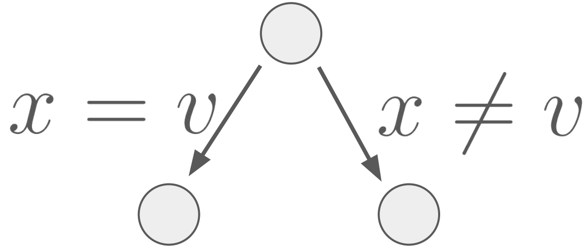
Black-Box Value Heuristics for solving Optimization problems with Constraint Programming

Augustin Delecluse, Pierre Schaus

TRUSTED AI LABS
BY DIGITALWALLONIA4.AI / SPW-RECHERCHE

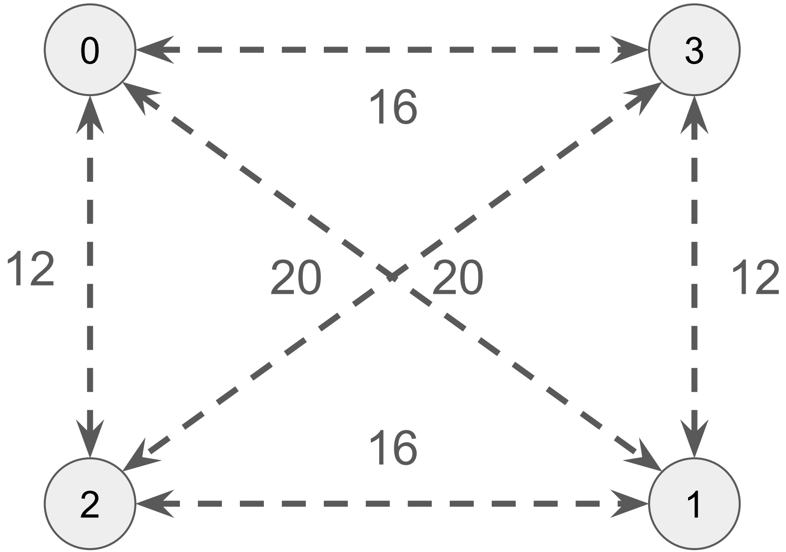
CP = Model + **Search**



- Branching commonly decomposed in 2 steps
  - Select an unfixed variable x to branch on
  - Select a value v to assign to the branching variable x
- Extensive work dedicated to variable selection based on *first-fail principle*
  - Pick variable with smallest domain
  - Pick variable involved in many failures / recent conflicts
  - Combinaison of the two (e.g. DomWDeg + Last Conflict)
- Few work dedicated on value selection
  - Picking the minimum value of the domain (MinDom) remains the popular default choice

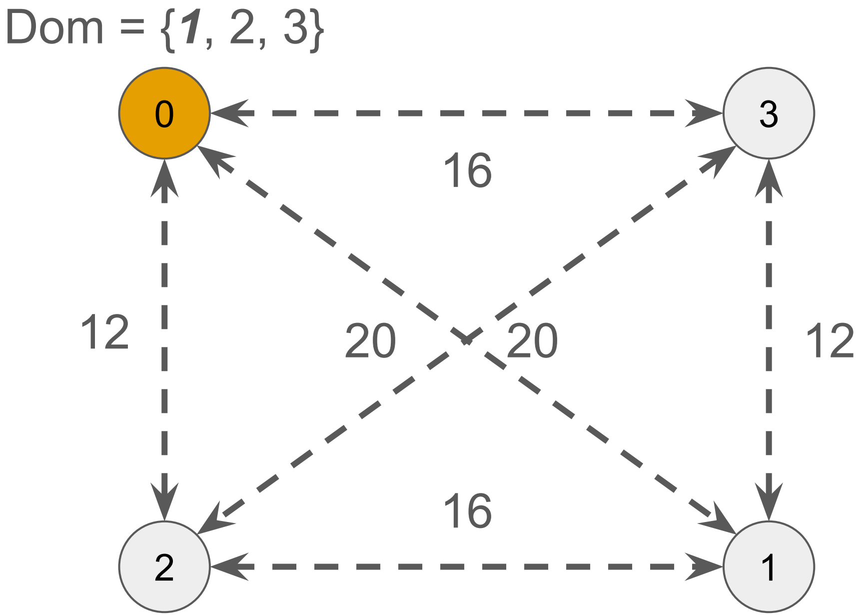
Problem with MinDom

- Has no consideration for the objective
- Can lead to bad first solution
    - Large search tree
    - Large runtime
- Example with TSP



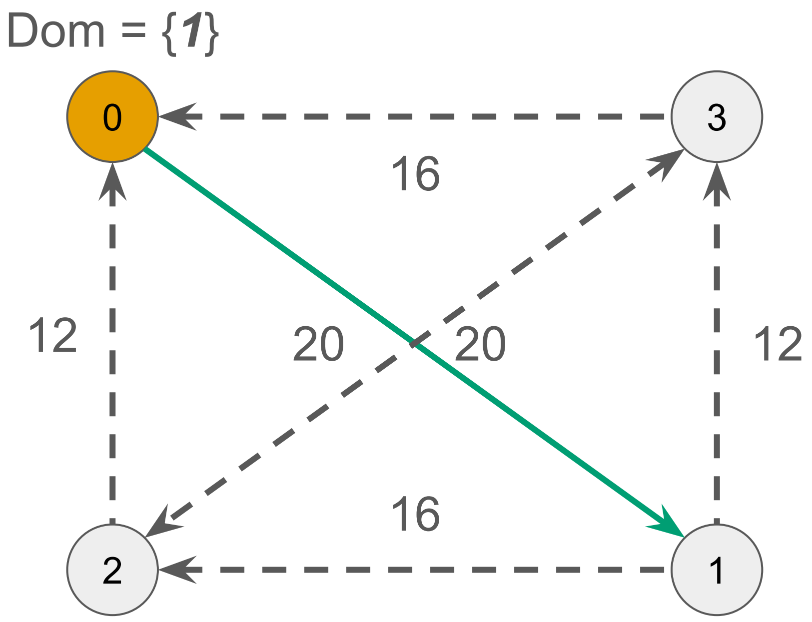
Problem with MinDom

- Has no consideration for the objective
- Can lead to bad first solution
    - Large search tree
    - Large runtime
- Example with TSP



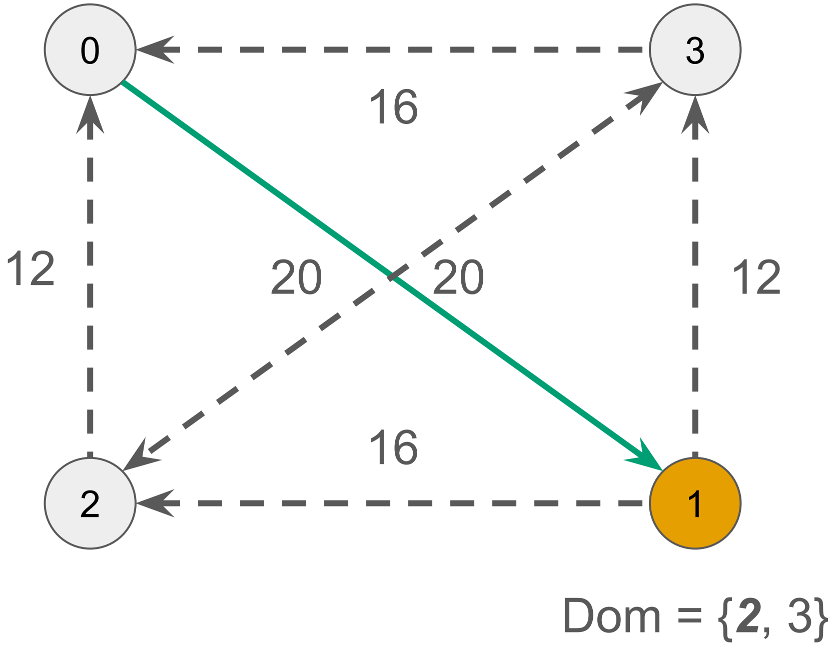
Problem with MinDom

- Has no consideration for the objective
- Can lead to bad first solution
  - Large search tree
  - Large runtime
- Example with TSP



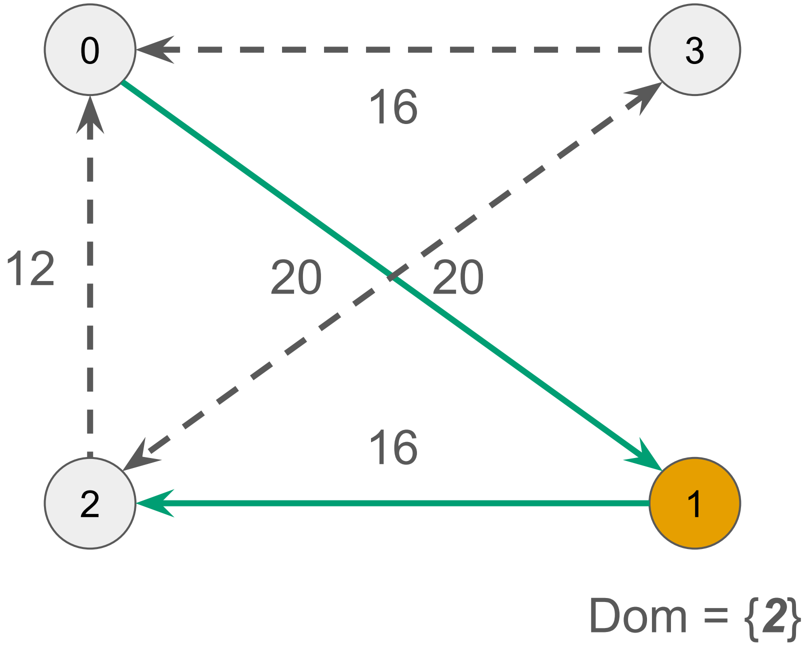
Problem with MinDom

- Has no consideration for the objective
- Can lead to bad first solution
  - Large search tree
  - Large runtime
- Example with TSP



