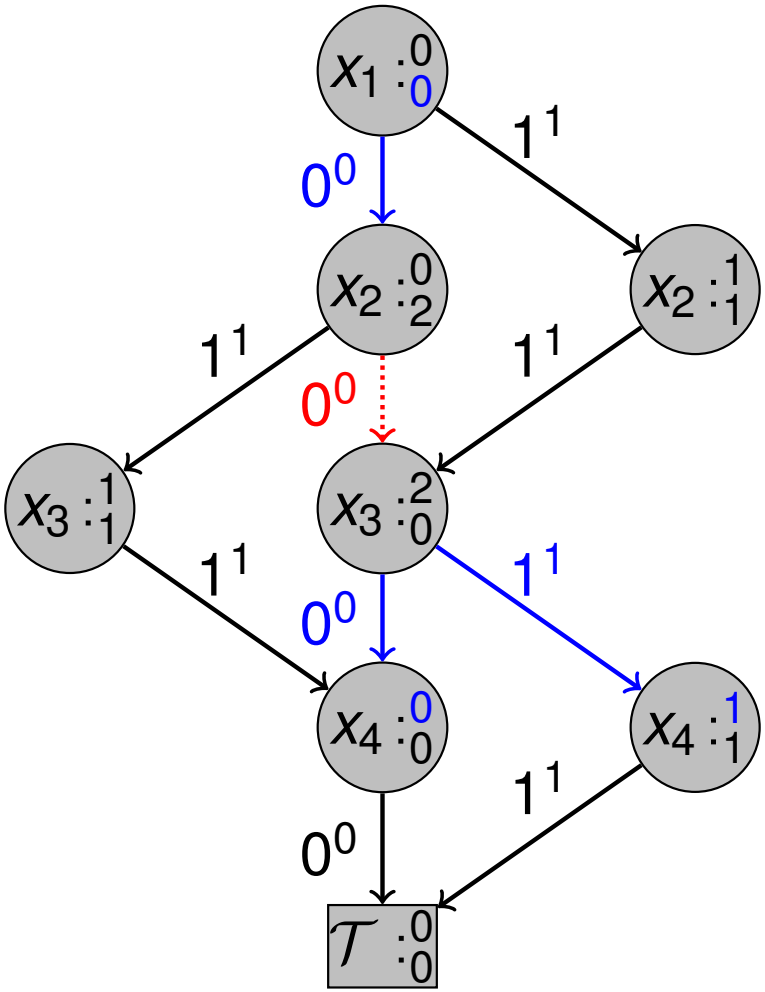
- Propagation is (conceptually) simple:
  - Maintain shortest path to/from each node.
  - When a value is removed from the domain, update the corresponding costs.
  - Eliminate any edges that only occur on infeasible paths.



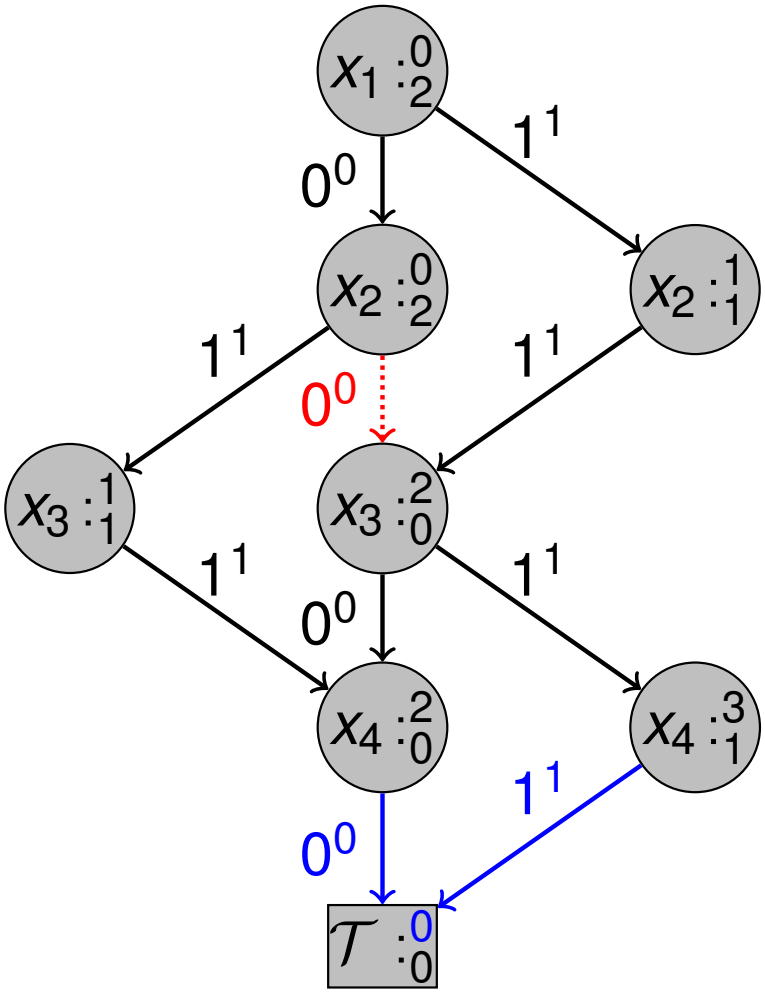
Example: Propagating $[\![x_2 \neq 0]\!]$



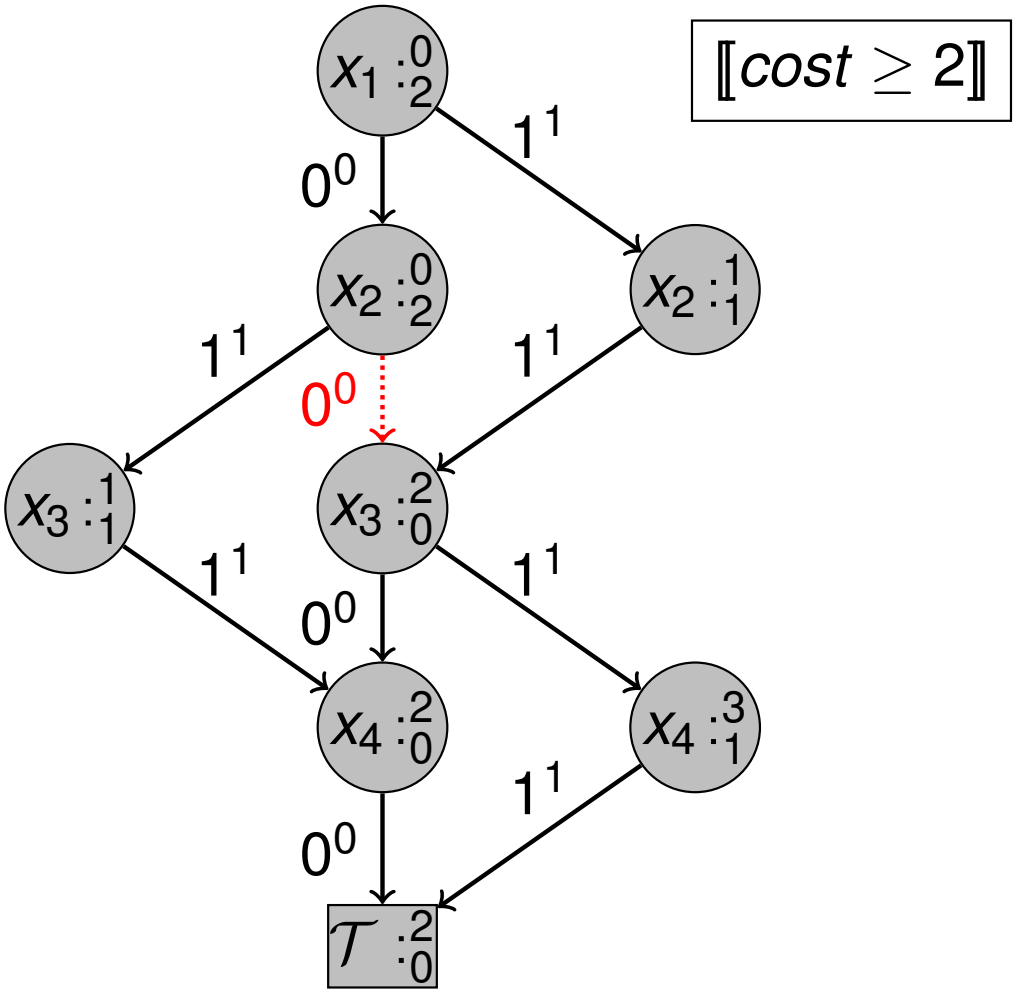
Example: Propagating $[\![x_2 \neq 0]\!]$



Example: Propagating $[\![x_2 \neq 0]\!]$



Example: Propagating $[\![x_2 \neq 0]\!]$



Propagating from ub(*cost*)

When ub(*cost*) changes, edges may become dead even though path lengths don't change.

- We don't want to scan the whole graph every time ub(*cost*) changes.
- Instead, keep a list of possibly-live edges for each value.
  - When ub(*cost*) changes, scan the list until we find a feasible edge.
  - Any examined infeasible edges are removed.
  - If no feasible edges are found, remove the value from the domain.

