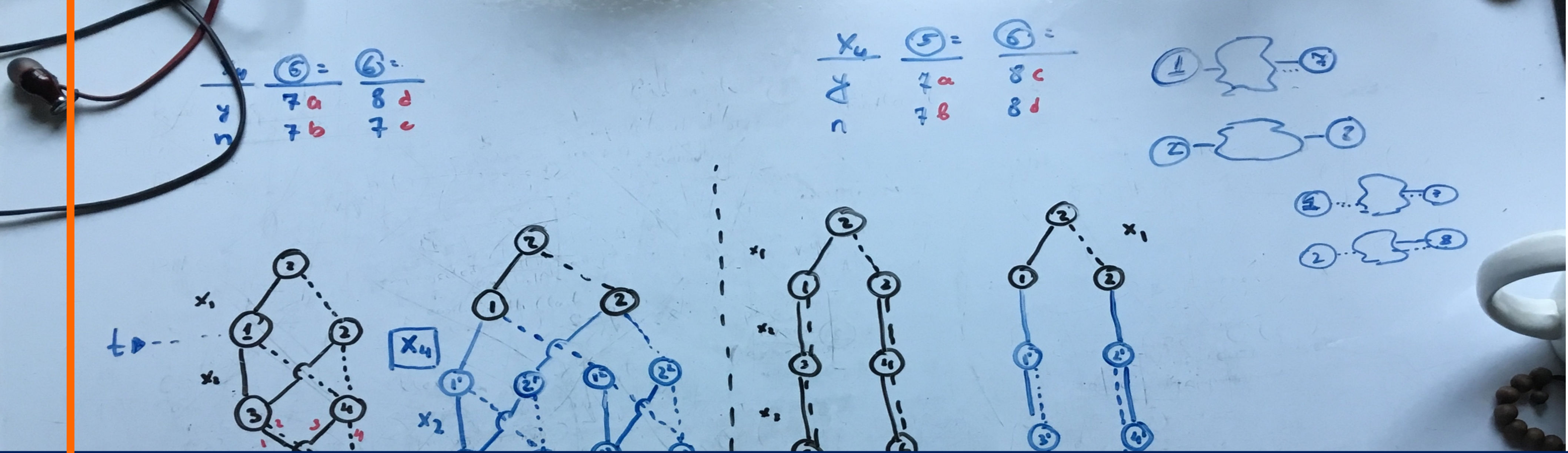


On Aligning Non-Order-Associated Binary Decision Diagrams

Alexey A. Bochkarev, J. Cole Smith

INFORMS (virtual) Annual Meeting 2020

Outline *(15 minutes)*

- General intro
  - BDD: what and why?
  - Order of variables matters
- Making BDDs “order-associated”
  - Problem formulation
  - BDD transformations
  - Idea: problem simplification
- What we did: branch-and-bound
- How it worked: numerical results
- Further research

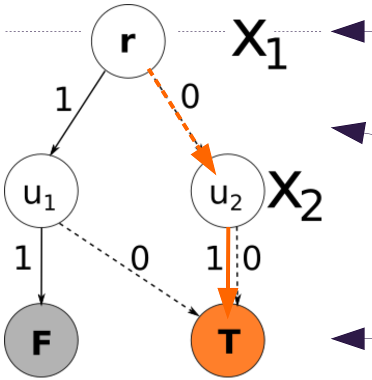
High-level intro: (Ordered) Binary Decision Diagrams

WHAT is a BDD?

- A (maybe weighted) layered DAG
- Two outgoing arcs from each node
- One root, two terminal nodes (**T**, **F**)

WHY a BDD?

- Can encode a Boolean function …
- … or a combinatorial opt problem.



Each **layer** corresponds to a decision – say, choice of a variable value (0 or 1)

Arcs: “1”-arc = a “1” decision,
”0”-arc = a “0” decision.

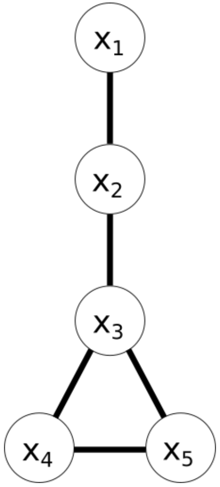
A **path** from root to T or F corresponds to an assignment of all variables.

Note: *arc weights can be chosen arbitrarily; more details: e.g., [Knuth2009]*

Order of variables matters: a MIS example.

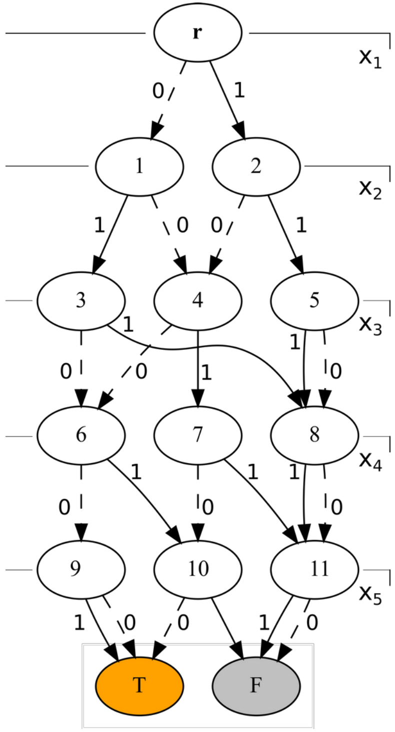
(min BDD is a well-known NP-hard problem)

- Consider encoding a Max Independent Set (MIS) problem for a graph:

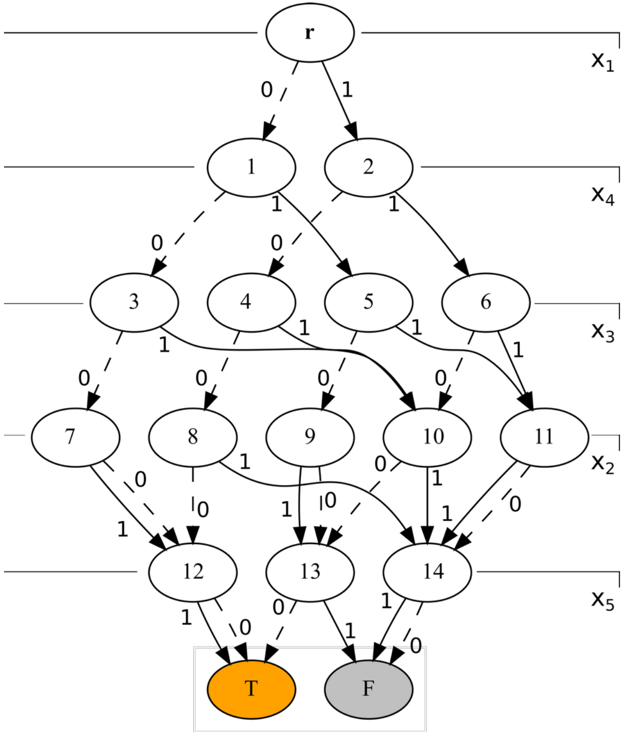


- Define x_i=*1 iff* we pick node x_i to the solution.

Possible BDD representations:



14 nodes



17 nodes

Note: *more details: e.g., [Bergman2016]*

The *same* order of variables also does matter.

- Many problems can be reformulated as a set of (interconnected) instances over a collection of BDDs.
- Under a structured ordering property – if the diagrams share the same order of variables – [Lozano2020] proposed an algorithm that might perform well in practice.
- So, **finding a good shared variable order might allow applying some new classes of algorithms**

The problem of aligning BDDs

Let's call $\mathrm{T}^*[D, \vec{v}]$ A (properly) revised, or "transformed" version of a BDD *D* to variable order ***v***=(v_1, …, v_N)

THEN: $$s^* = \min_{\vec{v}} \left(\overbrace{|T^*[A, \vec{v}]|} + \overbrace{|T^*[B, \vec{v}]|} \right)$$

An "Alignment problem", or "Multiple Variable Order" problem[Cabodi98] = *to align, minimizing* ***the total size* of two BDDs.***

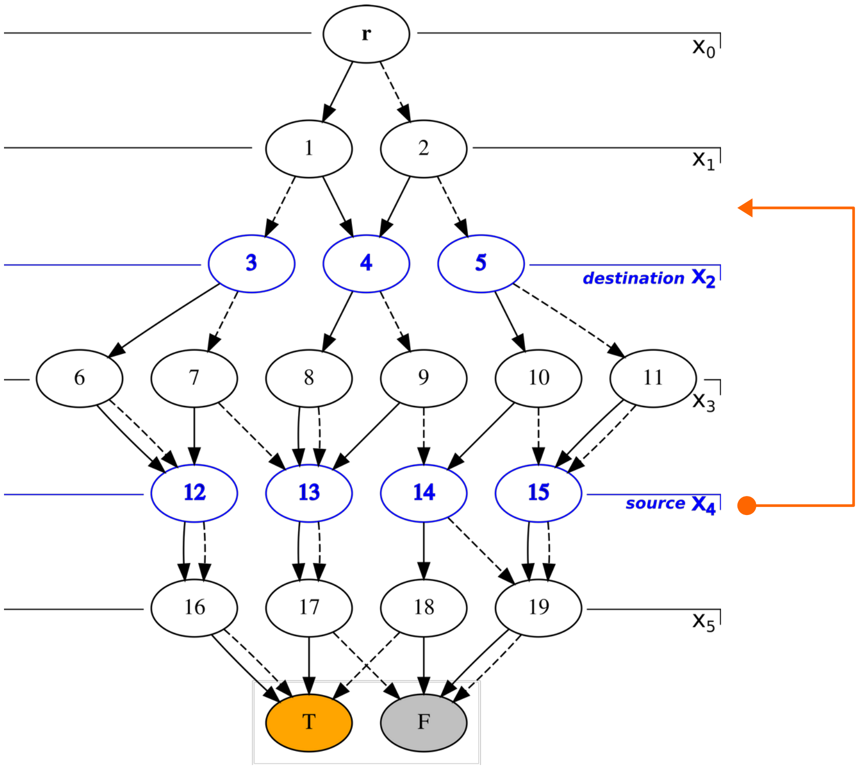
Obviously, **NP-hard** (since min single-BDD is NP-hard[Bollig96])

* here **size** of a BDD *D*, denoted |D|:= total number of nodes = sum of "layer widths"

Some background

- There is a vast literature on single-BDD minimization
- One of the central ideas: “Dynamic variable reordering” aka **“Sifting”**[Rudell93].
- We could transform both diagrams to some starting order and apply one of these methods (we will use it as baseline)
- Some other approaches were presented earlier [Cabodi98],[Scholl2001]
- However, all these deal with the BDDs directly (which can grow large).
- **The purpose of this work:** try to avoid some BDD manipulations by introducing an intermediate, “simpler” problem.