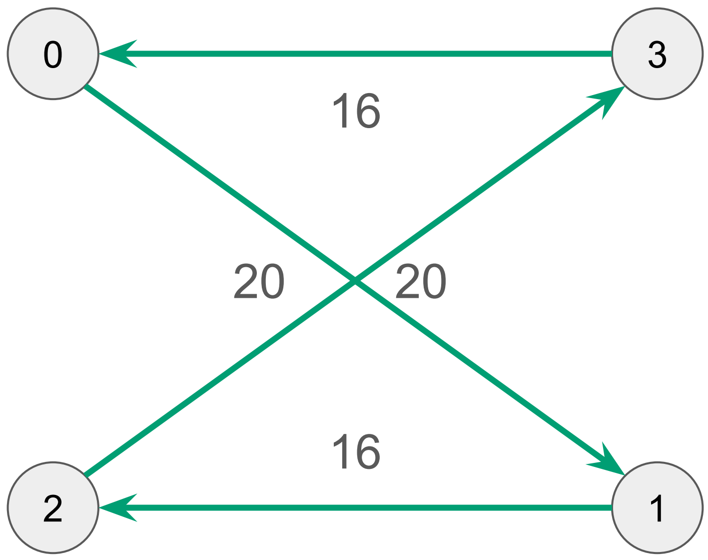
Problem with MinDom

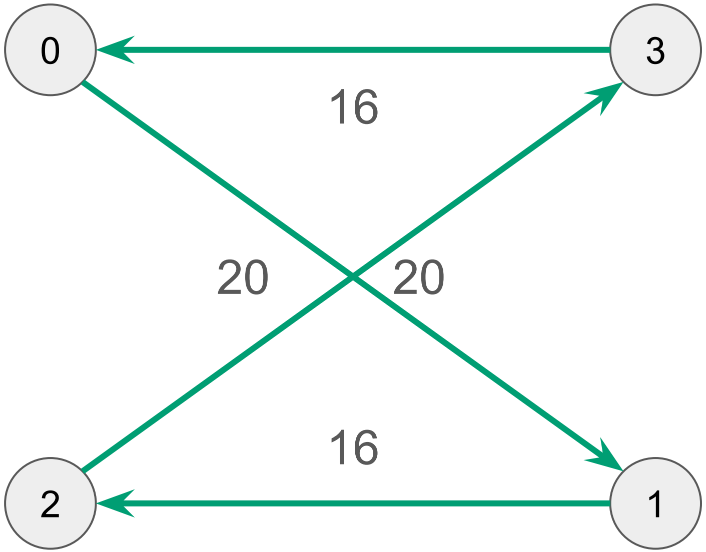
- Has no consideration for the objective
- Can lead to bad first solution
  - Large search tree
  - Large runtime
- Example with TSP



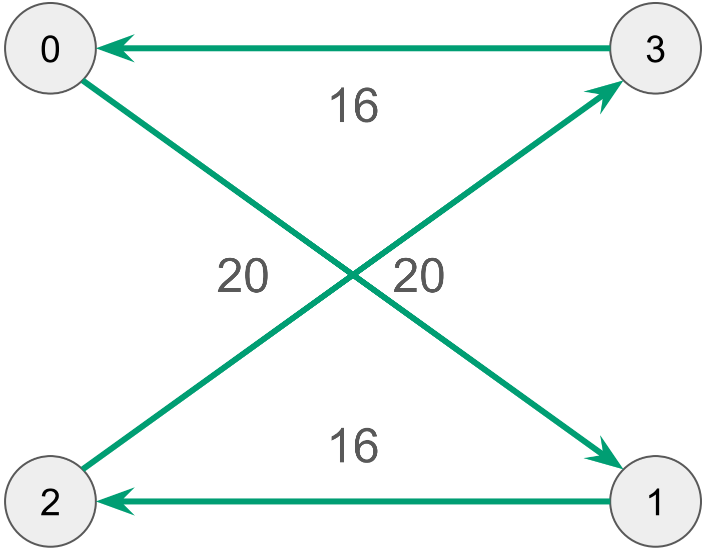
Problem with MinDom

- Has no consideration for the objective
- Can lead to bad first solution
  - Large search tree
  - Large runtime
- Example with TSP

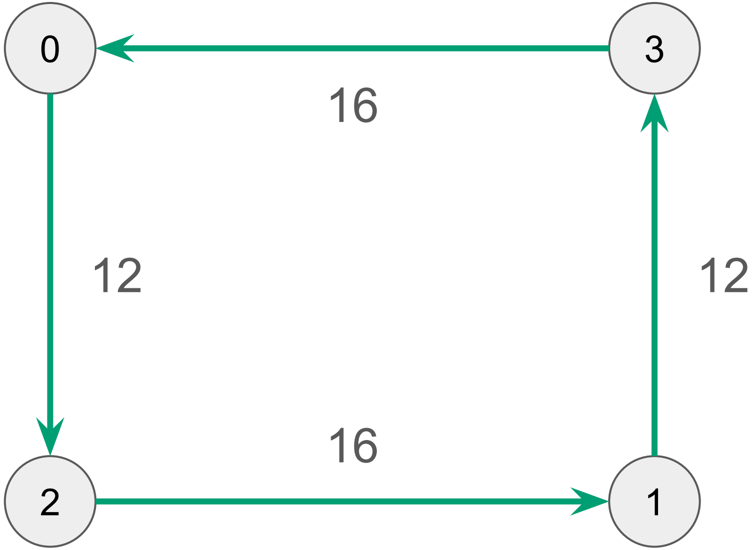




First solution cost with
MinDom: 72



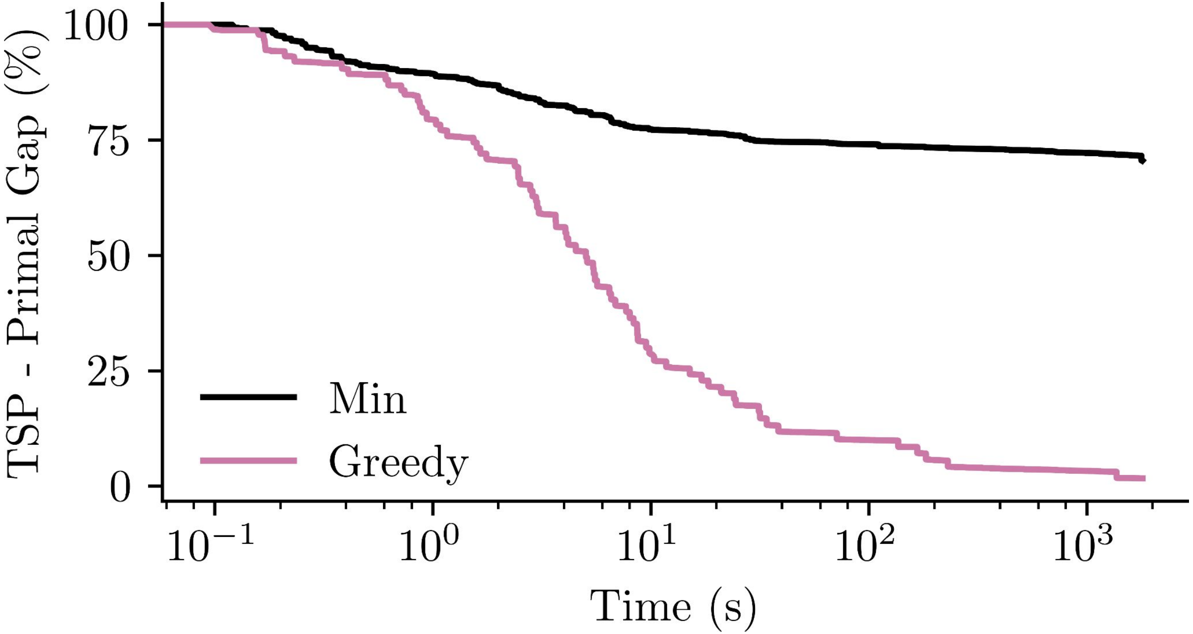
First solution cost with MinDom: 72



Best solution cost: 56

First solution found by selecting nearest neighbor!

MinDom (black-box) compared to nearest neighbor (greedy white-box)



- Instances from TSPLib
- Variable selection: DomWDeg + Last Conflict
- Average primal gap reported
- Value close to 100% : no solution
- Value close to 0% : best found solution
- The lower the better

How to mimic nearest neighbor selection in a black-box fashion?

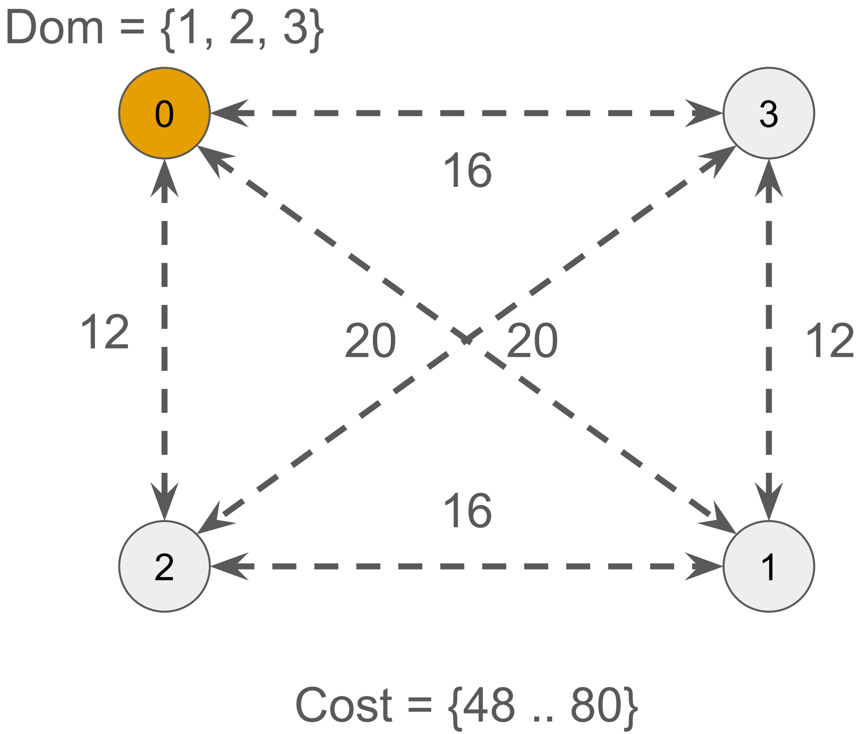
Bound-Impact Value Selection (BIVS)

- Look at every value v of the branching variable x
- What is the impact on the objective if x = v ?
- Return the value with the best impact on the objective

Fages, J. G., & Prud'Homme, C. (2017, November). Making the first solution good!. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 1073-1077). IEEE.