Explaining Inferences

Three kinds of inferences:

1. Infeasibility:

$$[\![cost \leq k]\!] \wedge [\![x_1 \neq v_1]\!] \wedge \ldots \wedge [\![x_m \neq v_m]\!] \models \bot$$

2. Variable domain:

$$[\![cost \leq k]\!] \wedge [\![x_1 \neq v_1]\!] \wedge \ldots \wedge [\![x_m \neq v_m]\!] \models [\![x \neq v]\!]$$

3. lb($cost$):

$$[\![x_1 \neq v_1]\!] \wedge \ldots \wedge [\![x_m \neq v_m]\!] \models [\![cost \geq k]\!]$$

Explaining Inferences

We can restate cases (2) and (3) in terms of feasibility:

1. Infeasibility:

$$[\![cost \leq k]\!] \wedge [\![x_1 \neq v_1]\!] \wedge \ldots \wedge [\![x_m \neq v_m]\!] \models \bot$$

2. Variable domain:

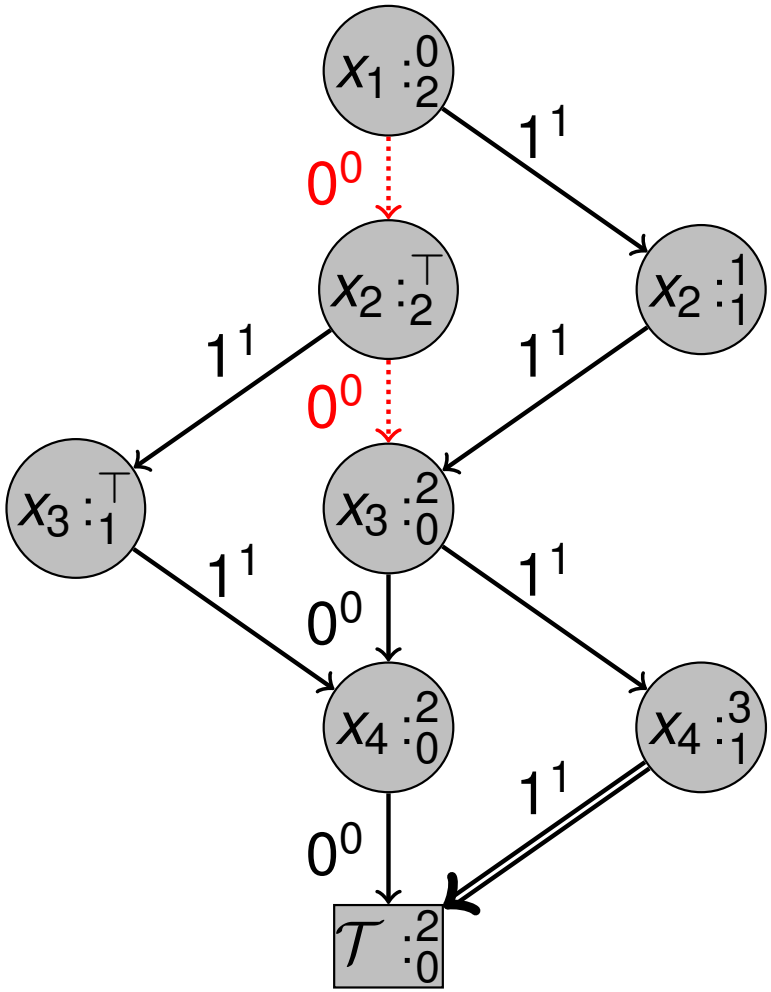
$$[\![cost \leq k]\!] \wedge [\![x_1 \neq v_1]\!] \wedge \ldots \wedge [\![x_m \neq v_m]\!] \wedge [\![x = v]\!] \models \bot$$

3. lb($cost$):

$$[\![x_1 \neq v_1]\!] \wedge \ldots \wedge [\![x_m \neq v_m]\!] \wedge [\![cost \leq k - 1]\!] \models \bot$$

Naive Explanation

- Always use current cost bound and current domain
- $[\![cost \leq 2]\!] \wedge [\![x_1 \neq 0]\!] \wedge [\![x_2 \neq 0]\!] \wedge [\![x_4 = 1]\!] \models \bot$
- Naive construction is valid.
- But produces terrible nogoods



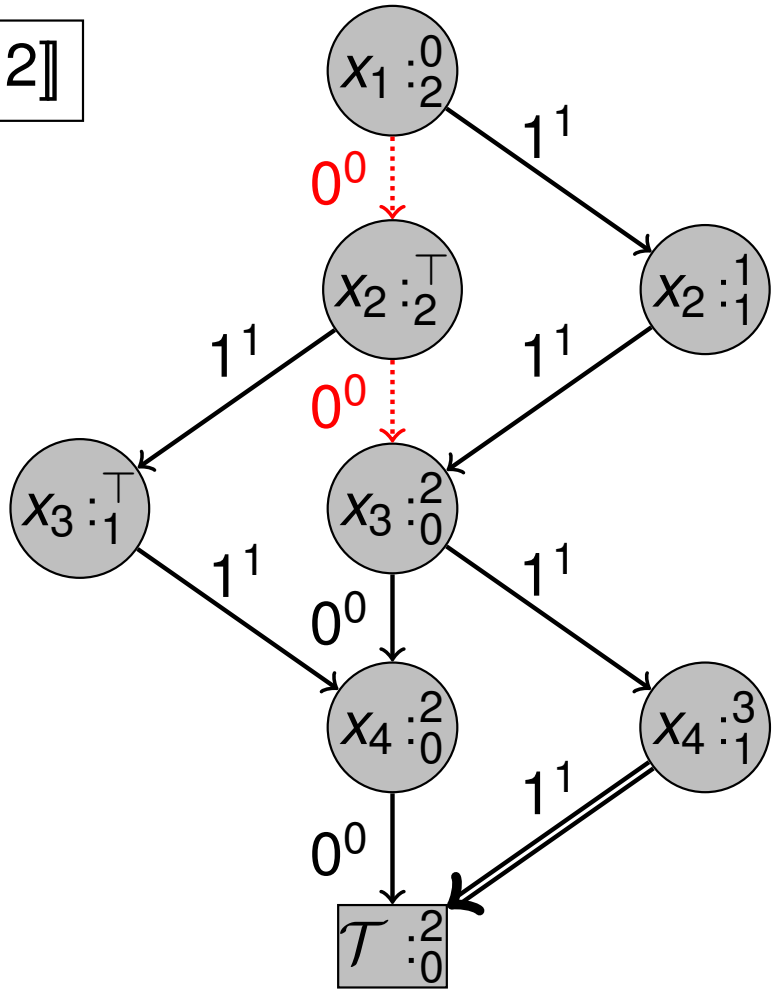
Minimal Explanation

Essentially a generalization of the MDD explanation algorithm.

- Compute the weakest possible cost bound k'
- Then progressively collect only those literals necessary to eliminate all paths of length $\leq k'$.

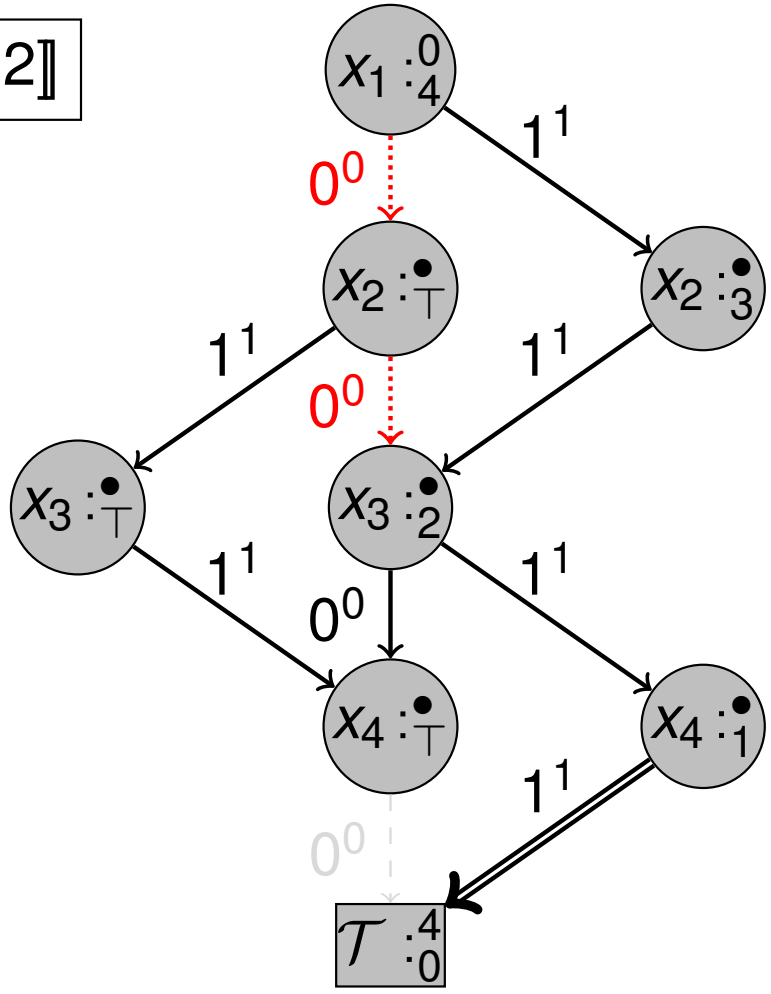
Example: Explaining $[\![x_4 \neq 1]\!]$