First: how BDDs are "transformed"?



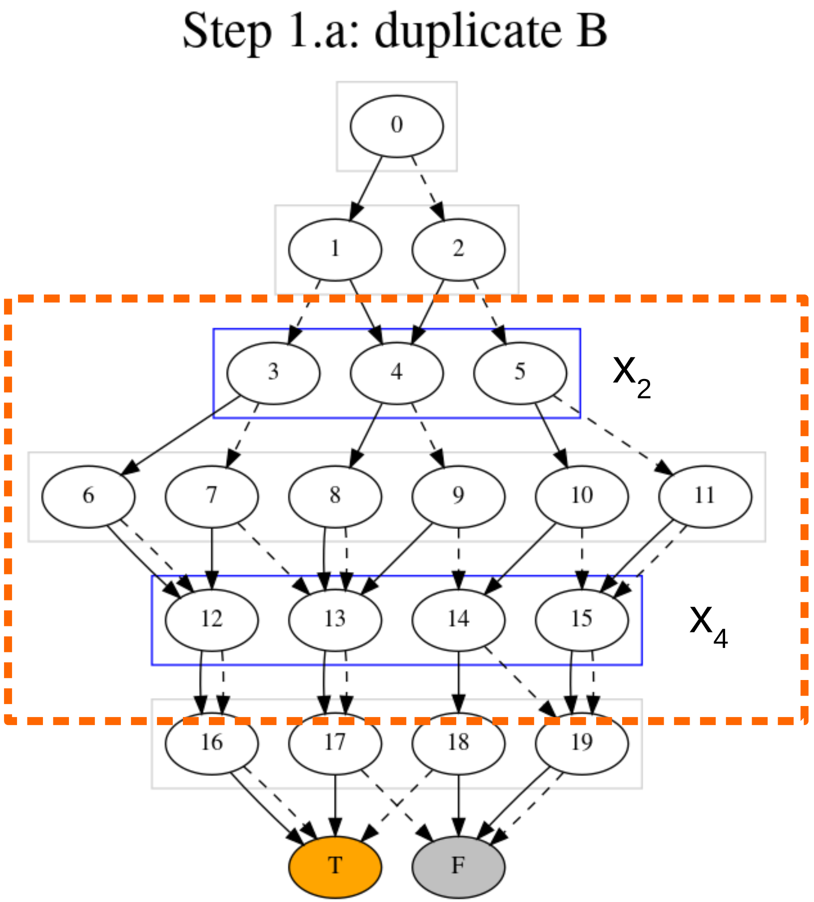
Consider moving x_4 right before x_2.
How the BDD would "**transform**"?

That is, change so that:
(1) paths with the corresponding x choices end in the same destination (T or F)

(2) we could assign arc costs so that corresponding paths' costs are the same.

How to build the "transformation"?

We start with the initial diagram...

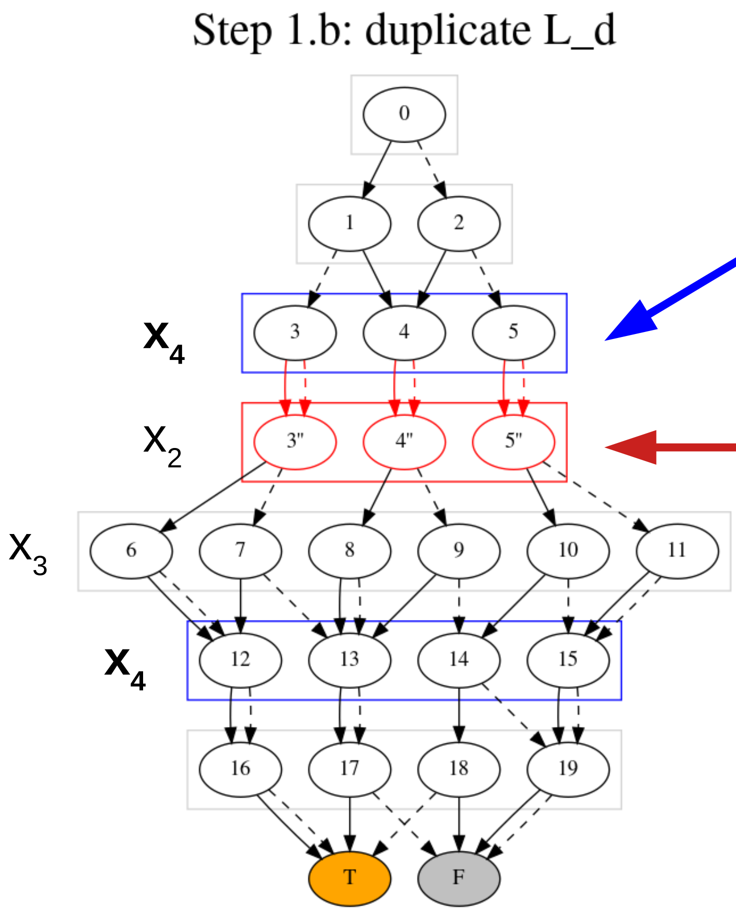


The idea: create two copies of the **changing area:**

- one for $x_4 = 0$,
- one for $x_4 = 1$,

and then just “wire” them to the initial BDD, so all paths would work as needed.

Duplicate the source layer

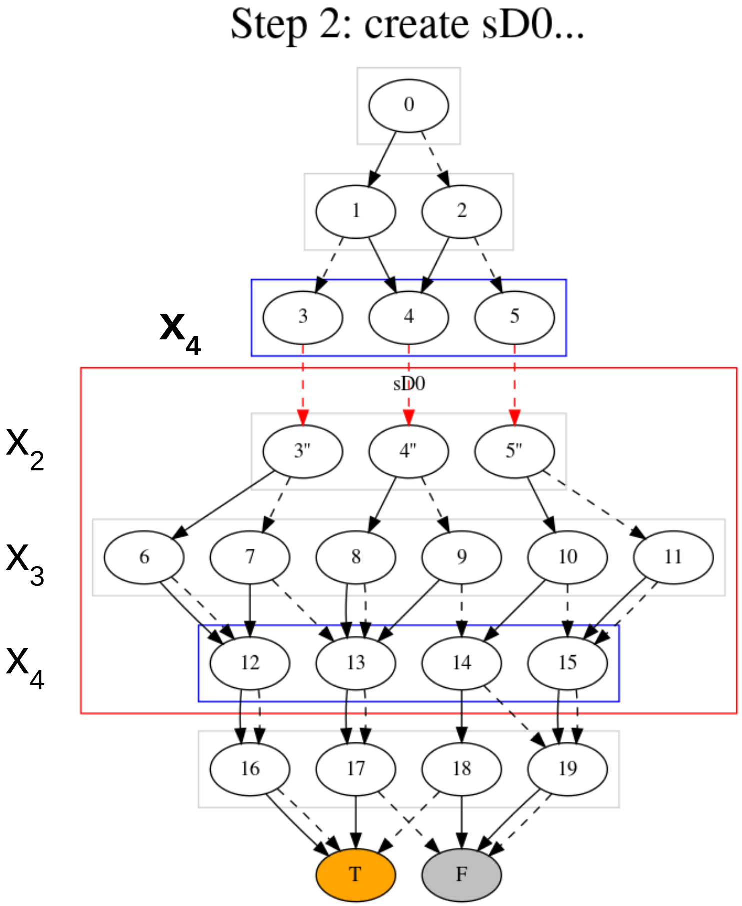


This will serve as x_4 now.

This one will still serve as x_2.

Note that we have two x_4 layers now: we will fix this soon.