Bound-Impact Value Selection (BIVS)

- Look at every value v of the branching variable x
- What is the impact on the objective if x = v ?
- Return the value with the best impact on the objective

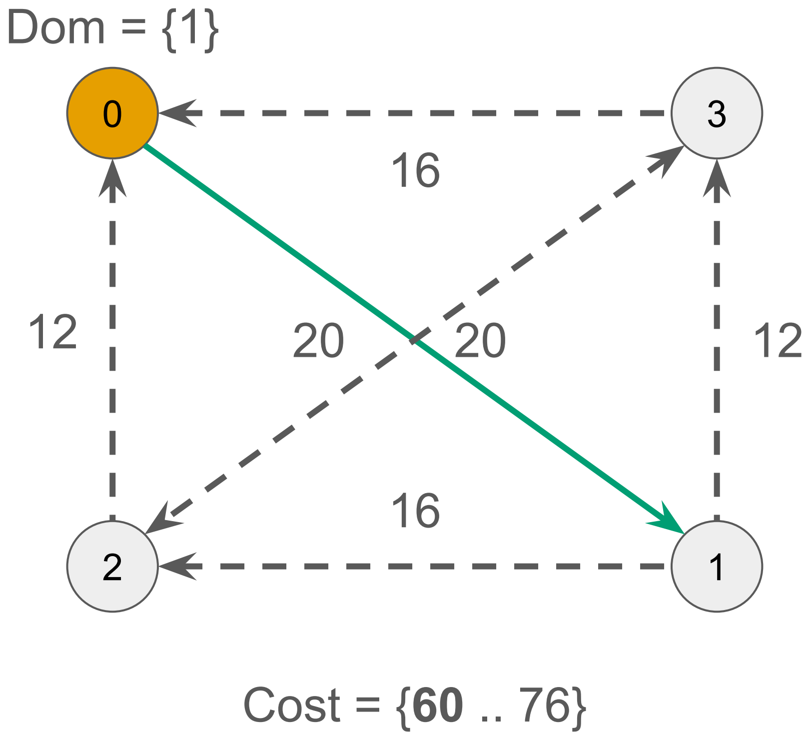


Fages, J. G., & Prud'Homme, C. (2017, November). Making the first solution good!. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)* (pp. 1073-1077). IEEE.

Bound-Impact Value Selection (BIVS)

- Look at every value v of the branching variable x
- What is the impact on the objective if x = v ?
- Return the value with the best impact on the objective

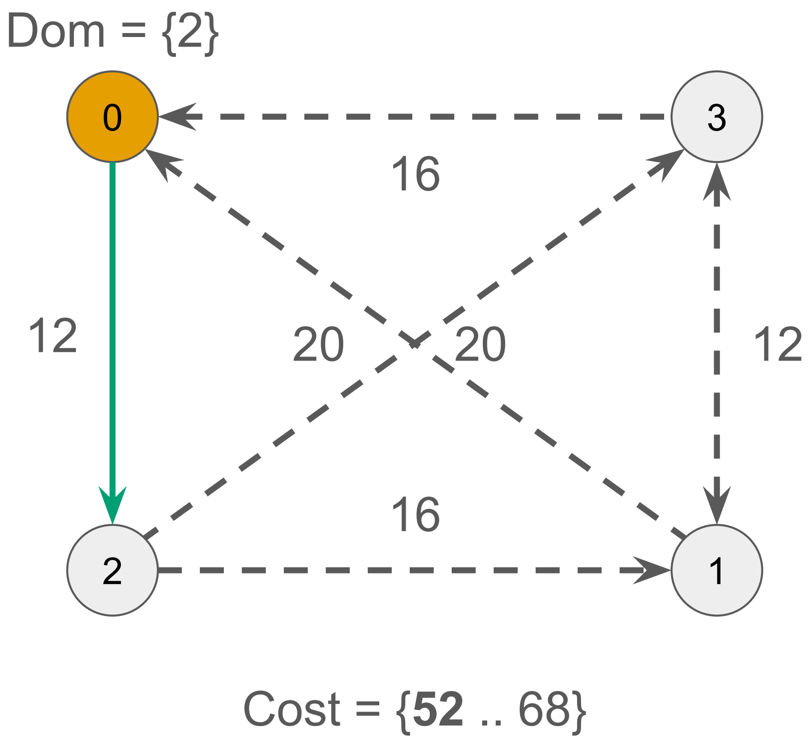
  - 1 LB = 60



Bound-Impact Value Selection (BIVS)

- Look at every value v of the branching variable x
- What is the impact on the objective if x = v ?
- Return the value with the best impact on the objective

    - 1 □ LB = 60
    - 2 □ LB = 52

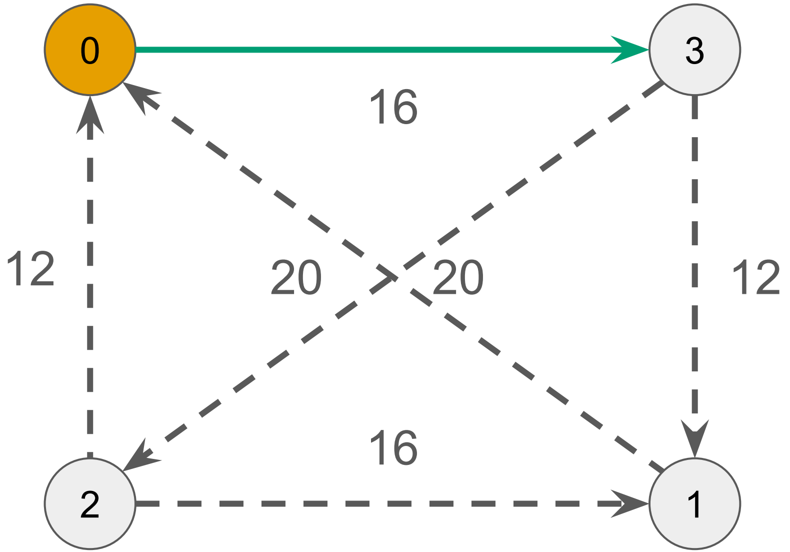


Bound-Impact Value Selection (BIVS)

- Look at every value v of the branching variable x
- What is the impact on the objective if x = v ?
- Return the value with the best impact on the objective

    - 1 □ LB = 60
    - 2 □ LB = 52
    - 3 □ LB = 56

Dom = {3}

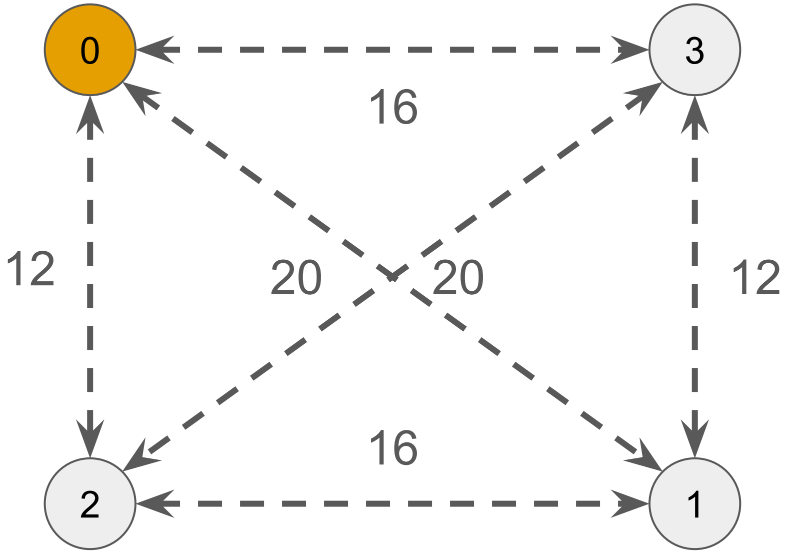


Cost = {**56** .. 72}

Bound-Impact Value Selection (BIVS)

- Look at every value v of the branching variable x
- What is the impact on the objective if x = v ?
- Return the value with the best impact on the objective

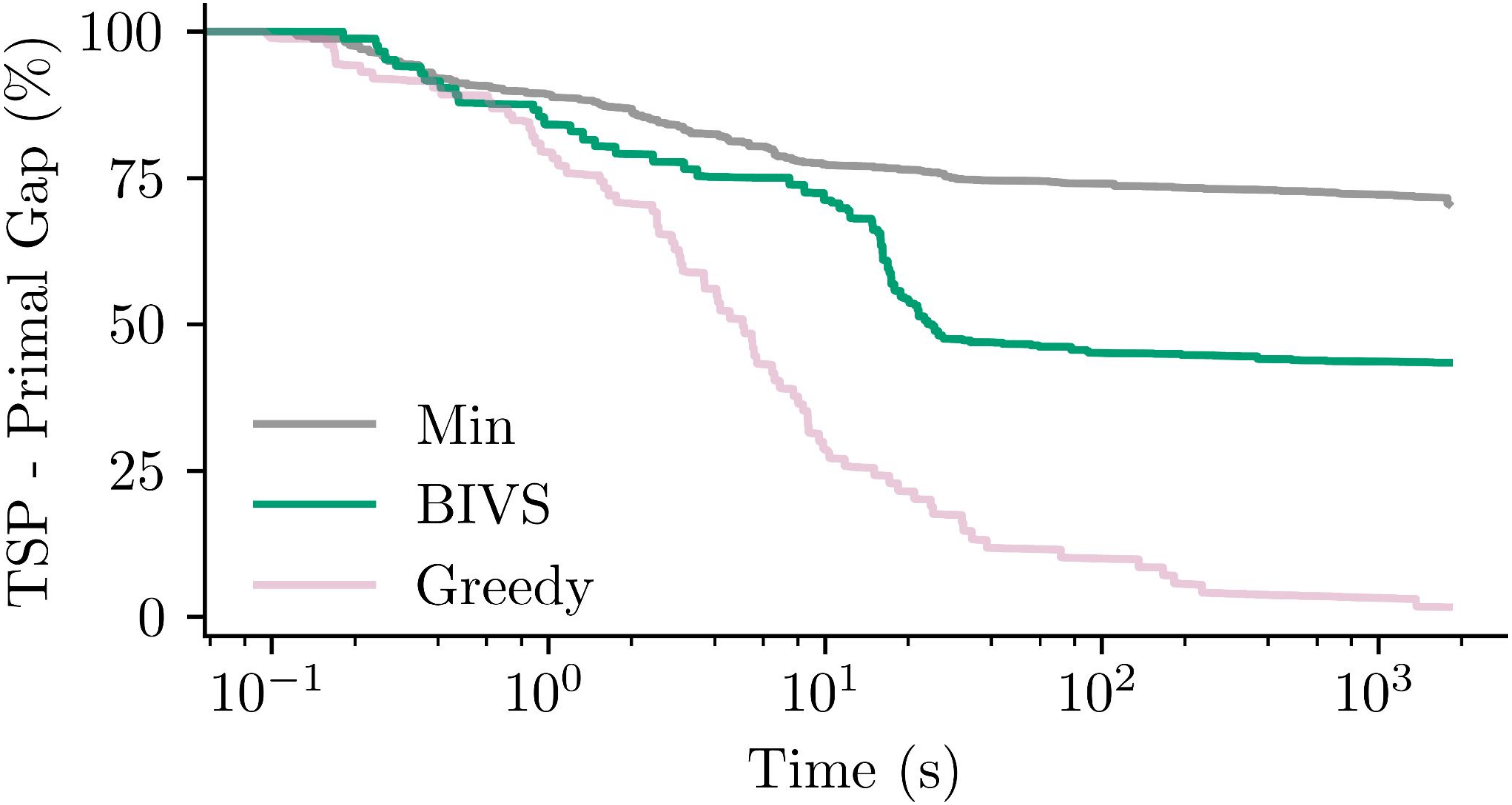
Dom = {1, 2, 3}



- 1 □ LB = 60
- **2 □ LB = 52 □ select value 2**
- 3 □ LB = 56

Cost = {48 .. 80}

BIVS in practice



Problem with BIVS: its cost

$$\mathcal{O}\left(|dom(x)| \cdot \mathcal{F}\right)$$

Size of domain

Cost of Fixpoint

- Time consuming
- Author's advice:
  - Use it for domain sizes <= 100
  - If domain size too large: consider only the bounds of the domain
- Not suited for full exhaustive search on large problems