$[\![cost \leq 2]\!]$

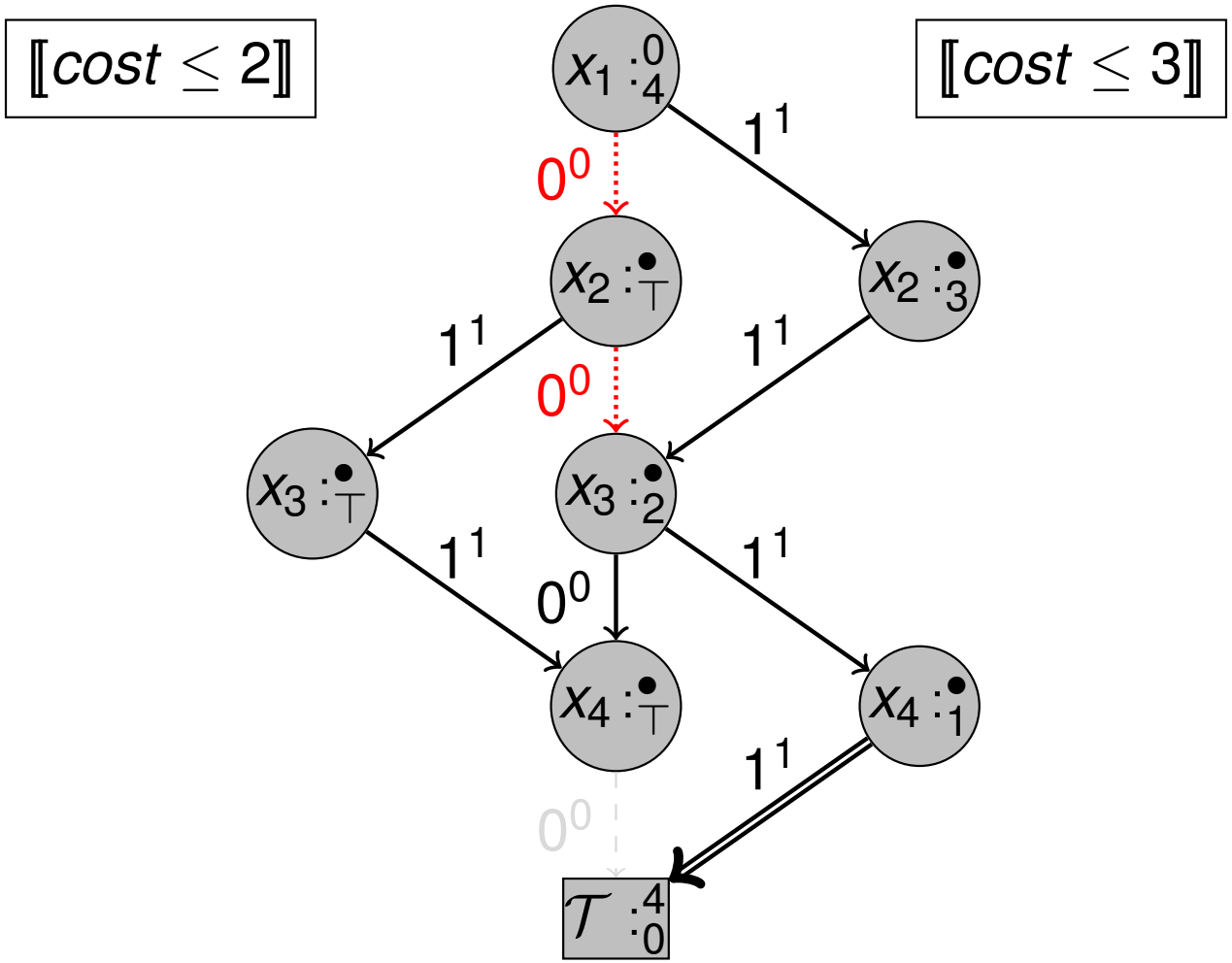


Example: Explaining $[\![x_4 \neq 1]\!]$

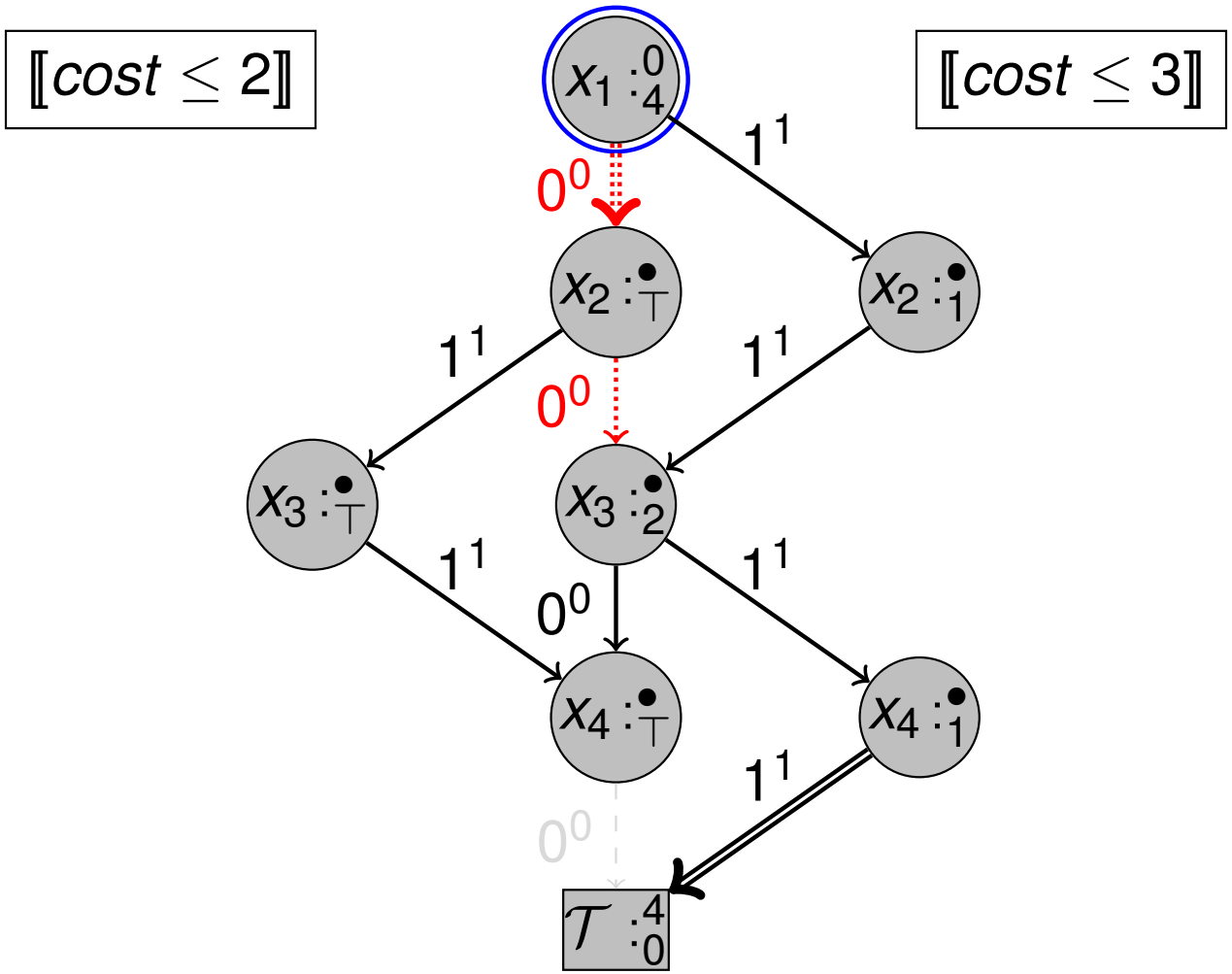
$[\![cost \leq 2]\!]$



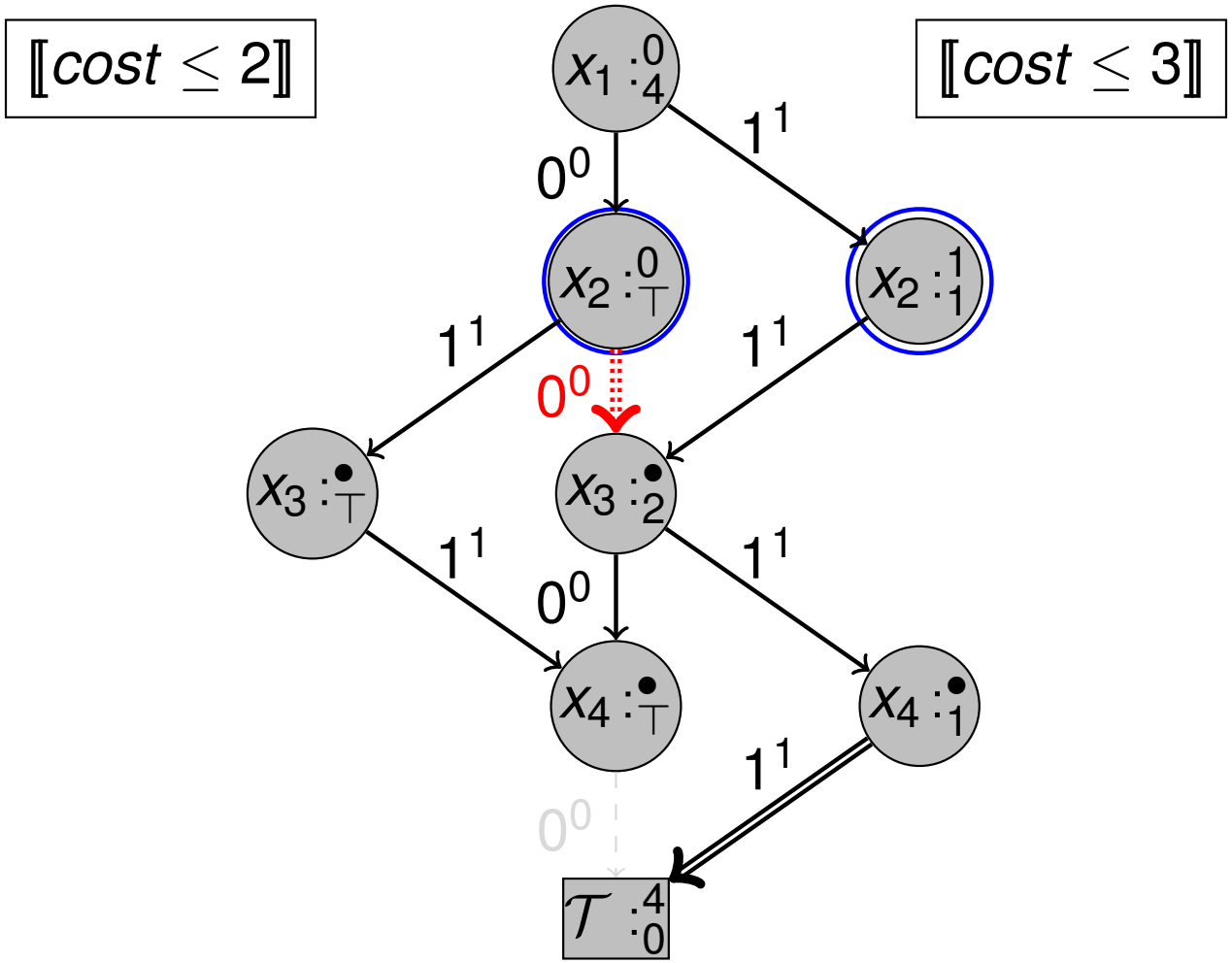
Example: Explaining $[\![x_4 \neq 1]\!]$