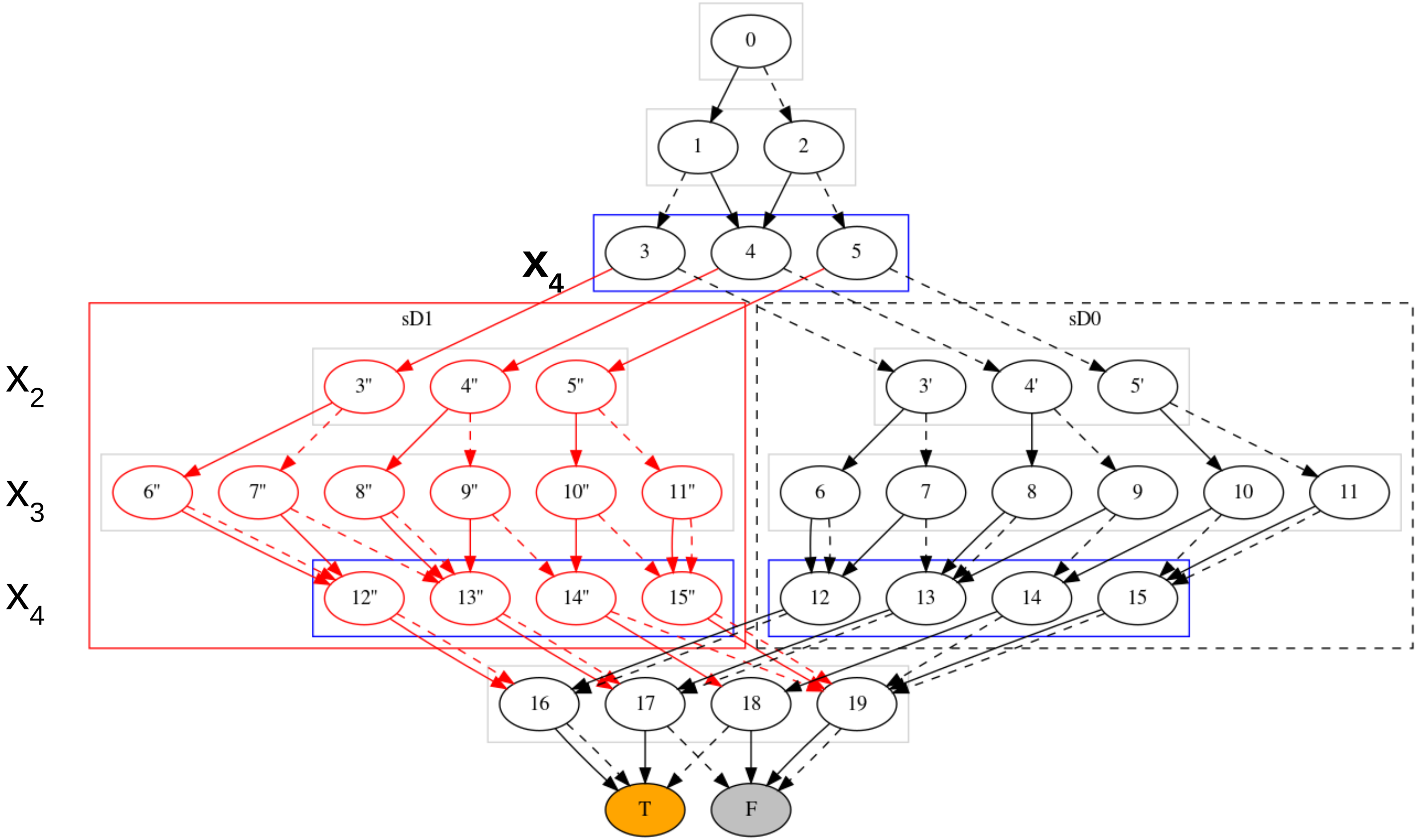
Create "x_4=0" copy



This will be our "x_4=0" copy

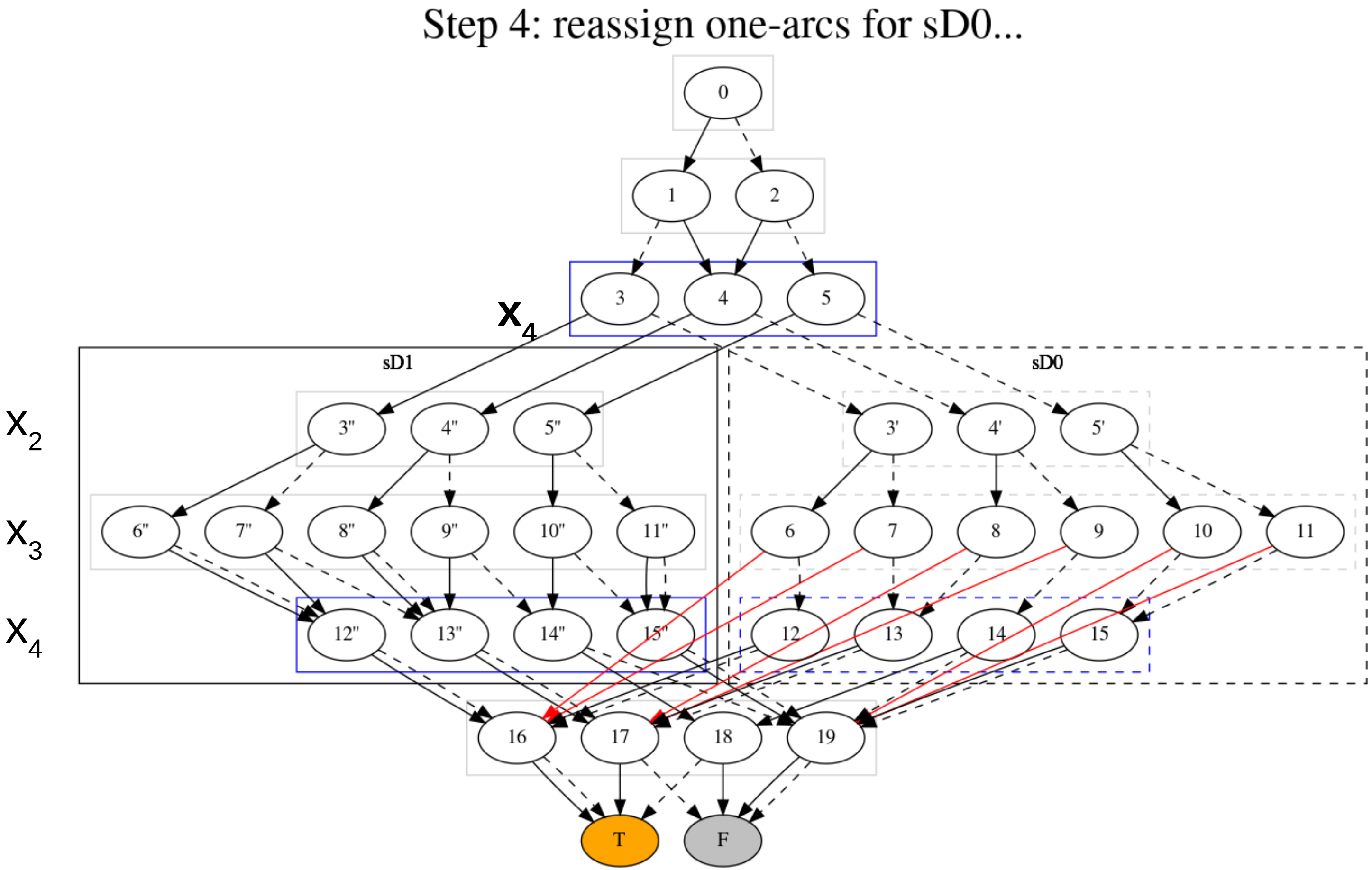
Create “x_4=1” copy

Step 2: ... and create sD1

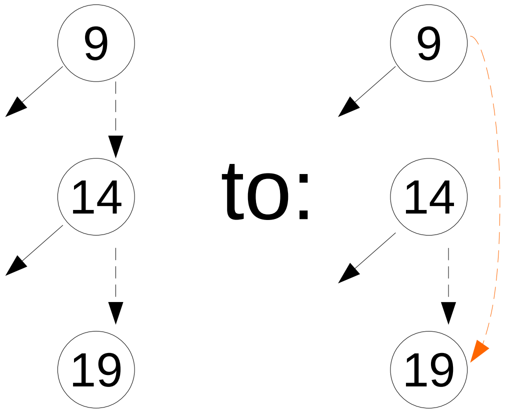


Now we duplicate the sub-graph to create **“x_4=1” copy**

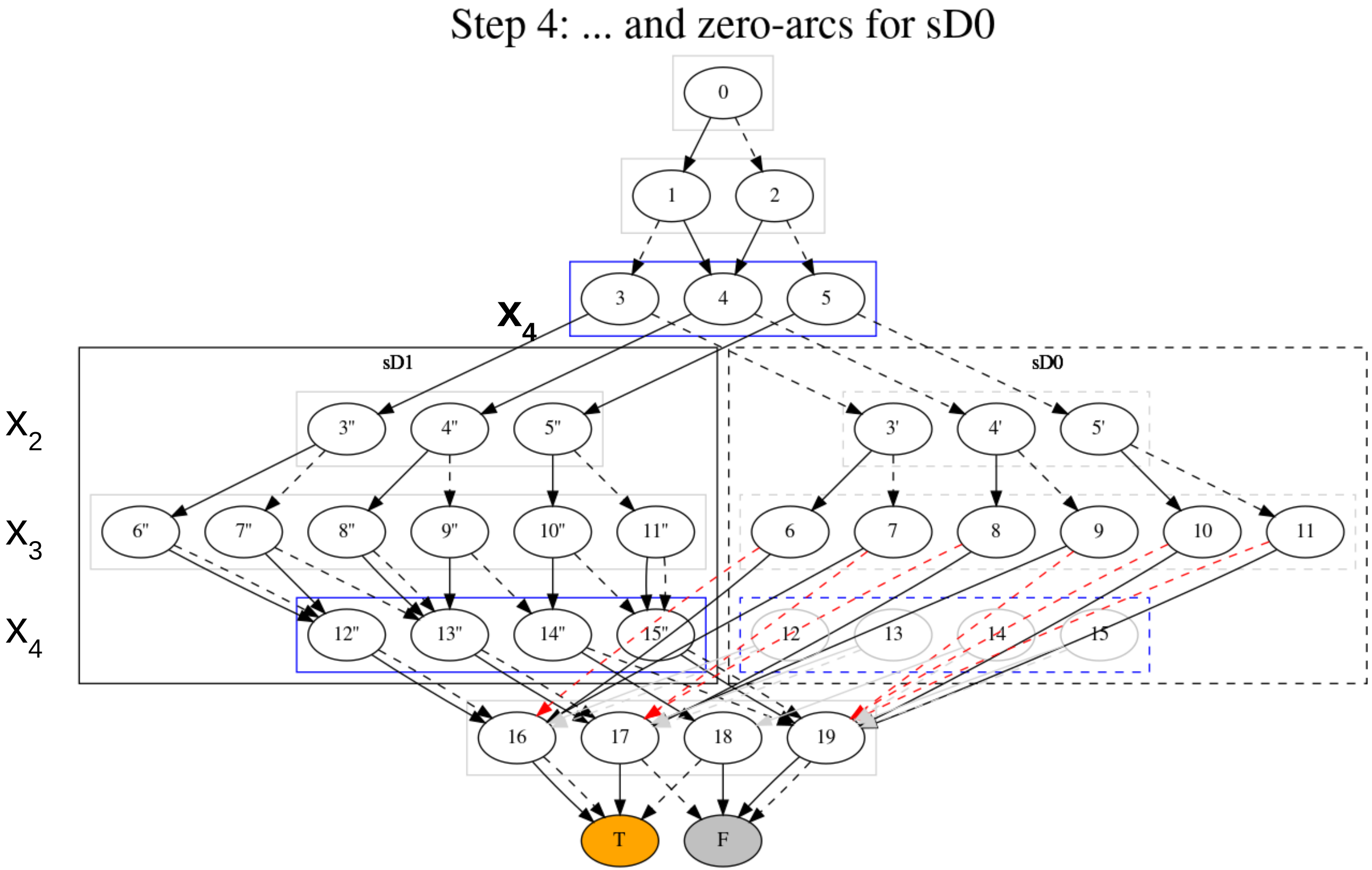
Finally, reassign the arcs. (1/4)