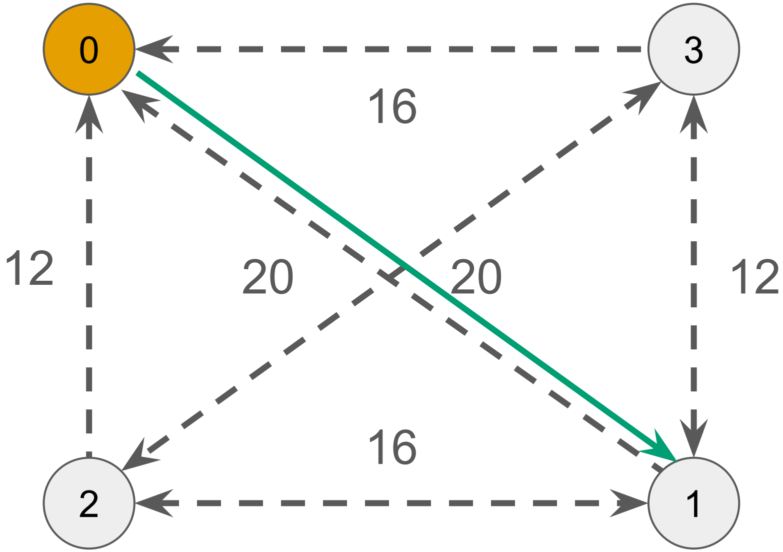
Let’s try to reduce the cost (part 1)

$$\mathcal{O}\left(|dom(x)| \cdot \mathcal{F}\right)$$

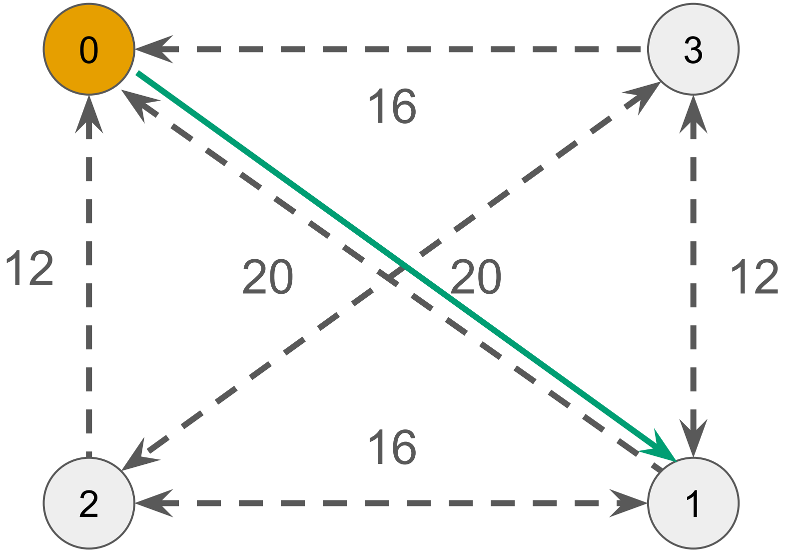
Size of domain

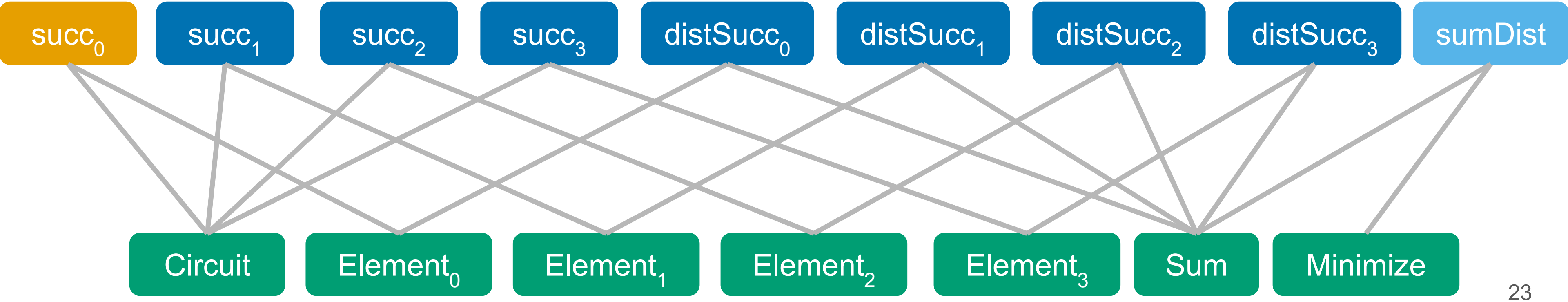
Cost of Fixpoint

Do we really need a fixpoint computation?

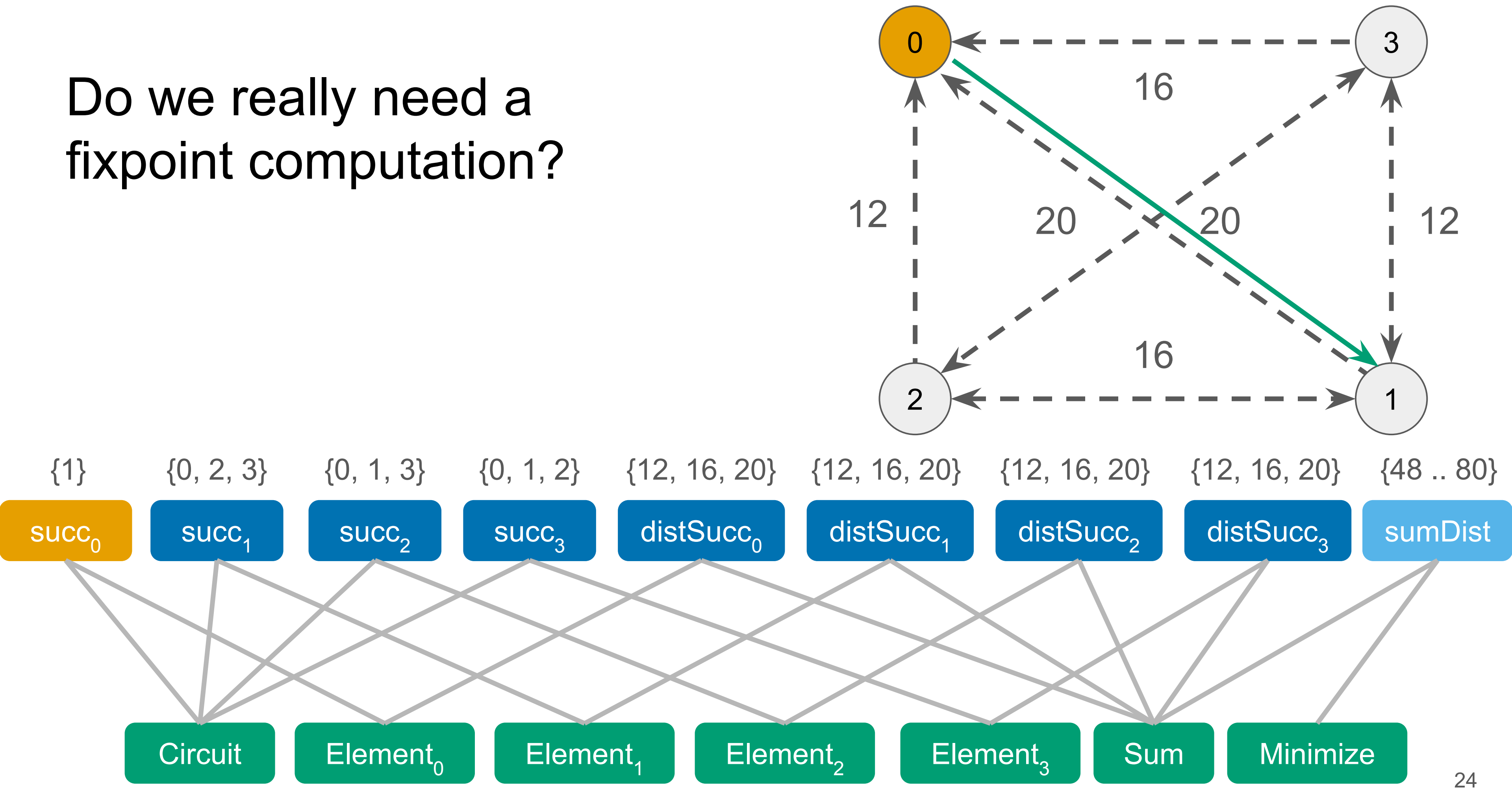


Do we really need a fixpoint computation?