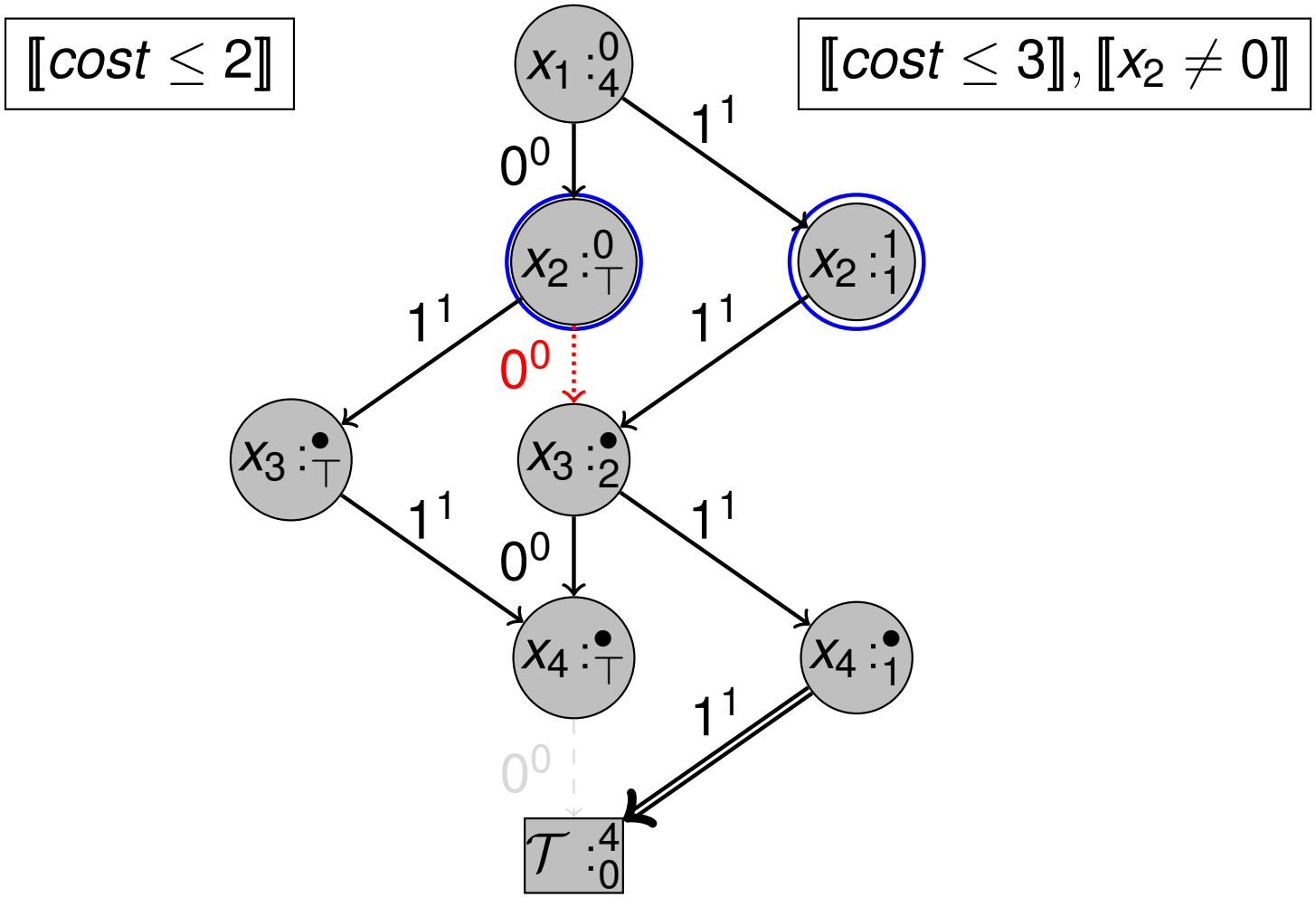
Example: Explaining $[\![x_4 \neq 1]\!]$



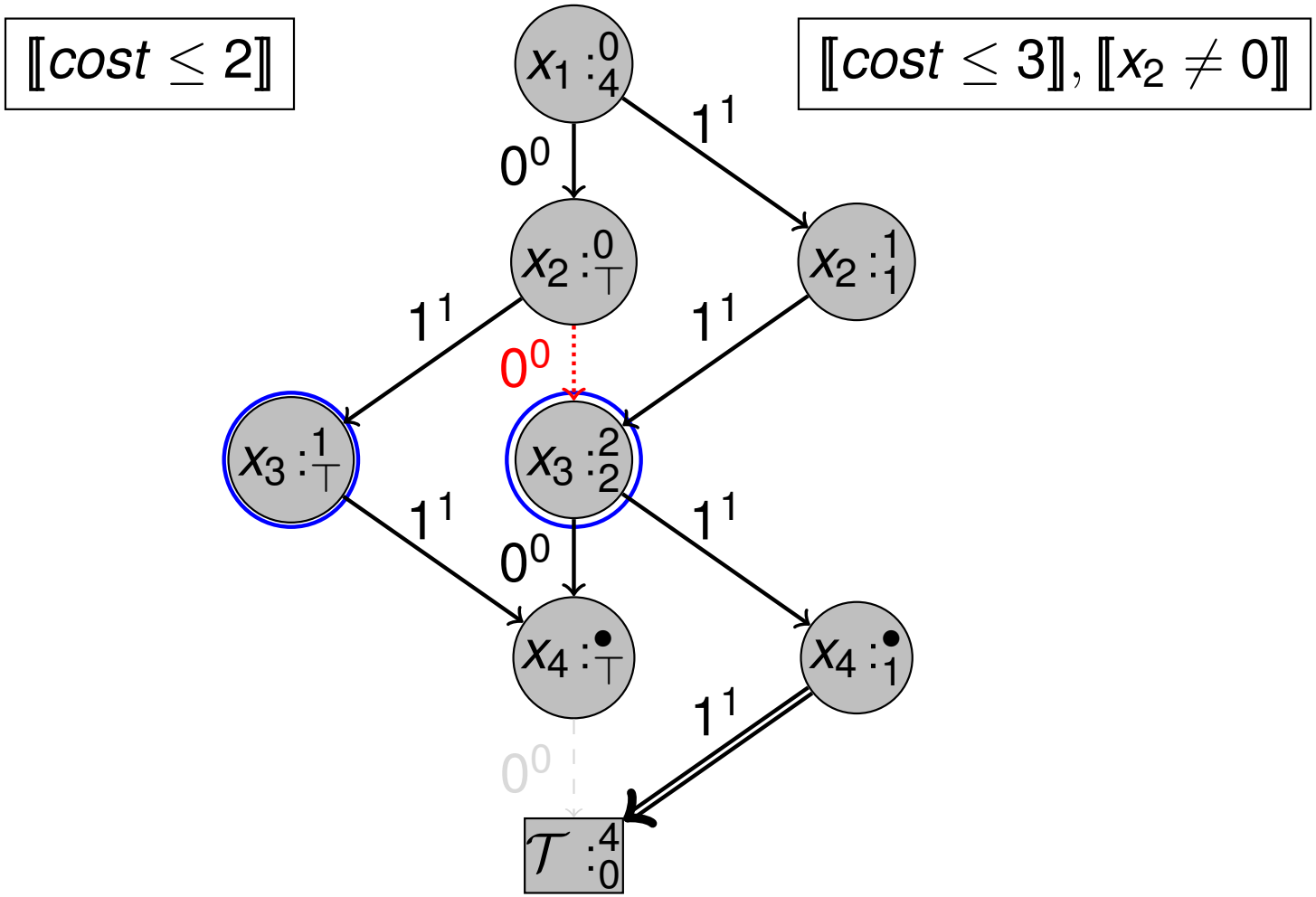
Example: Explaining $[\![x_4 \neq 1]\!]$