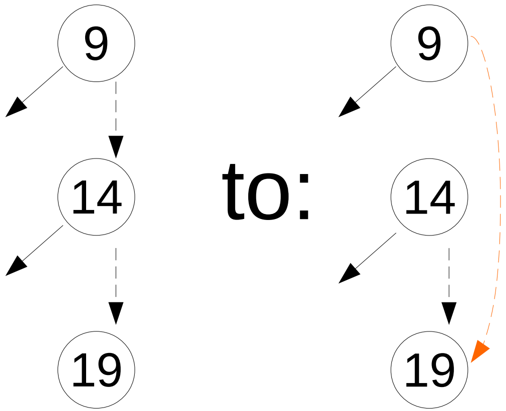
E.g., on x_4=0 copy we can "shortcut"



Finally, reassign the arcs. (2/4)

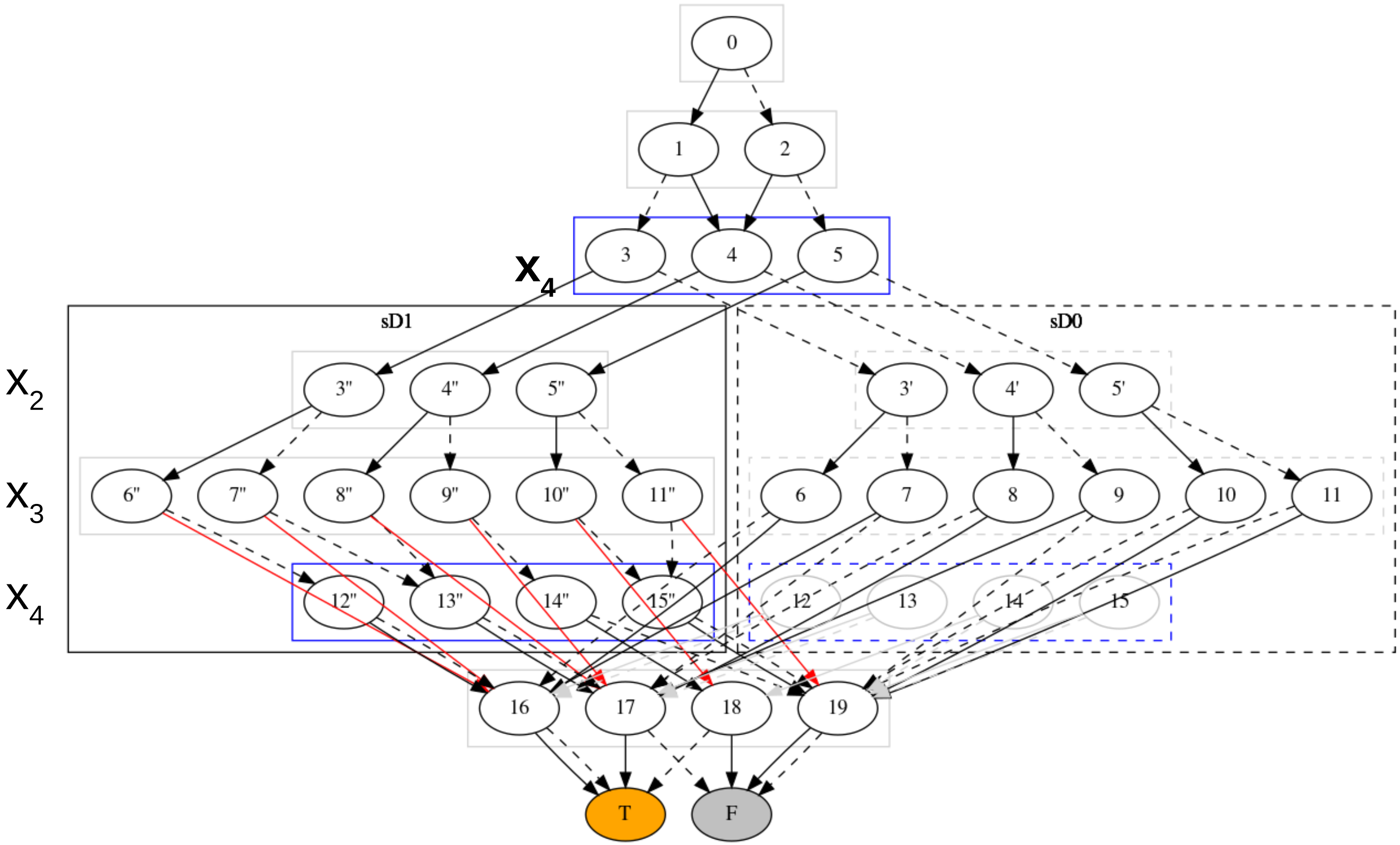


E.g., on x_4=0 copy we can "shortcut":

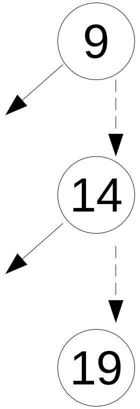


Finally, reassign the arcs. (3/4)

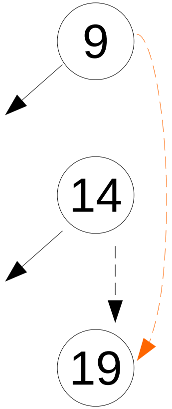
Step 5: reassign one-arcs for sD1...



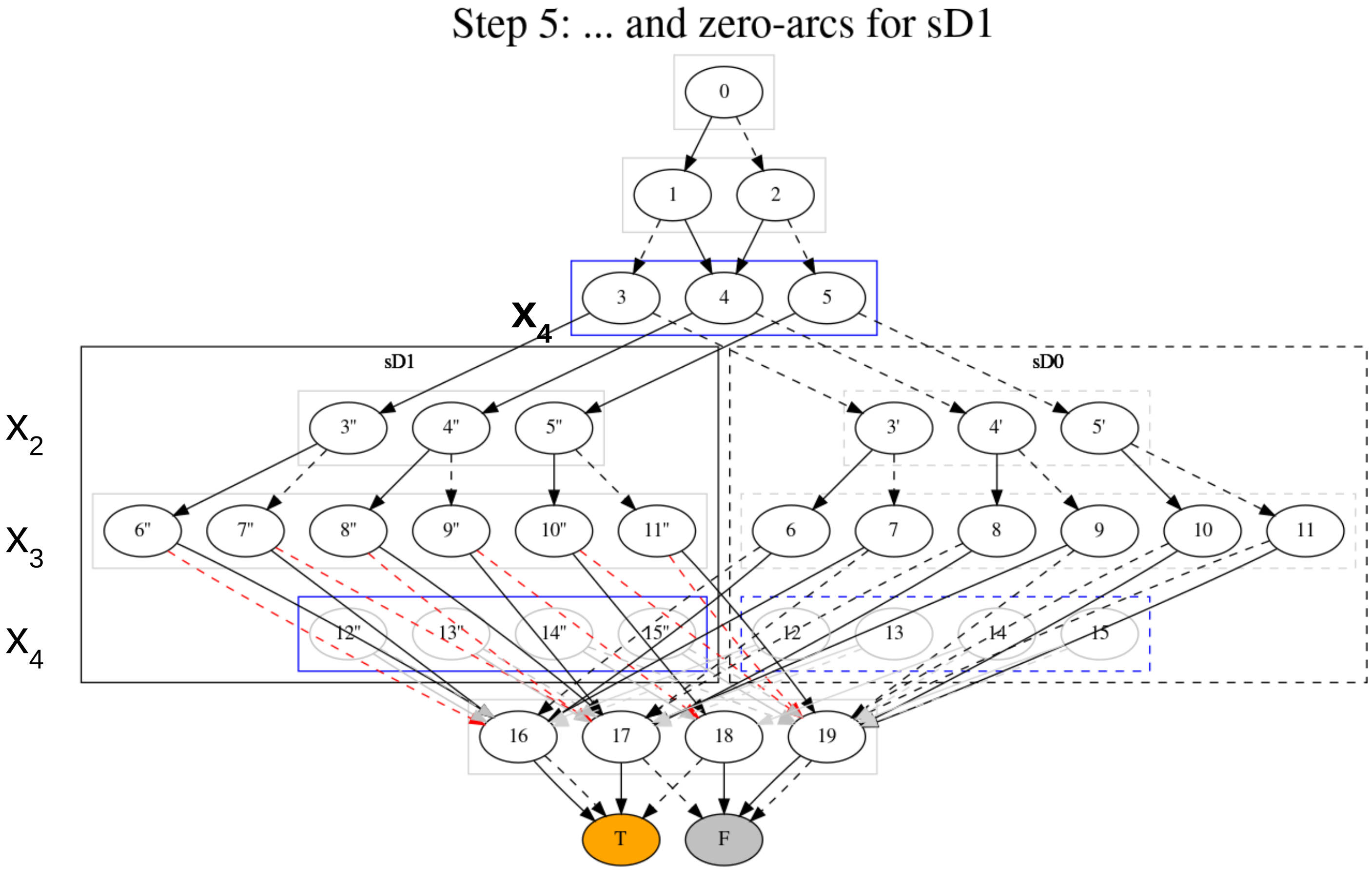
E.g., on x_4=0 copy we can "shortcut":