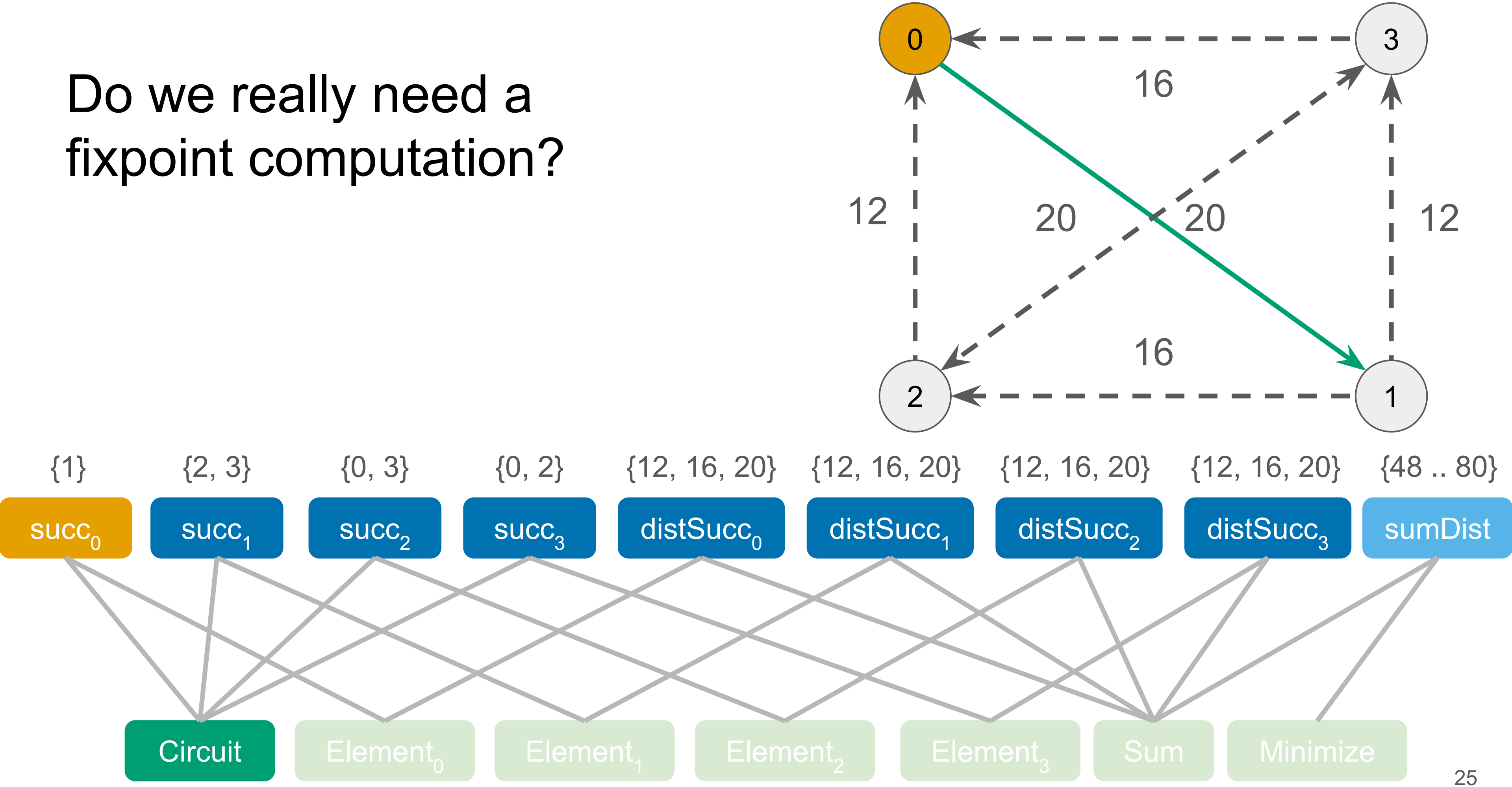




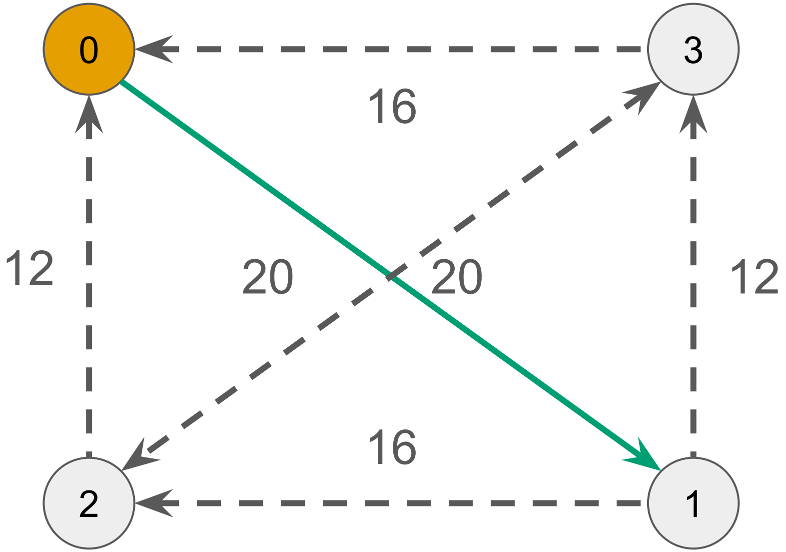
Do we really need a fixpoint computation?

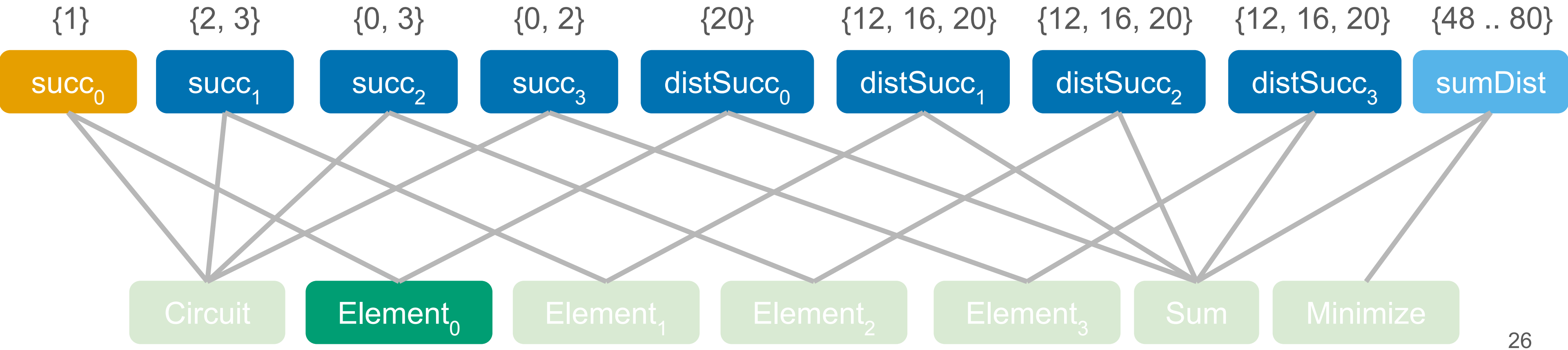


Do we really need a fixpoint computation?

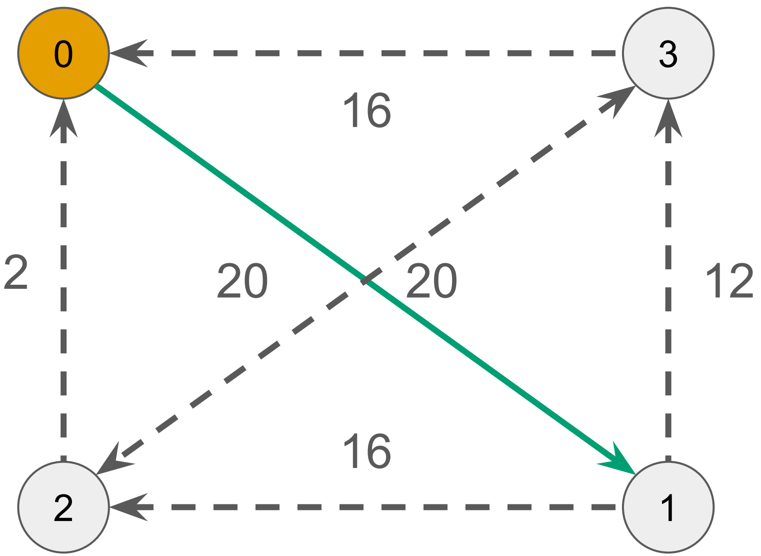


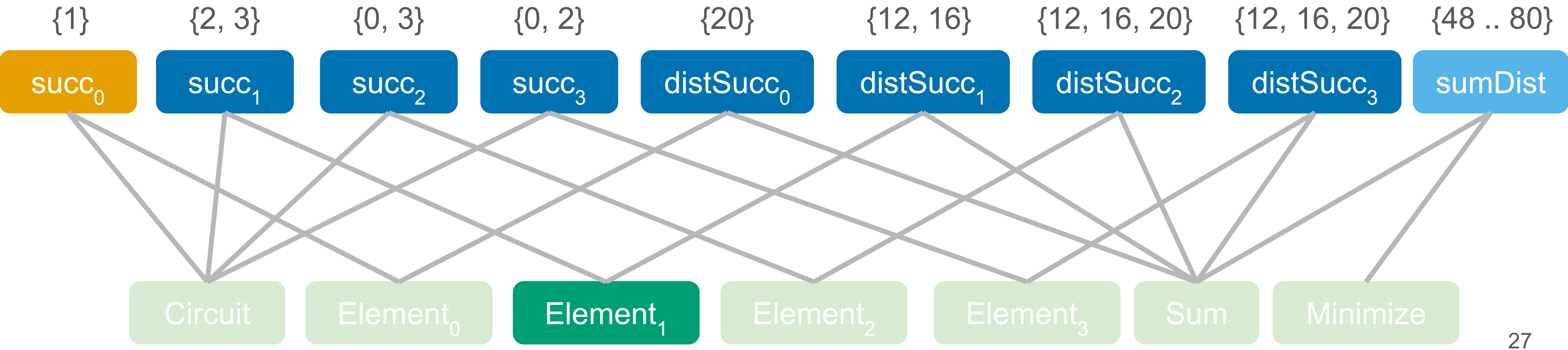
Do we really need a fixpoint computation?