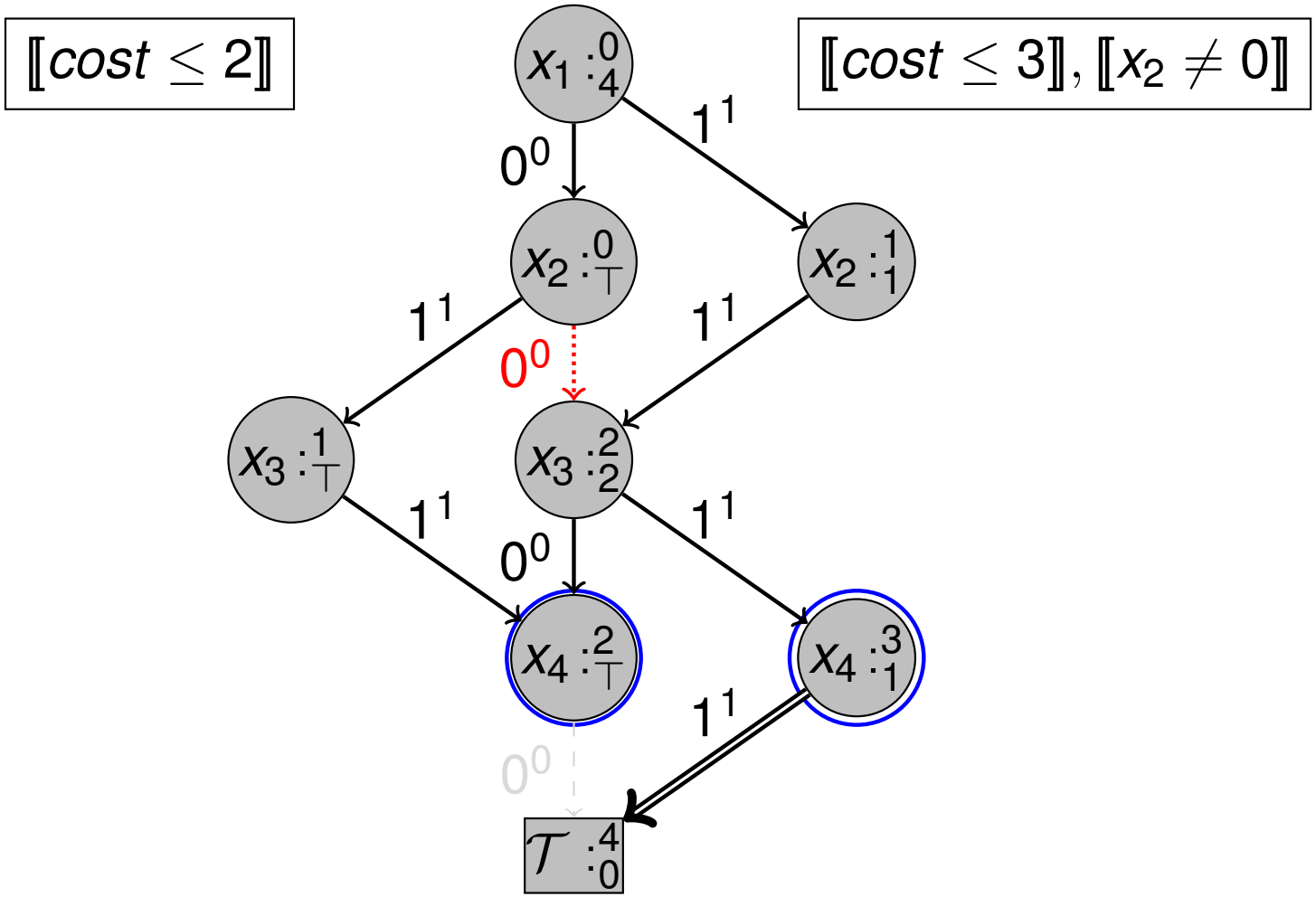
Example: Explaining $[\![x_4 \neq 1]\!]$



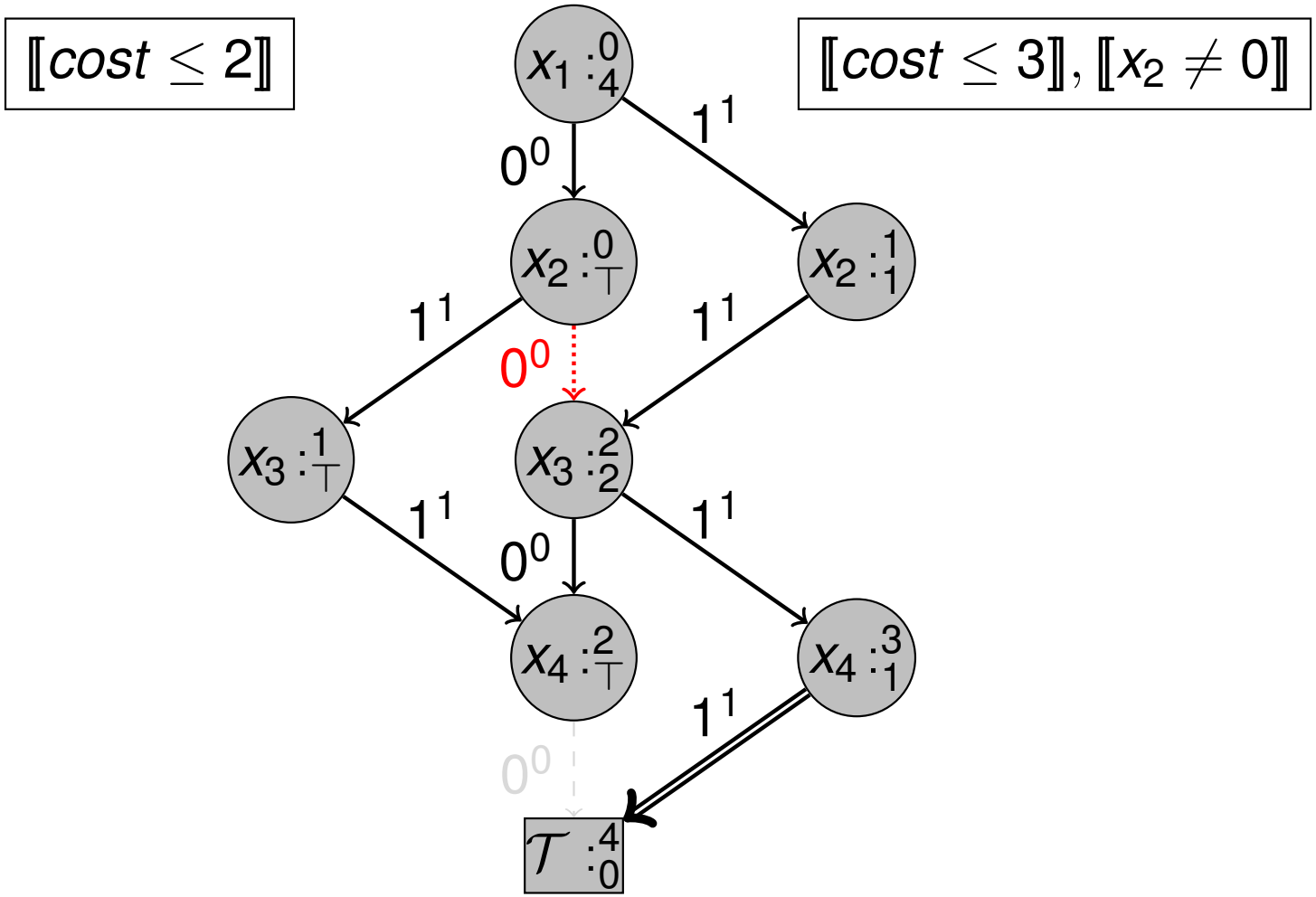
Example: Explaining $[\![x_4 \neq 1]\!]$



Example: Explaining $[\![x_4 \neq 1]\!]$



Example: Explaining $[\![x_4 \neq 1]\!]$



Explanation

We can cut off early by not exploring any unconstrained nodes.

Explanation

We can cut off early by not exploring any unconstrained nodes.

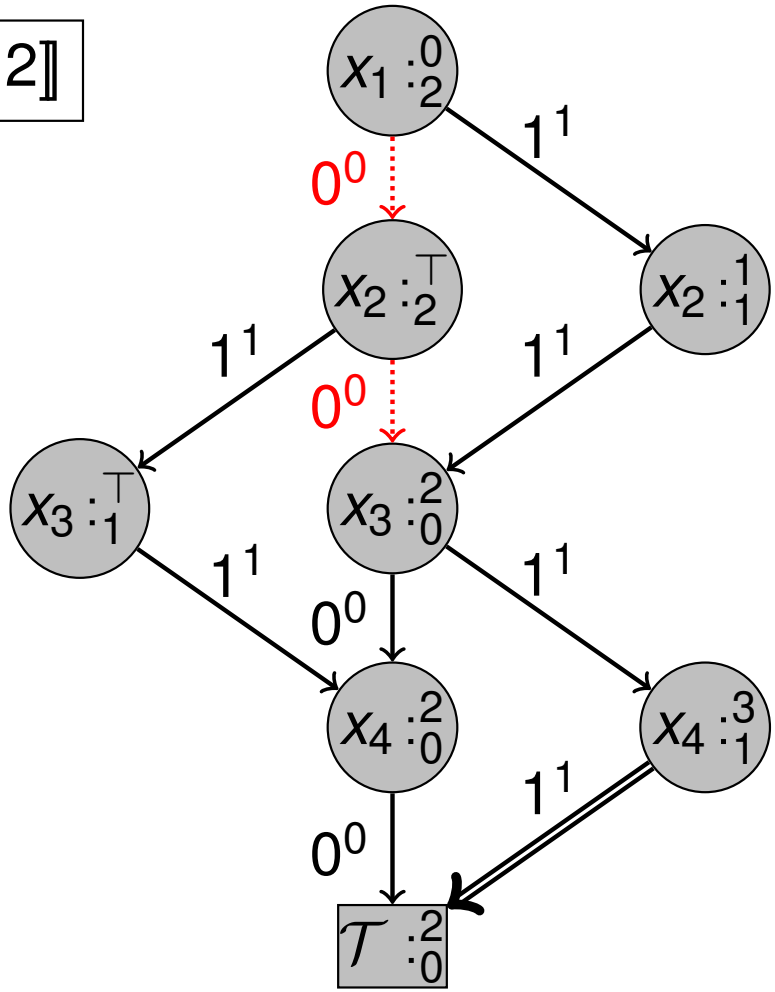
This still requires two full traversals of the graph in the worst case.
Can we do something cheaper?