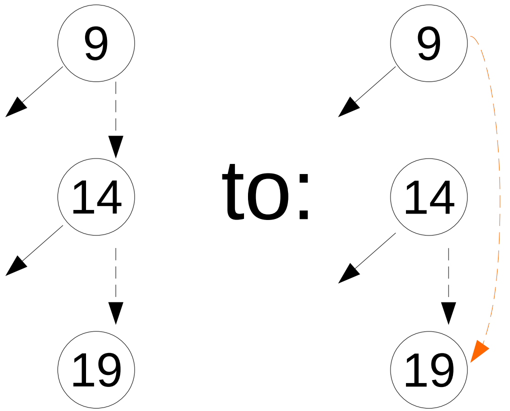
to:



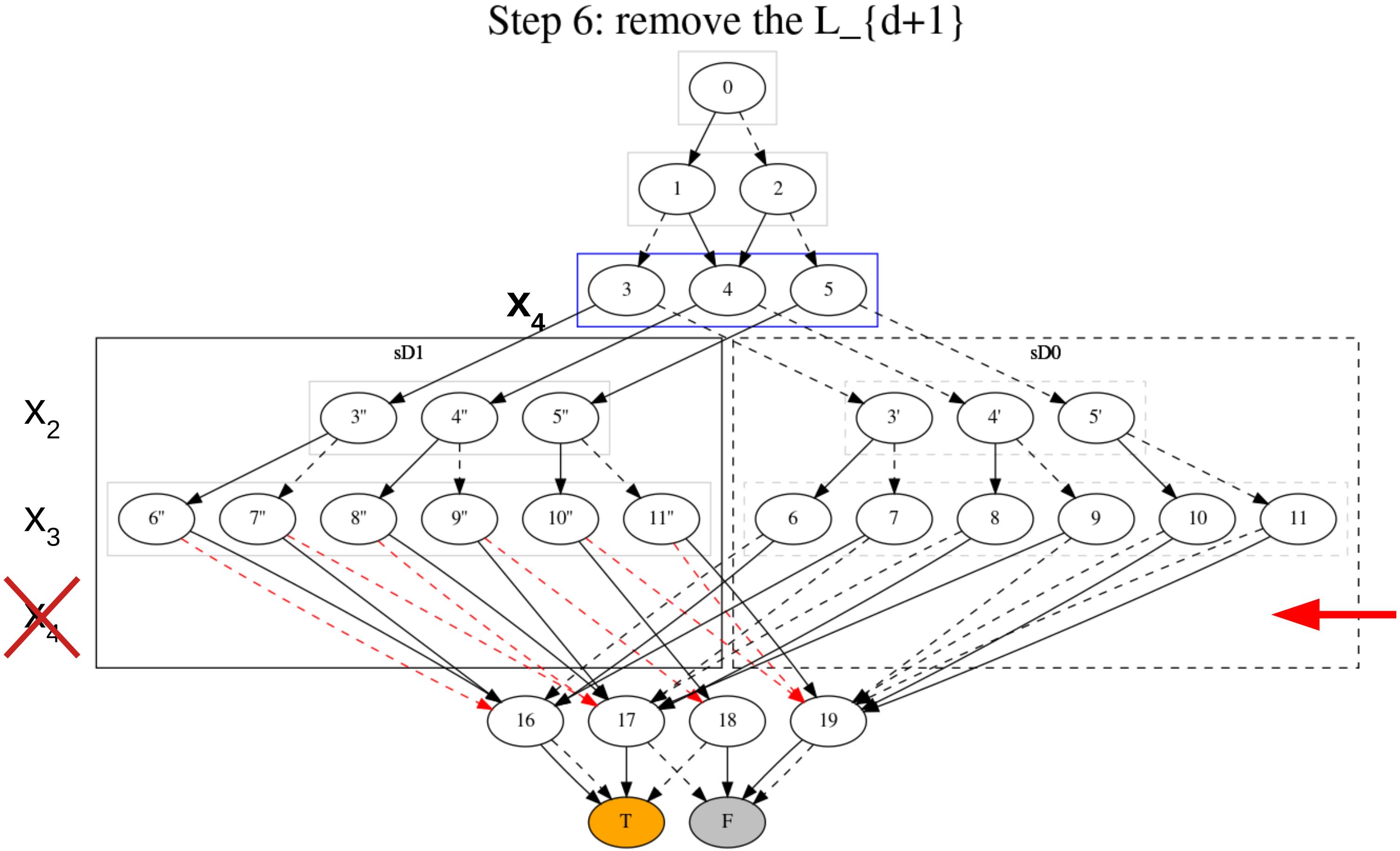
Finally, reassign the arcs. (4/4)



E.g., on x_4=0 copy we can "shortcut":

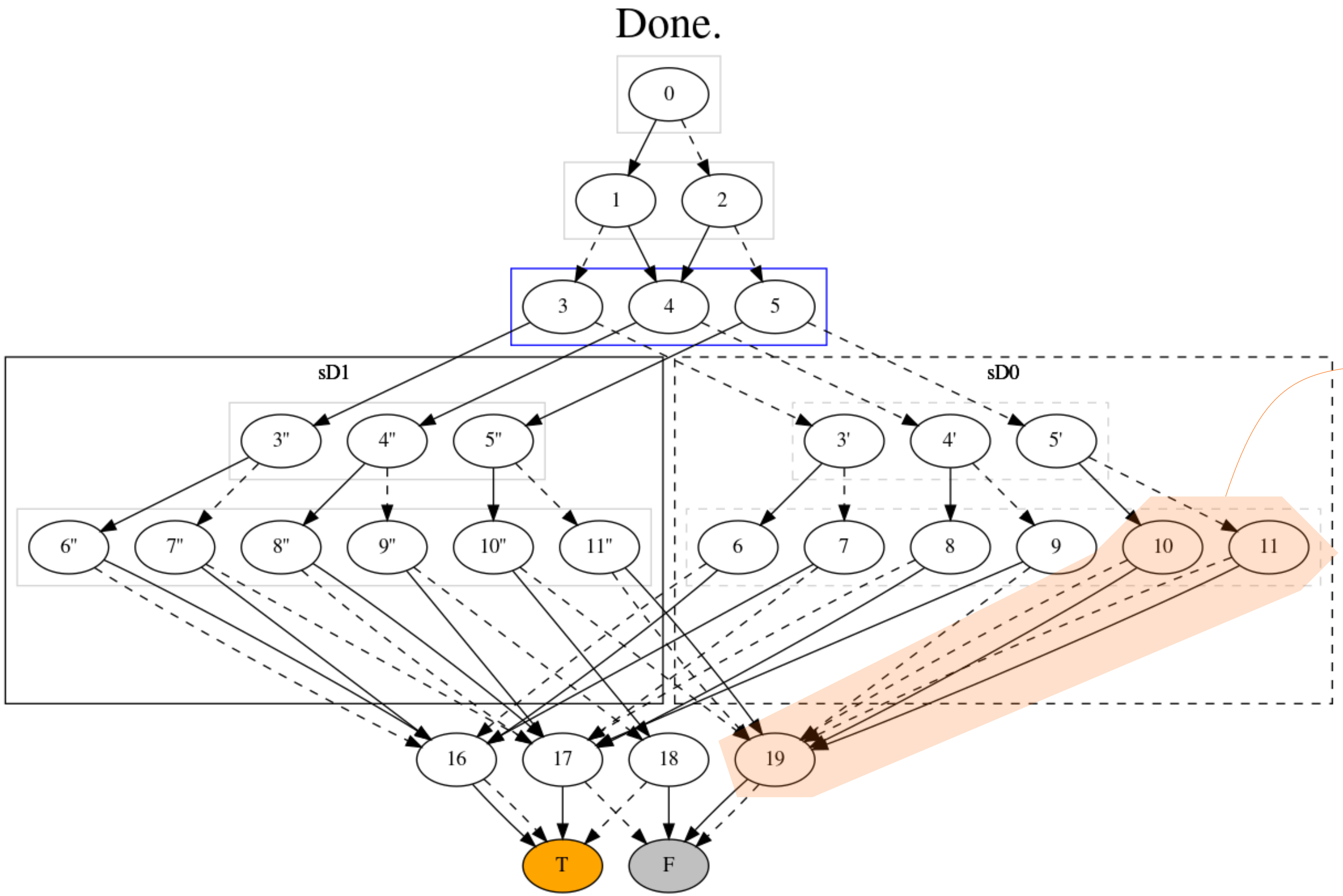


Remove old “x_4“ layer

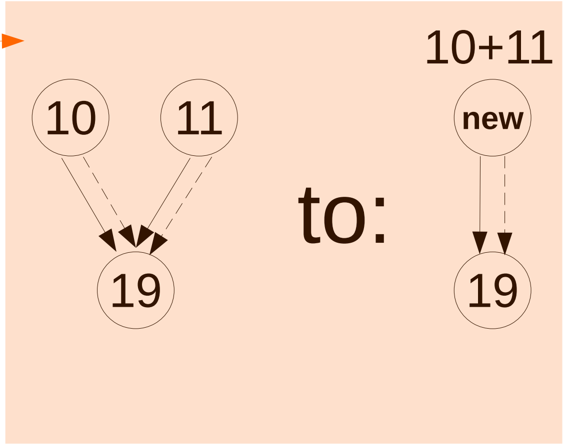


Don’t need this layer anymore.