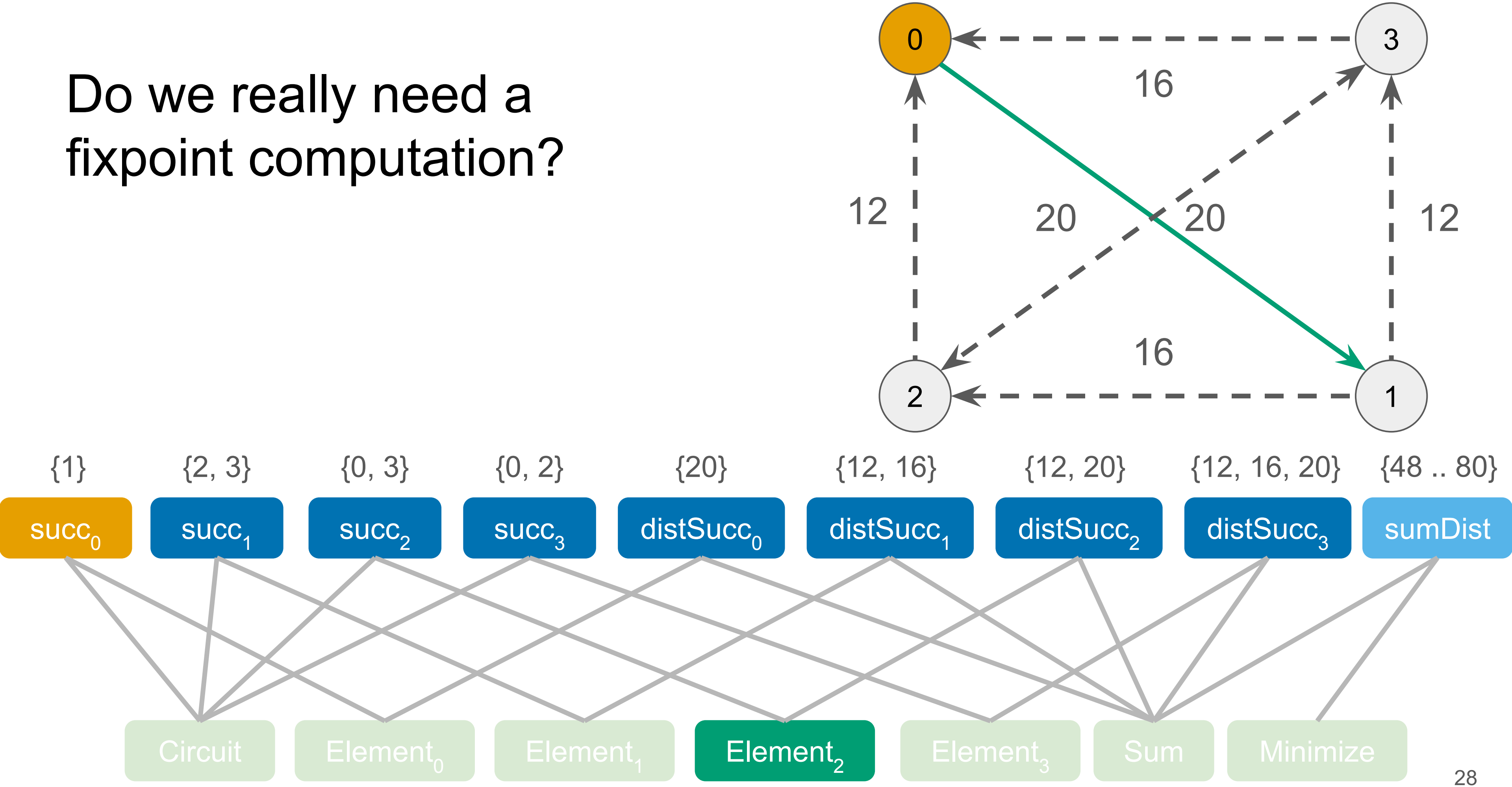


Do we really need a fixpoint computation?

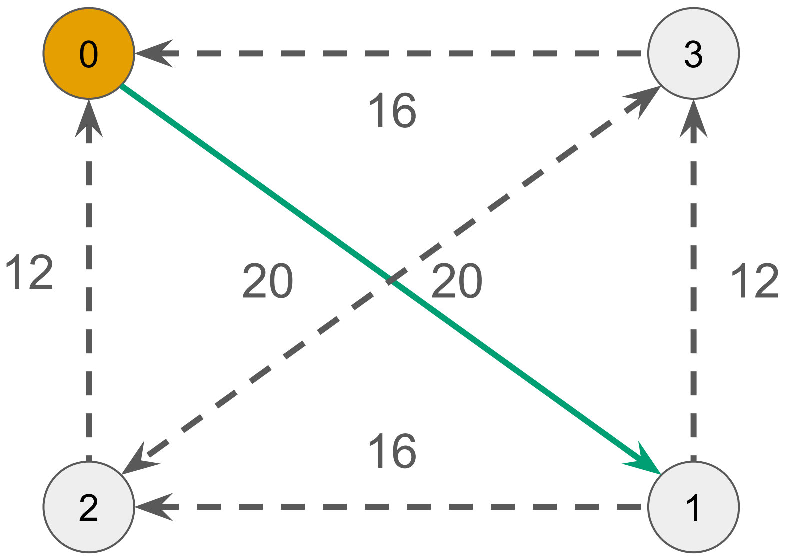


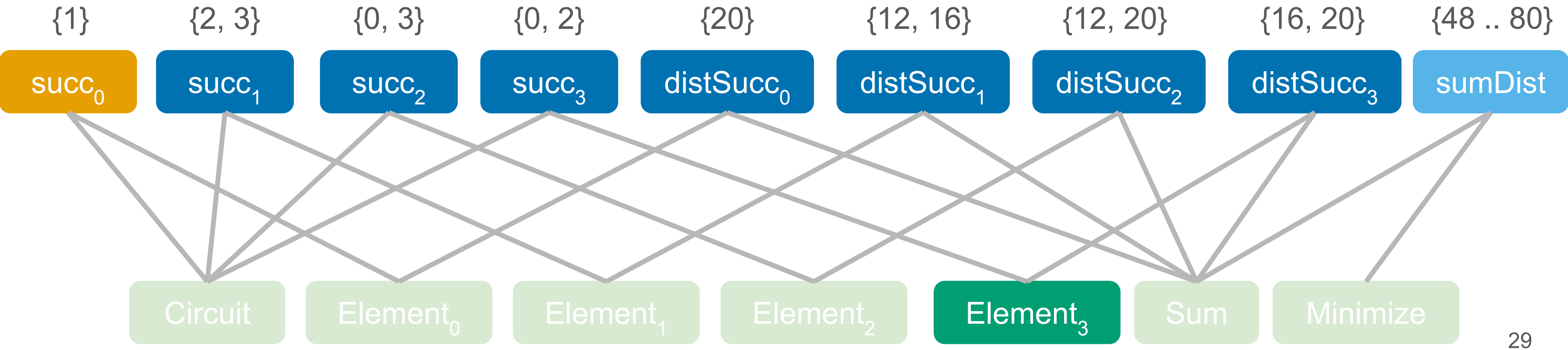


Do we really need a fixpoint computation?

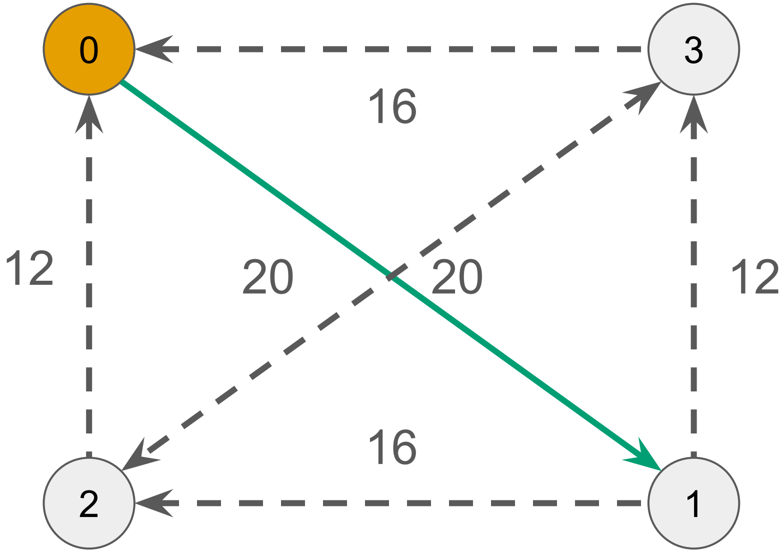


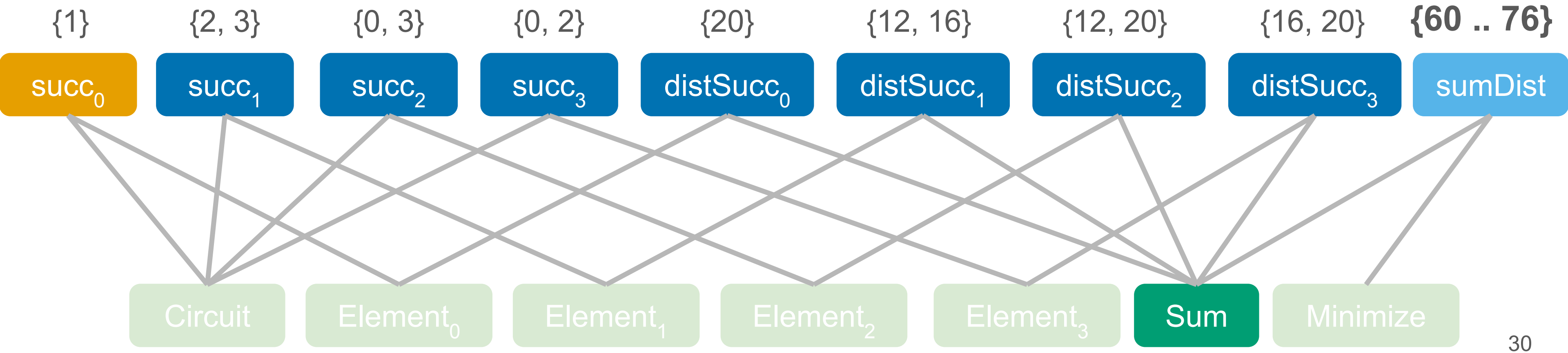
Do we really need a fixpoint computation?





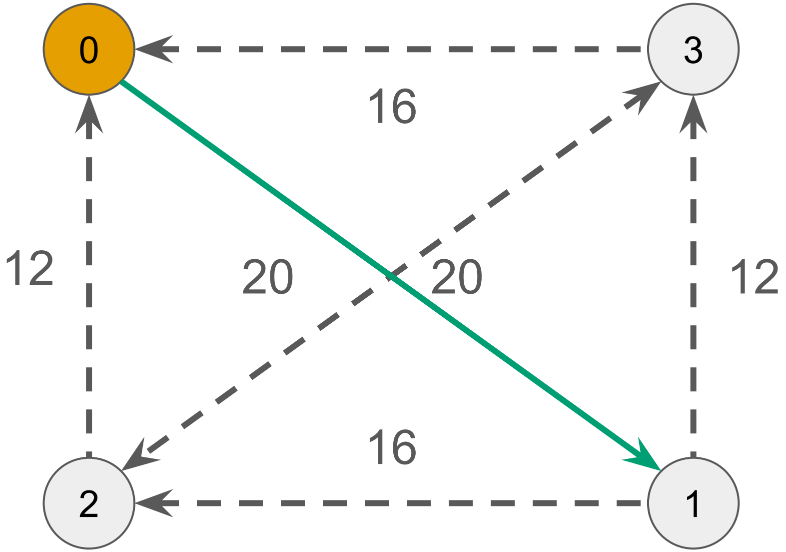
Do we really need a fixpoint computation?