Greedy Explanation

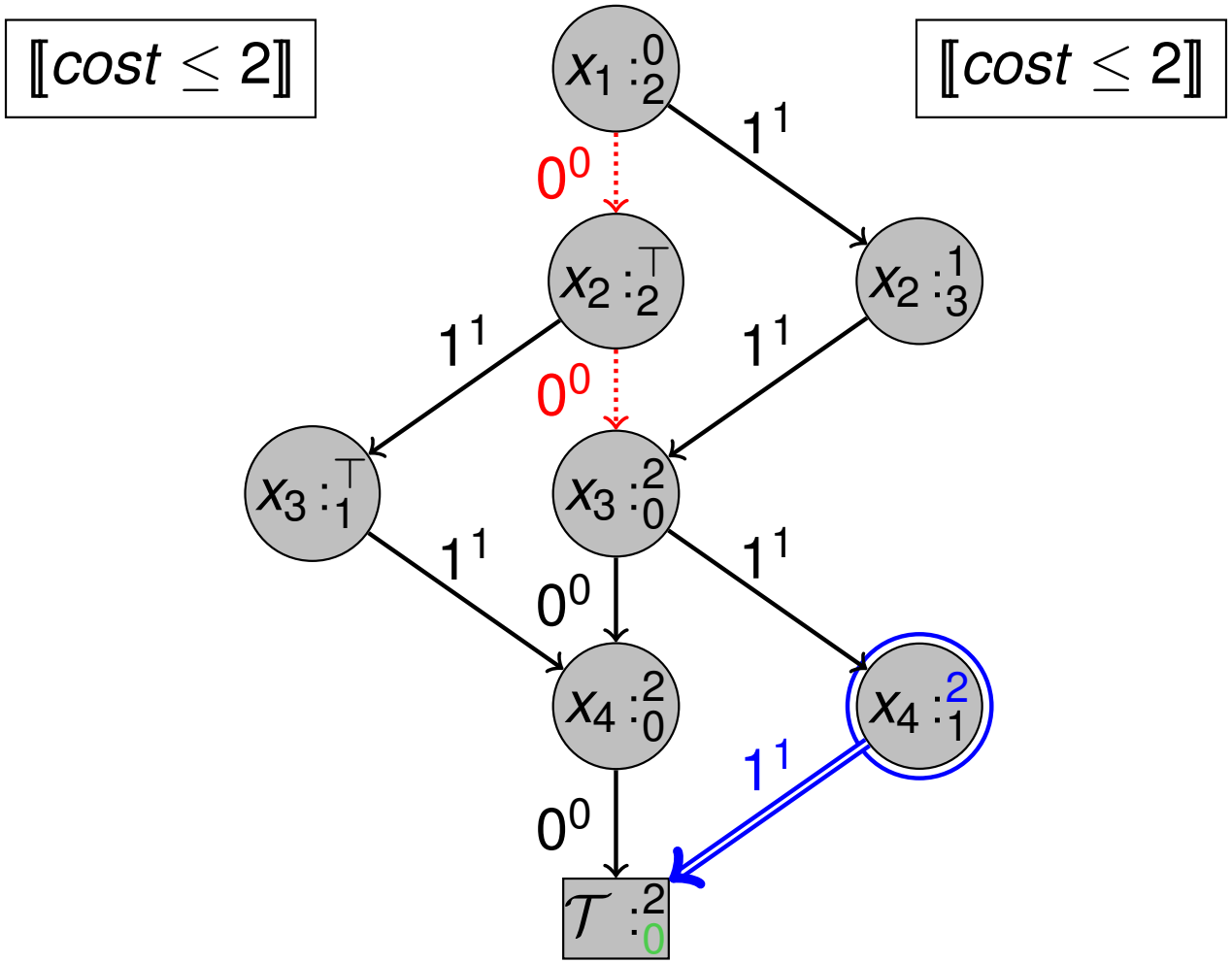
1. For each edge to be explained
   - compute the smallest in- and out- costs that guarantee infeasibility.
   - If there is slack, allocate it arbitrarily.
2. Sweep the frontier out from the current set of edges, collecting any literals necessary to ensure the path cost.