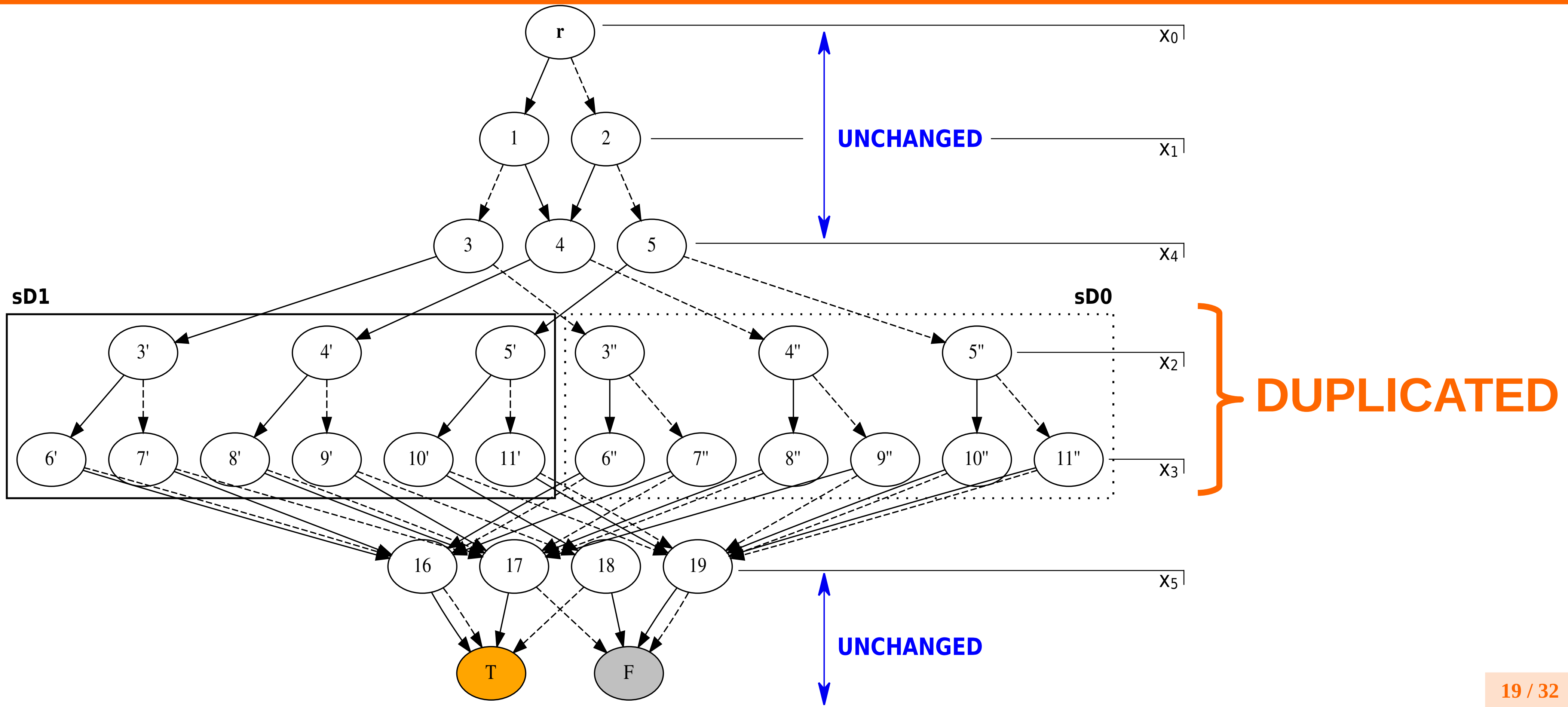
If we are lucky: remove redundancy.



Note: we can merge, e.g.:



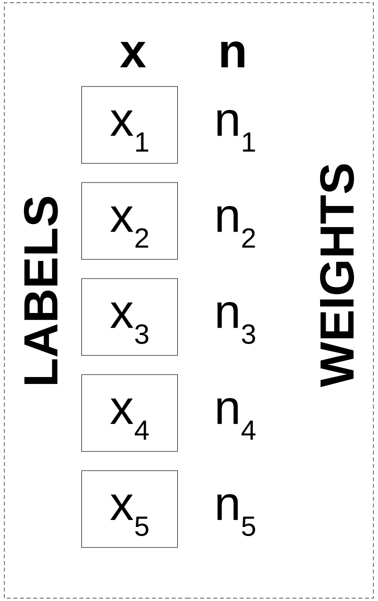
But here is the worst case.



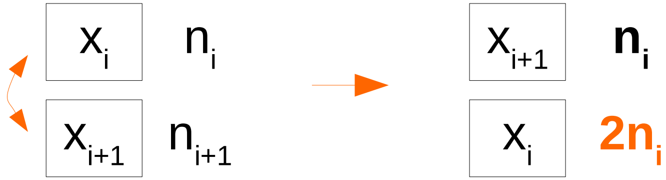
The idea: simplified model

Let's introduce **"weighted variable sequence"** – an object to keep track of **upper bounds on layer sizes** during BDD transformations

Ordered list of **labels** amended with **weights**...

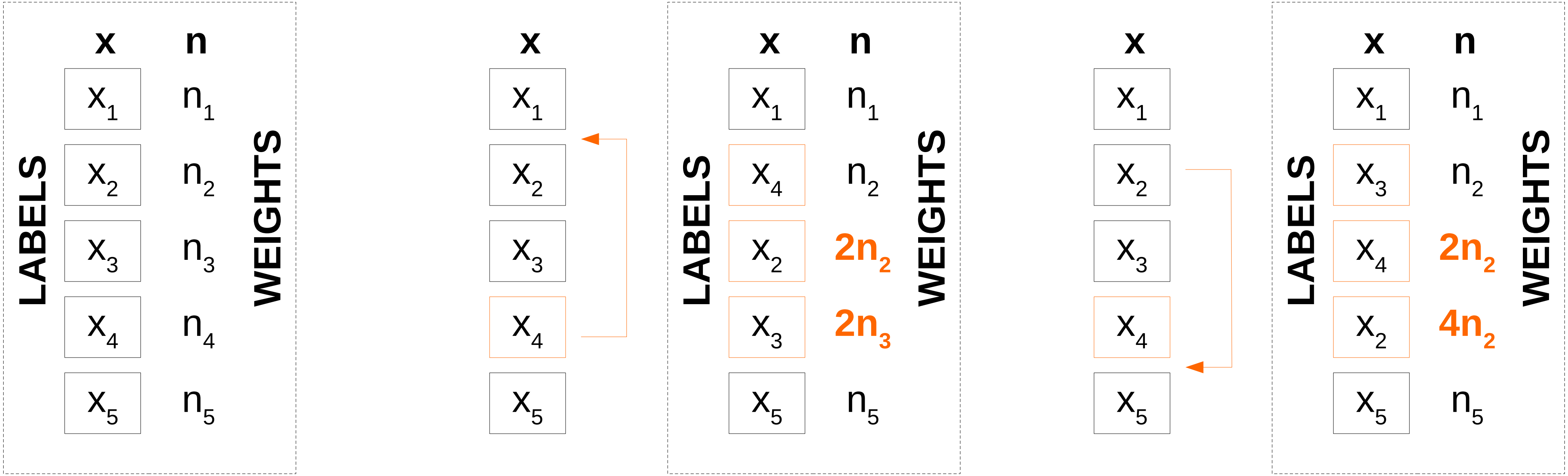


...and a "**swap**" operation



The idea: simplified model

This allows for **“sift-up”** and **“sift-down”** operations
to model sifting BDD layers up and down



(this gives an upper bound on BDD layer widths and, consequently, BDD sizes)

Plan of attack: aligning two BDDs

Initial problem: align two BDDs, *A* and *B*

$$\min_{\vec{v}}(|T^*[A,\vec{v}]| + |T^*[B,\vec{v}]|) \quad \text{(AP-BDD)}$$

Simplified problem: generate and align two varseq-s, S_A and S_B

A → S_A, B → S_B; then solve*: $\min_{\vec{v}}(|T^*[S_A,\vec{v}]| + |T^*[S_B,\vec{v}]|)$ (AP-VS)

Solve using branch-and-bound search

v^* *Solution to the simplified problem*

Solution (heuristic): transform A to $\boldsymbol{v}^*$ and B to $\boldsymbol{v}^*$

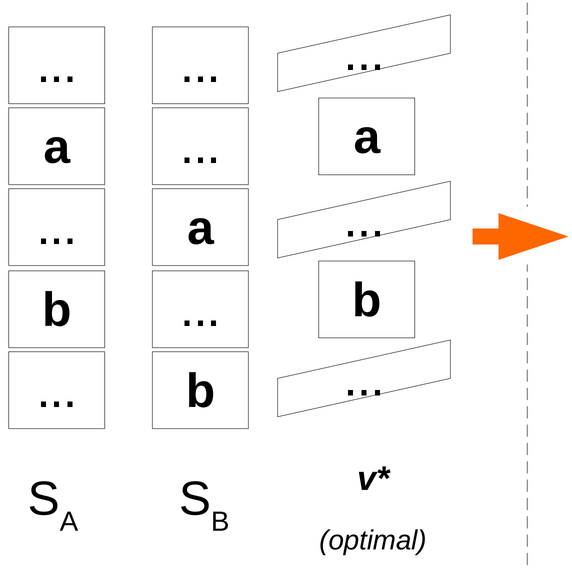
Obtained solution is an upper bound for (AP-BDD)

* here **size** of a varseq S, denoted |S| := sum of weights.