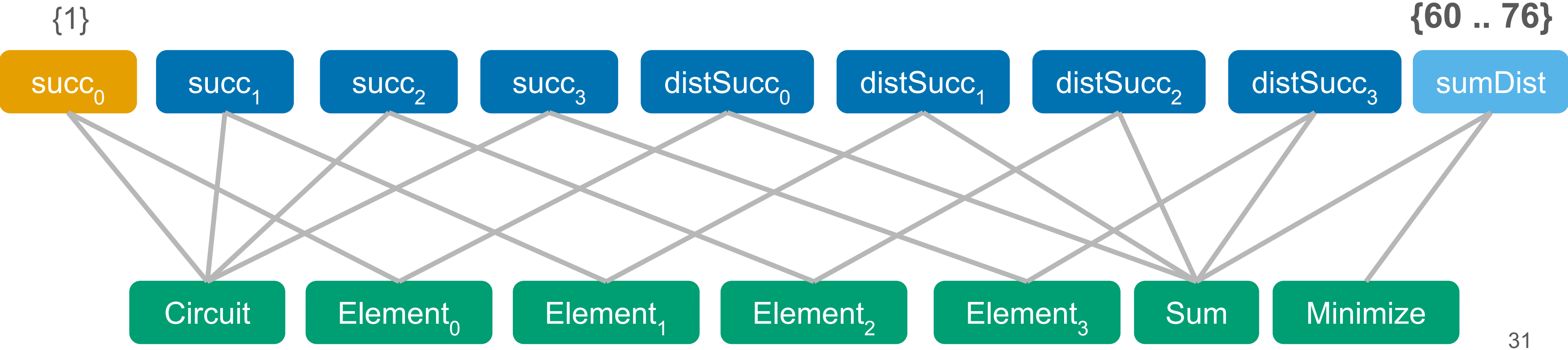




Do we really need a fixpoint computation?

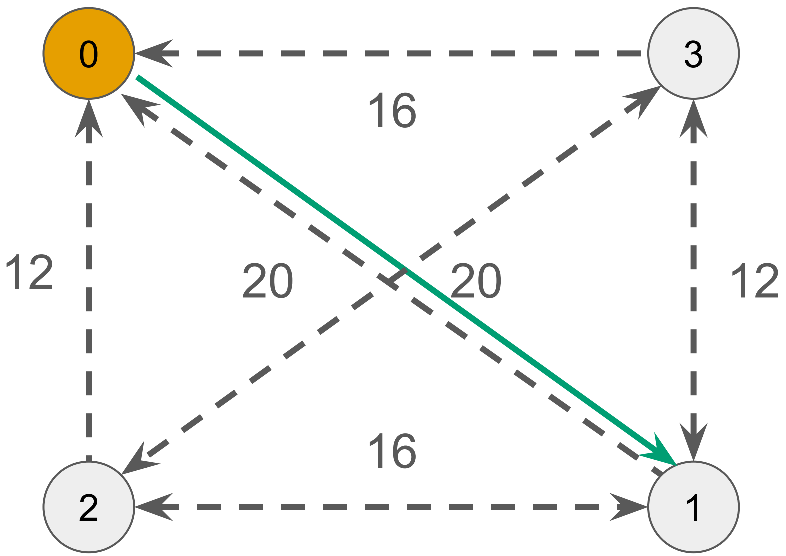
- Very little contributions from many constraints
- Let’s skip some of them

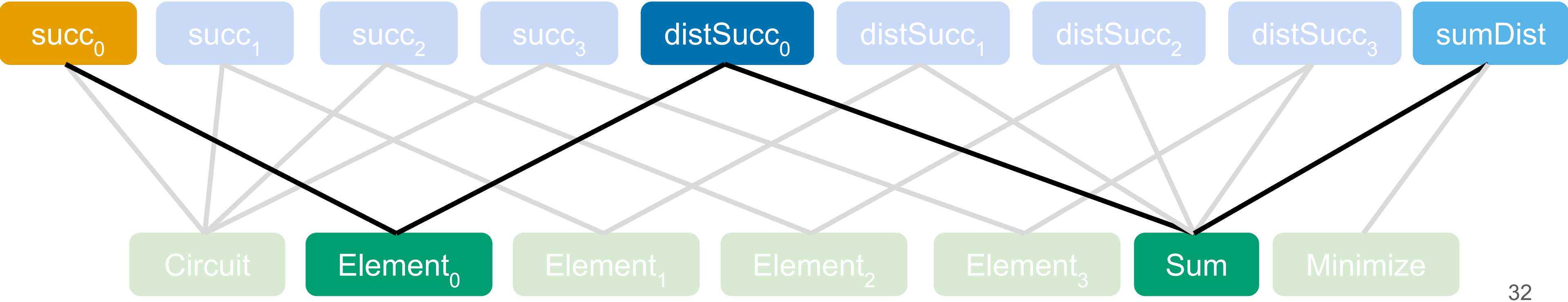




Restricted Fixpoint (RF)

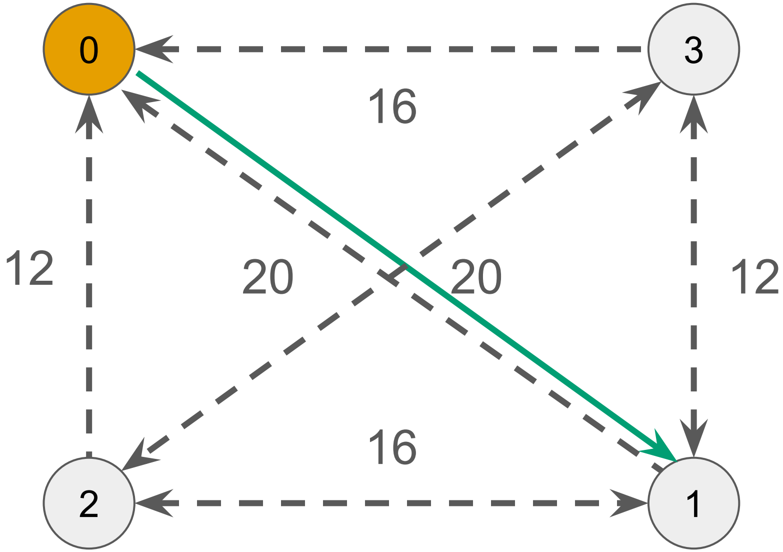
- Consider only the constraints on the shortest path of the constraint network

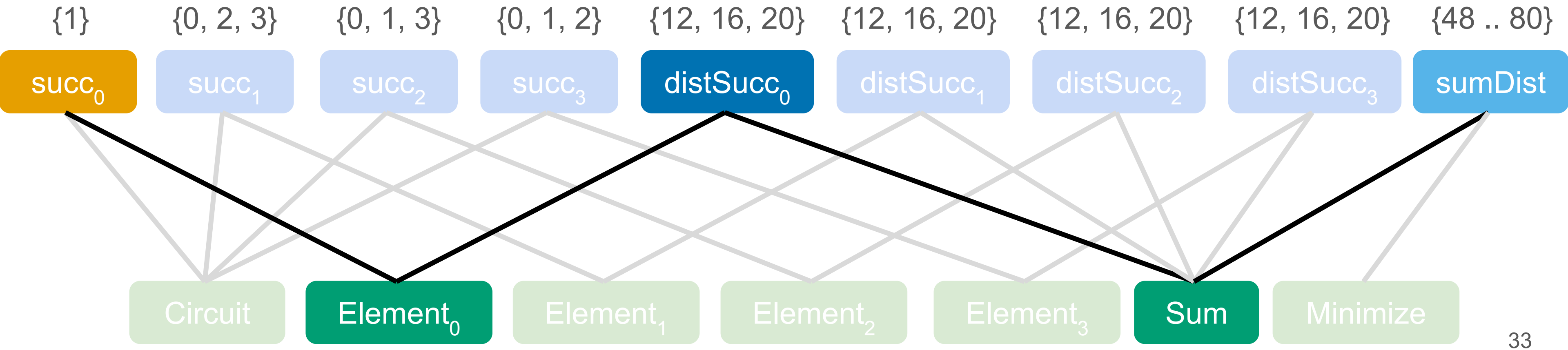




Restricted Fixpoint (RF)

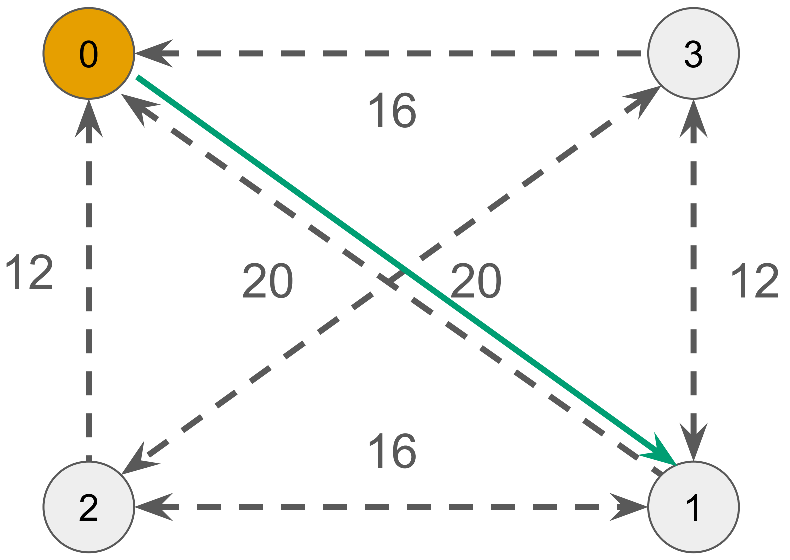
- Consider only the constraints on the shortest path of the constraint network

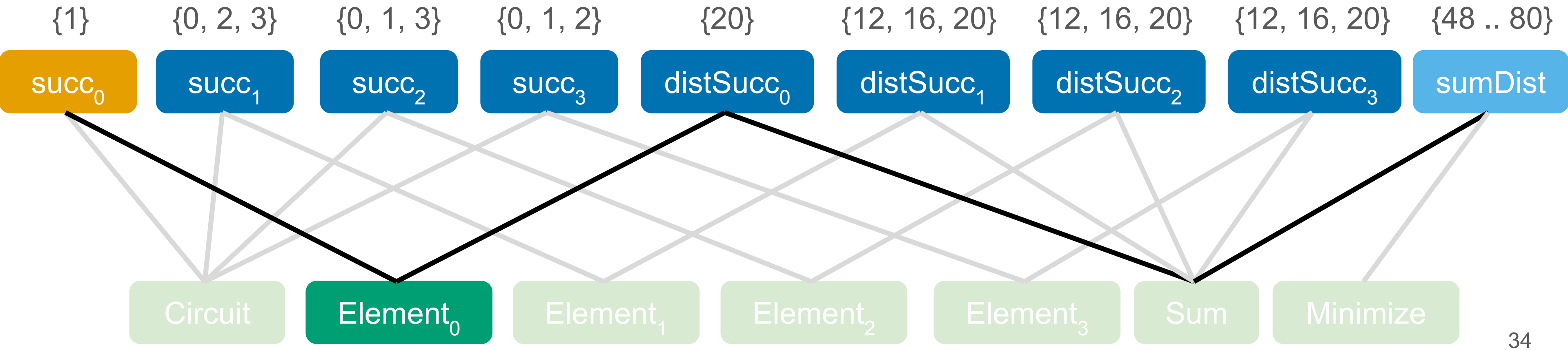




Restricted Fixpoint (RF)