Example: Explaining $[\![x_4 \neq 1]\!]$ again

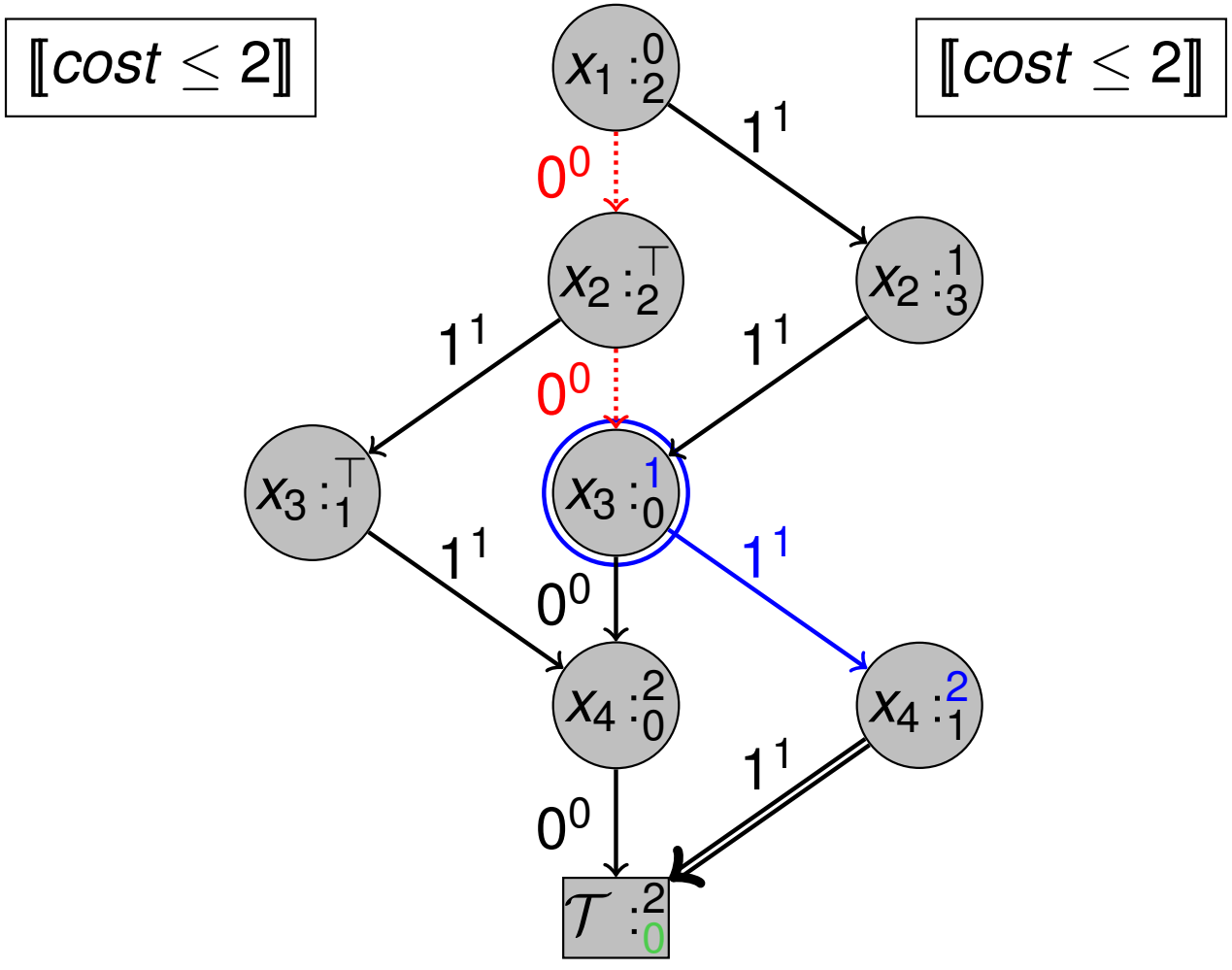
$[\![cost \leq 2]\!]$



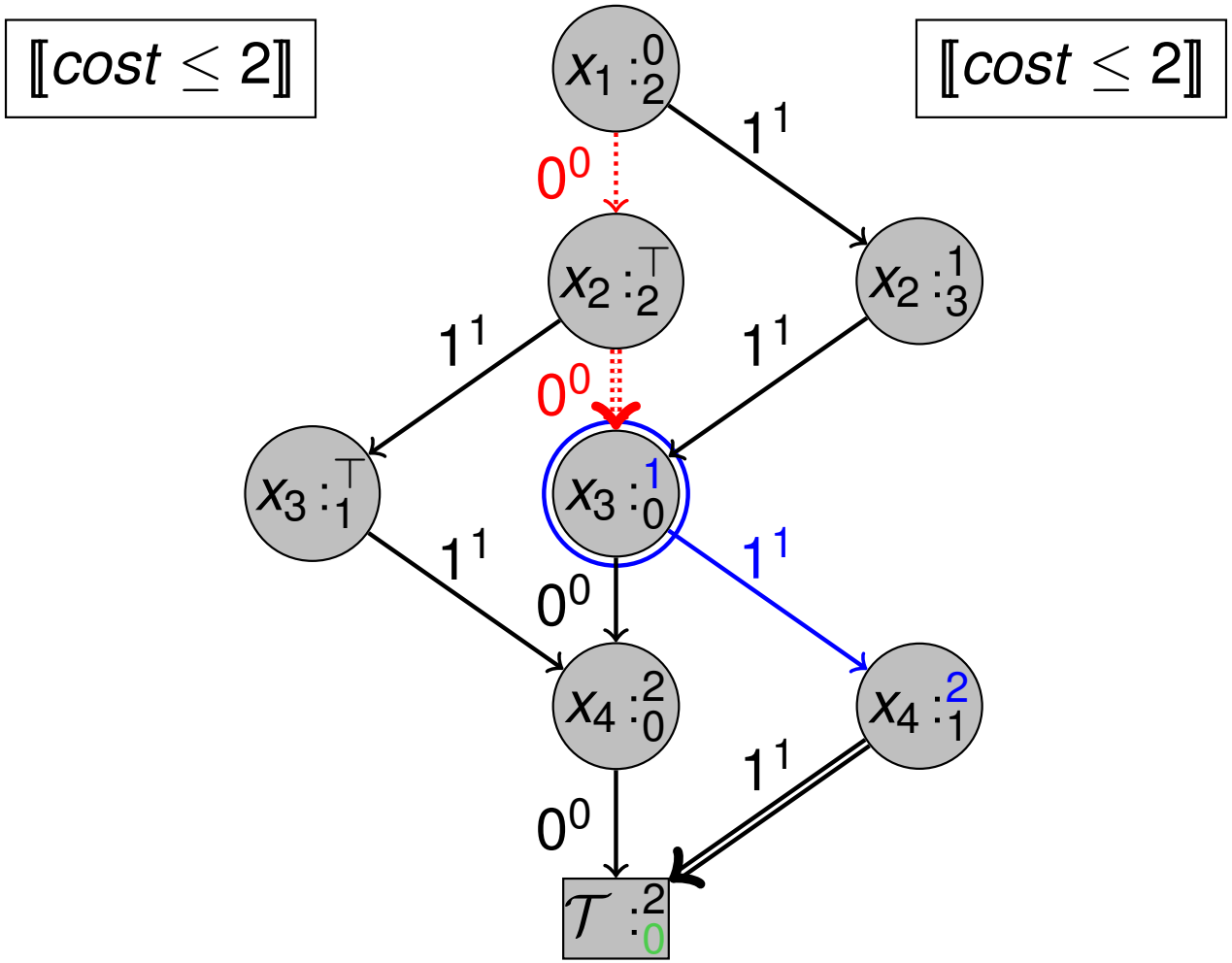
Example: Explaining $[\![x_4 \neq 1]\!]$ again



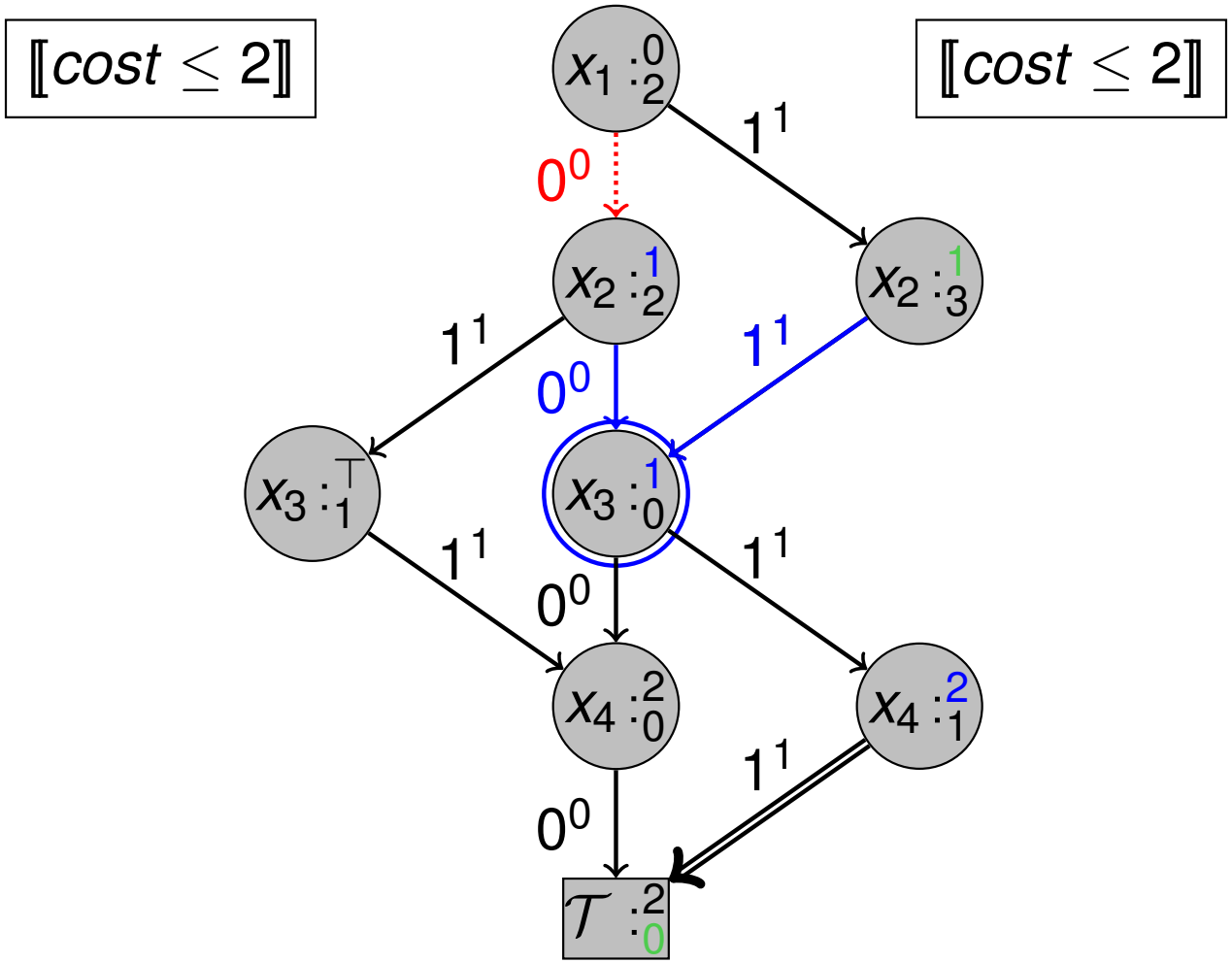
Example: Explaining $[\![x_4 \neq 1]\!]$ again



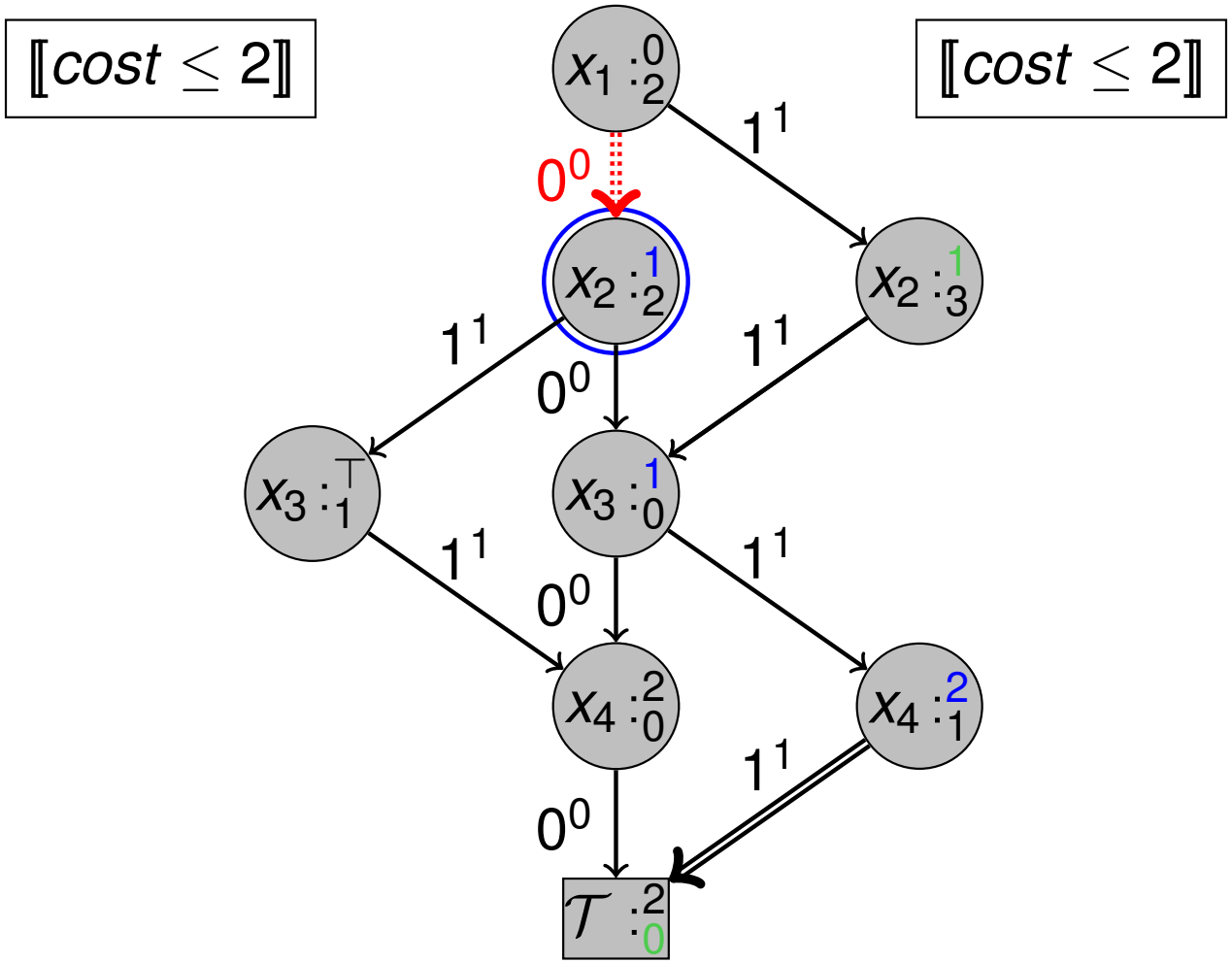
Example: Explaining $[\![x_4 \neq 1]\!]$ again