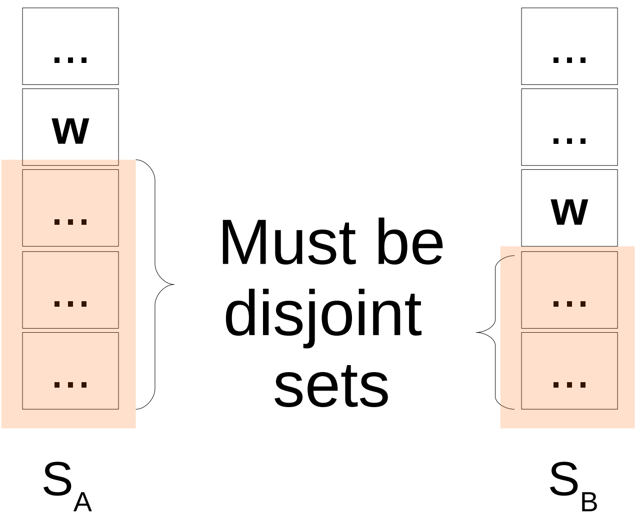
Key nice properties

relevant to the simplified problem.

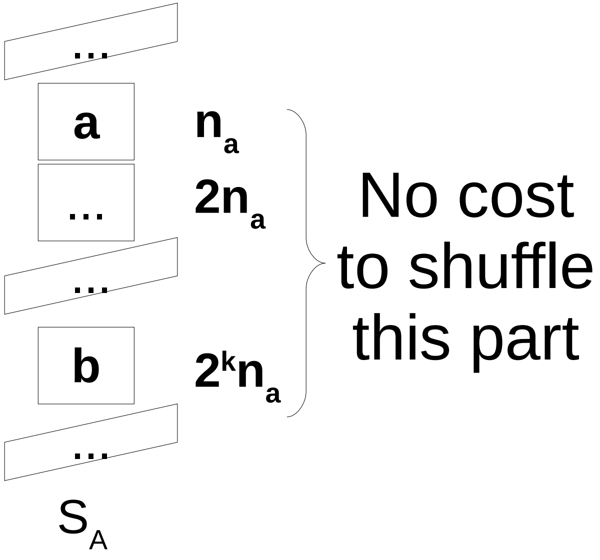
"Aligned pair"



Candidates for the last element

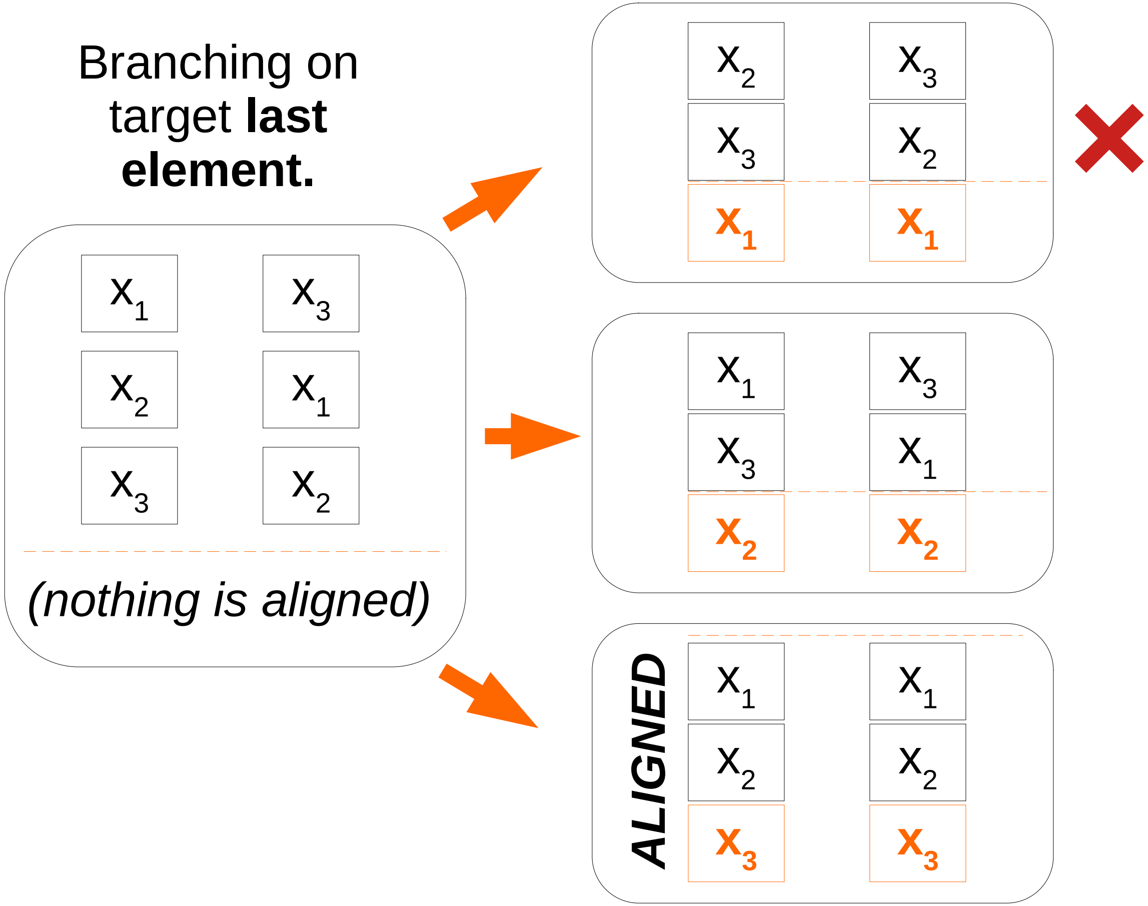


"Exp. weighted subsequence"



Simplified problem: Branch...

Branching on target **last element.**



We can safely disregard: violates “aligned pair”
($x_1 \prec x_2$ both in A and B)

One position is aligned. Further branching needed

Immediately gives an upper bound
(not necessarily optimum)

Simplified problem: … and bound.

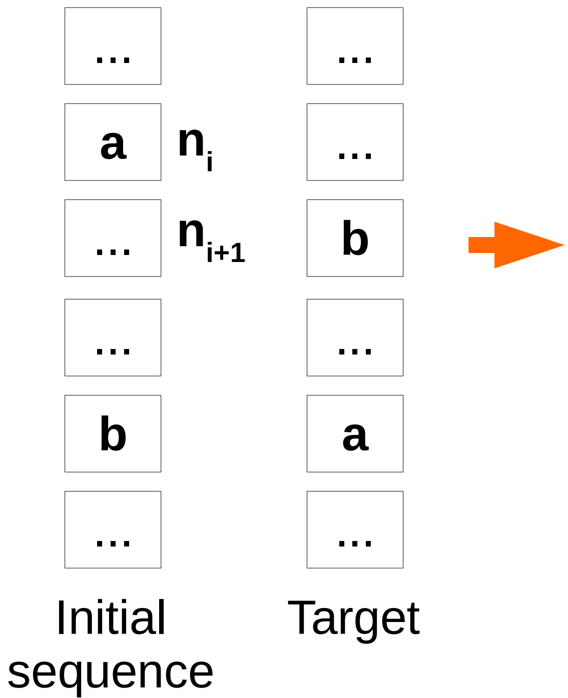
UPPER bound:

Any shared order would work:

- random order
- A
- B
- any simple rule
(“cheapest” pair of sifts)
- …

LOWER bound:

Based on the following **Lemma**:



Size increase is at least:

$$2n_i - n_{i+1}$$

Simplified problem: lower bound.

BEFORE alignment:

…		…	
a	$\mathbf{n_i}$	…	
…	$\mathbf{n_{i+1}}$	**b**	$\mathbf{m_j}$
…		…	$\mathbf{m_{j+1}}$
b		**a**	
…		…	
S_A		S_B	

Size: $|S_A|+|S_B|$

AFTER alignment:

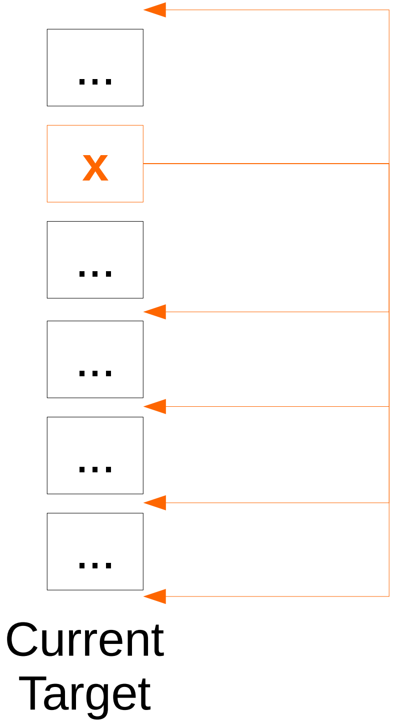
$$\textbf{Size} \geq |S_A| + |S_B| + \min(2n_i - n_{i+1}, 2m_j - m_{j+1})$$

(so we can iterate through all such {a,b} pairs to obtain a lower bound)

The baseline: Greedy BDD sifts.

1) Transform both BDDs to a starting (shared) order – best of A and B.

2) Try to improve the order – iterate through all elements, for each one: try to move it to all possible positions.



Try **every other** possible position and fix the best one.

Then repeat for the next variable.

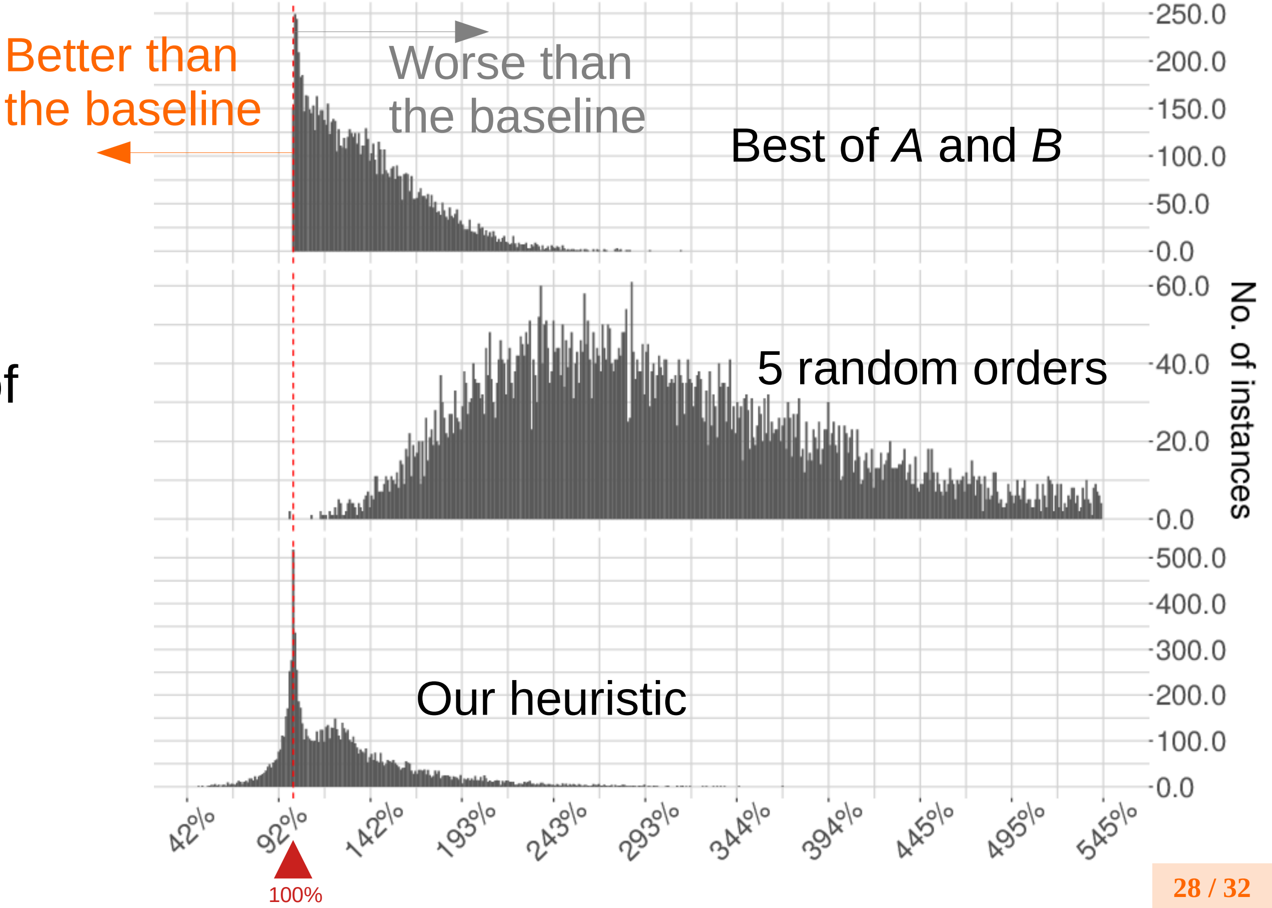
How did it work

against 10k random 15-var instances

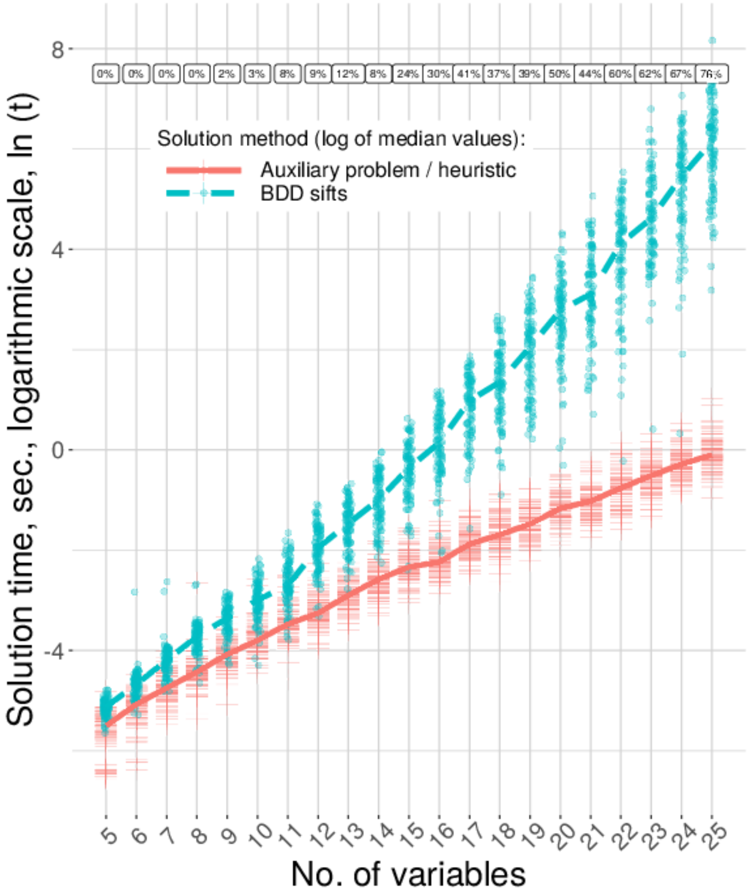
Distribution of the objective values relative to one obtained with *"greedy sifts."*

- 100% = same performance
- 50% = the method yielded half of the baseline objective

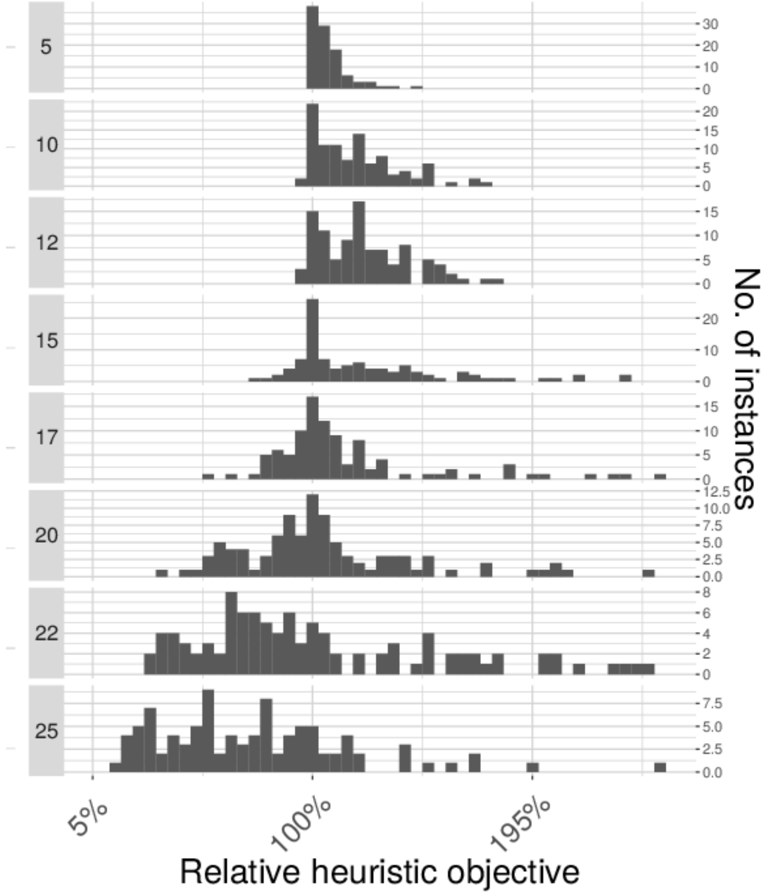
- **Our heuristic was within 30% of greedy sifts heuristic on >60% of instances.**



… and it scales like $O(N)$, not $O(2^N)$



(a)



(b)

Further research directions

- Improving the branch-and-bound approach
  (e.g., better bounds, maybe different branching principle, etc.)

- Alternative “simplifications”
  (e.g., possibility for size decrease, other encodings of BDD, etc.)

- Leverage interconnections between simplified and original problems (e.g., random starting orderings, interleaving varseq- and BDD-based subproblems, etc.)

- Of course, **applications** – in the context of Consistent Path problem or not.

Summary

- **Problem:** aligning two BDDs (enforcing the property of being "order-associated" – sharing the variable order)
- **We propose:**
  - introduce a simplified problem, based on "weighted variable sequences"
  - We then solve the simplified problem to align the constructed *variable sequences*, and use the resulting variable order as a target for the initial pair of BDDs
- It did work better than our baseline heuristic ("greedy BDD sifts"), and the advantage becomes more as the instance size grows

Alexey Bochkarev ($\leftarrow$ *me*)
Clemson University
✉ abochka@g.clemson.edu,

J. Cole Smith,
Syracuse University
✉ colesmit@syr.edu

Selected literature / bibliography

- **[Lozano2020]** Lozano, L., Bergman, D., & Smith, J. C. (2020). On the consistent path problem. Operations Research, to appear.
- **[Bergman2016]** Bergman, D., Cire, A. A., Hoeve, W.-J. van, & Hooker, J. N. (2016). Decision diagrams for optimization. Springer International Publishing. https://www.springer.com/us/book/9783319428475
- **[Knuth2009]** Knuth, D. E. (2009). Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams. In The Art of Computer Programming (1st ed., Vol. 4). Addison-Wesley Professional. https://www.informit.com/store/art-of-computer-programming-volume-4-fascicle-1-bitwise-9780321580504
- **[Scholl2001]** Scholl, C., Becker, B., & Brogle, A. (2001). The multiple variable order problem for binary decision diagrams: Theory and practical application. Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No.01EX455), 85–90.
- **[Cabodi98]** Cabodi, G., Quer, S., Meinel, C., Sack, H., Slobodová, A., & Stangier, C. (1998). Binary decision diagrams and the multiple variable order problem. International Workshop on Logic Synthesis, 346–352.
- **[Bollig96]** Bollig, B., & Wegener, I. (1996). Improving the variable ordering of OBDDs is NP-complete. IEEE Transactions on Computers, 45(9), 993–1002. https://doi.org/10.1109/12.537122
- **[Rudell93]** Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)