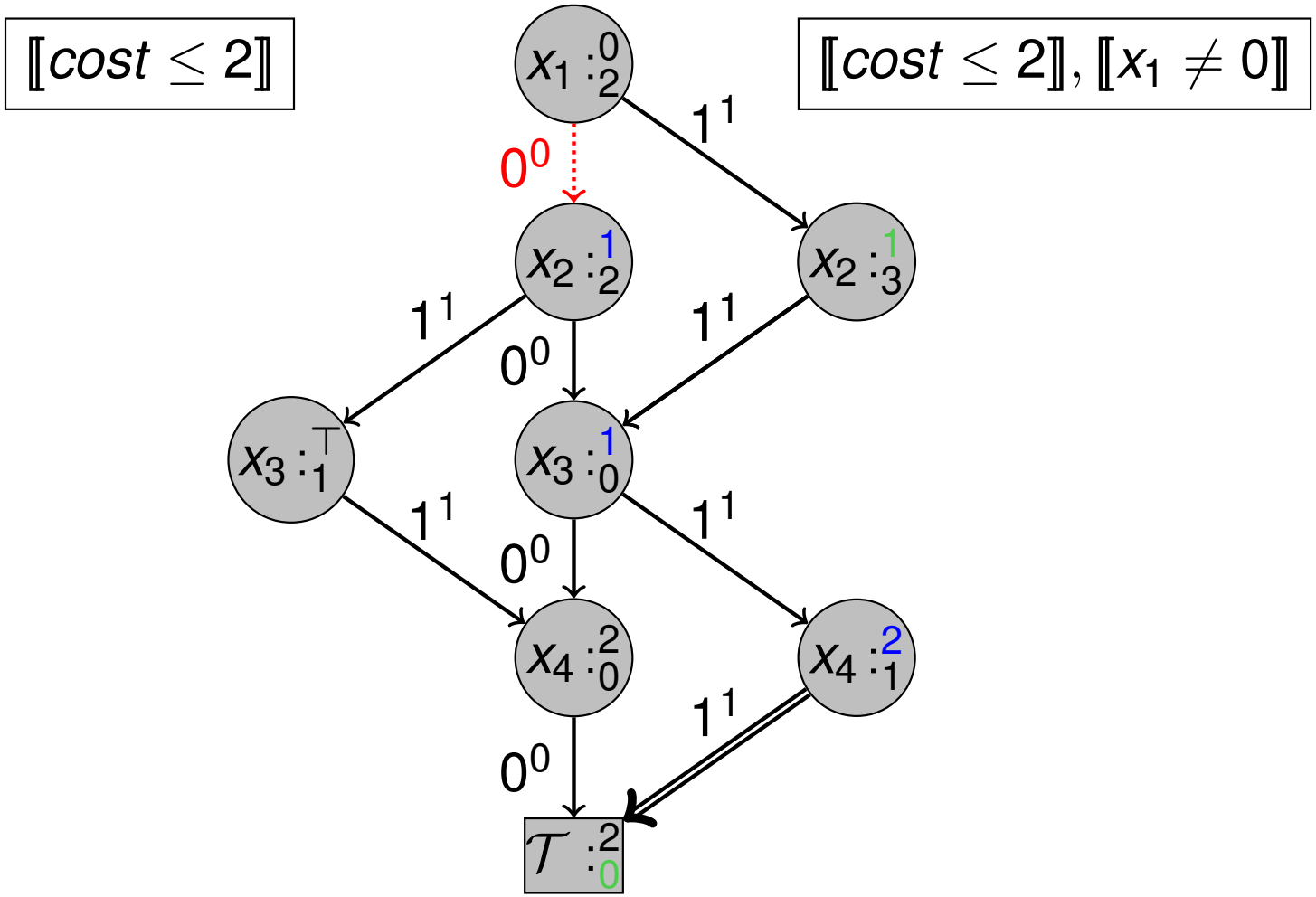
Example: Explaining $[\![x_4 \neq 1]\!]$ again



Example: Explaining $[\![x_4 \neq 1]\!]$ again



Example: Explaining $[\![x_4 \neq 1]\!]$ again



Aside: Decomposition

This seems like a lot of effort. Can't we just use a decomposition?

Aside: Decomposition

This seems like a lot of effort. Can't we just use a decomposition?

Better yet, intermediate variables can help learning!

Aside: Decomposition

This seems like a lot of effort. Can't we just use a decomposition?

Better yet, intermediate variables can help learning!

Introduce a variable for the cost to/from each node and edge, and add constraints maintaining the costs.

Aside: Decomposition

This seems like a lot of effort. Can't we just use a decomposition?

Better yet, intermediate variables can help learning!

Introduce a variable for the cost to/from each node and edge, and add constraints maintaining the costs. Except:

- To handle infeasible paths, the variable domains must be either:
  - Augmented with ∞
    Not currently supported by solvers
  - Very large (using a big-M encoding)
    Causes solver performance to degrade horrifically
- Also introduces a large number of such variables

Aside: Decomposition

This seems like a lot of effort. Can't we just use a decomposition?

Better yet, intermediate variables can help learning!

Introduce a variable for the cost to/from each node and edge, and add constraints maintaining the costs. Except:

- To handle infeasible paths, the variable domains must be either:
  - Augmented with ∞
    Not currently supported by solvers
  - Very large (using a big-M encoding)
    Causes solver performance to degrade horrifically
- Also introduces a large number of such variables

In practice, this performs worse than the separate decompositions.

Shift Scheduling

Given n employees and k activities, find the minimum cost schedule, subject to:

- At least c_{ij} staff performing activity i at time j.
- A full-time employee works between 6 and 8 hours
  - 1 hour for lunch
  - One 15 minute break both before and after lunch
- A part-time employee works between 3 and 5.75 hours
  - One 15 minute break
- An employee must work on an activity for at least 1 hour, and cannot switch without a break

Shift constraints are described with a context-free grammar, which is then converted into an MDD.
Coverage constraints are implemented with a BDD decomposition.

Experimental Results - Shift Scheduling

Table : Comparison of different methods on shift scheduling problems.

Inst.	dec$_{mdd}$		mdd		ev-mdd		ev-mdd$_l$	
	time	fails	time	fails	time	fails	time	fails
1,2,4	14.51	39700	3.49	21888	0.20	607	**0.17**	635
1,3,6	11.25	40675	19.00	76348	**0.87**	4045	0.91	4156
1,4,6	2.62	7582	0.69	3518	**0.11**	350	0.27	1077
1,5,5	0.41	1585	0.52	3955	0.07	239	**0.06**	238
1,6,6	0.40	1412	0.21	1161	**0.08**	249	0.11	413
1,7,8	4.03	13149	2.43	12046	**0.73**	3838	0.83	4279
1,8,3	0.85	5002	0.39	3606	**0.06**	219	0.07	262
1,10,9	17.55	44222	19.77	68688	**1.23**	5046	1.31	7419
2,1,5	0.14	691	0.24	1490	0.02	78	**0.01**	45
2,2,10	—	—	131.29	286747	**43.62**	99583	49.05	100958
2,3,6	188.77	187760	144.99	289568	**2.39**	6443	5.94	13695
2,4,11	—	—	391.59	918438	**42.38**	111567	92.89	220568
2,5,4	25.85	59635	12.18	50340	0.65	1545	**0.48**	1541
2,6,5	83.78	104911	30.27	80046	**6.18**	12100	7.63	16074
2,8,5	90.28	153331	34.69	110917	**4.99**	15507	10.02	26565
2,9,3	6.10	20472	9.17	42105	0.86	1898	**0.47**	1593
2,10,8	349.88	303227	95.61	168720	**8.85**	26331	17.22	37356
Total	—	—	896.53	2139581	**113.29**	289645	187.44	436874
Mean	—	—	52.74	125857.71	**6.66**	17037.94	11.03	25698.47
Geom.	—	—	7.92	31465.82	**0.86**	2667.65	1.03	3428.35

Results for COST-MDD decomposition are omitted; they were awful.

Conclusions

- Edge valued decision diagrams are a powerful generic constraint: generalizing COST-REGULAR
- Adding explanation significantly improves problem solving
  - Trust us!
- Interesting as opposed to MDD:
  - Minimal explanation are more beneficial than greedy explanations

Questions?

COST-MDD variations

- We have concentrated on COST-MDD$(G, [x_1, \ldots, x_m], \leq, cost)$
  - Requires the cost to be less than a bound.
  - Propagation is domain consistent
  - Most useful in cost minimization problems
- Other variations:
  - COST-MDD$(G, [x_1, \ldots, x_m], \geq, cost)$
    - essentially the same, negate the arc costs and cost variable!
  - COST-MDD$(G, [x_1, \ldots, x_m], =, cost)$
    - Domain propagation is NP-hard
    - In practice treat as the conjunction COST-MDD$(G, [x_1, \ldots, x_m], \leq, cost)$ $\wedge$ COST-MDD$(G, [x_1, \ldots, x_m], \geq, cost)$