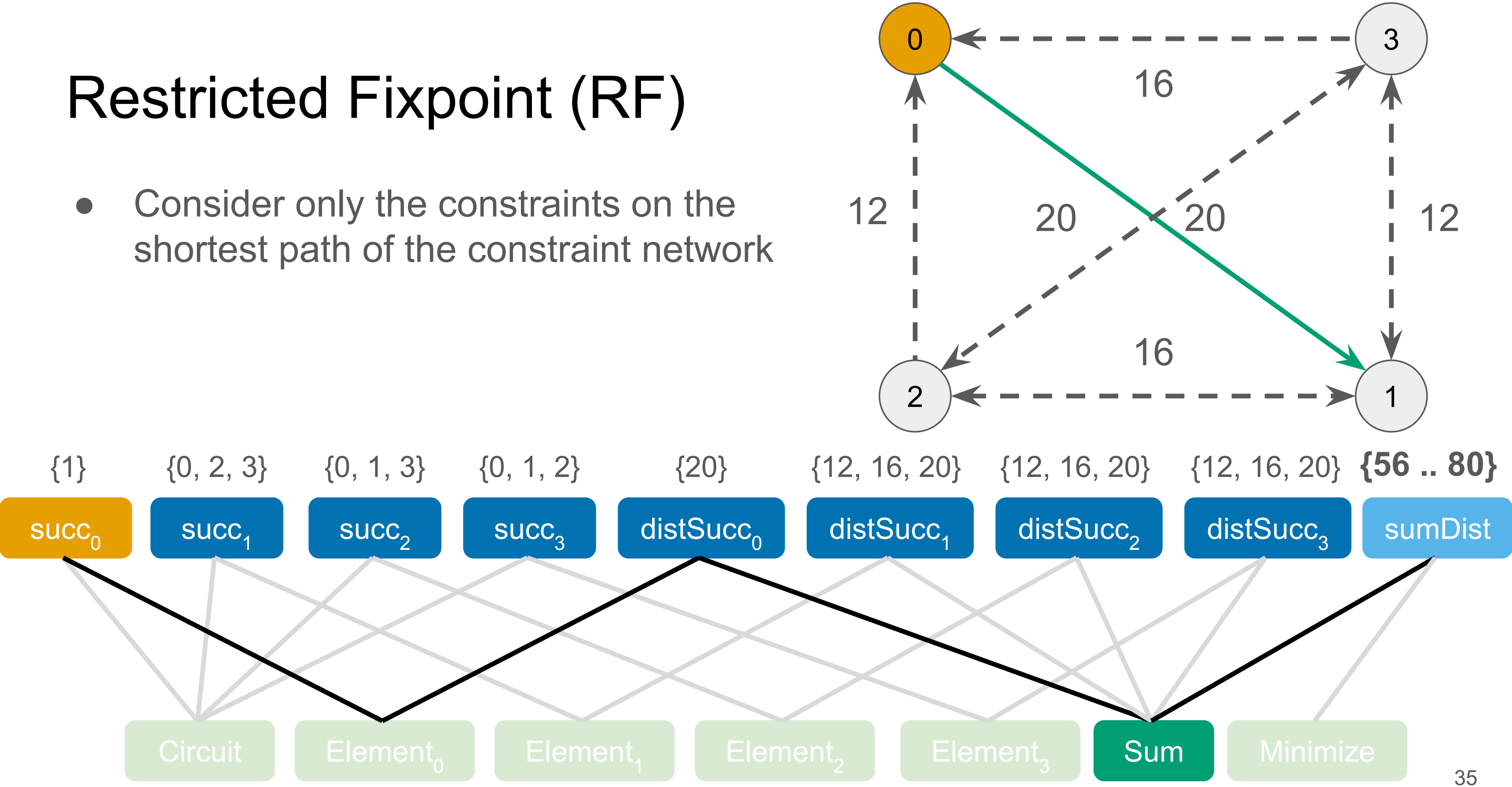
- Consider only the constraints on the shortest path of the constraint network





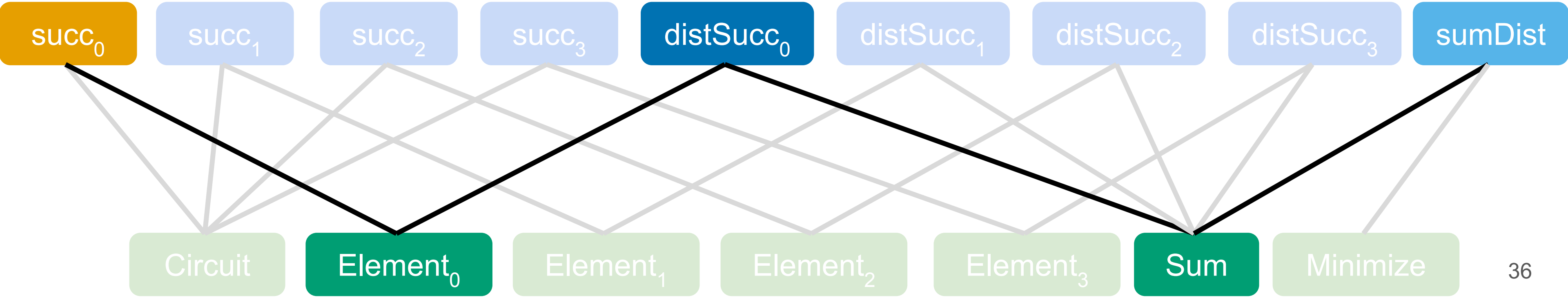
Restricted Fixpoint (RF)

- Consider only the constraints on the shortest path of the constraint network

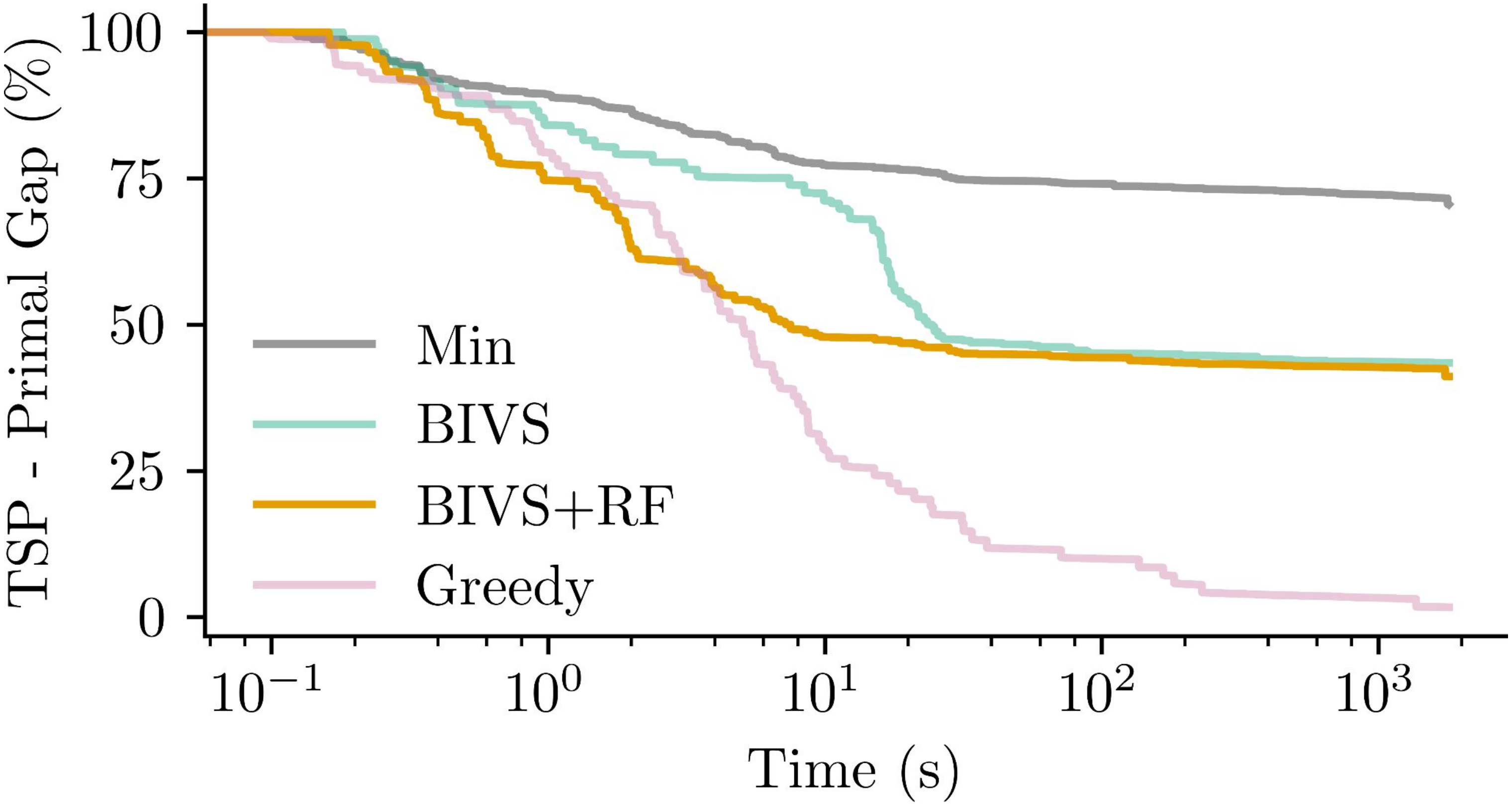


Some considerations about Restricted Fixpoint (RF)

- Less informed than a regular fixpoint
  - BIVS with and without RF might not select the same value
- But much faster
  - On a TSP, only 2 constraints are considered, no matter the TSP size
- Can be implemented by deactivating temporarily constraints out of interest
- Cheap to compute: shortests paths are precomputed before the search



Applying RF on BIVS



Let’s try to reduce the cost (part 2)

$$\mathcal{O}\left(|dom(x)| \cdot \mathcal{F}\right)$$

Size of domain

Cost of Fixpoint

Bound-Impact Value Selection (BIVS)

- Look at every value v of the **branching variable** x
- What is the **impact on the objective** if x = v ?
- Return the value with the best impact on the objective
- ***Start from variable, look at objective***

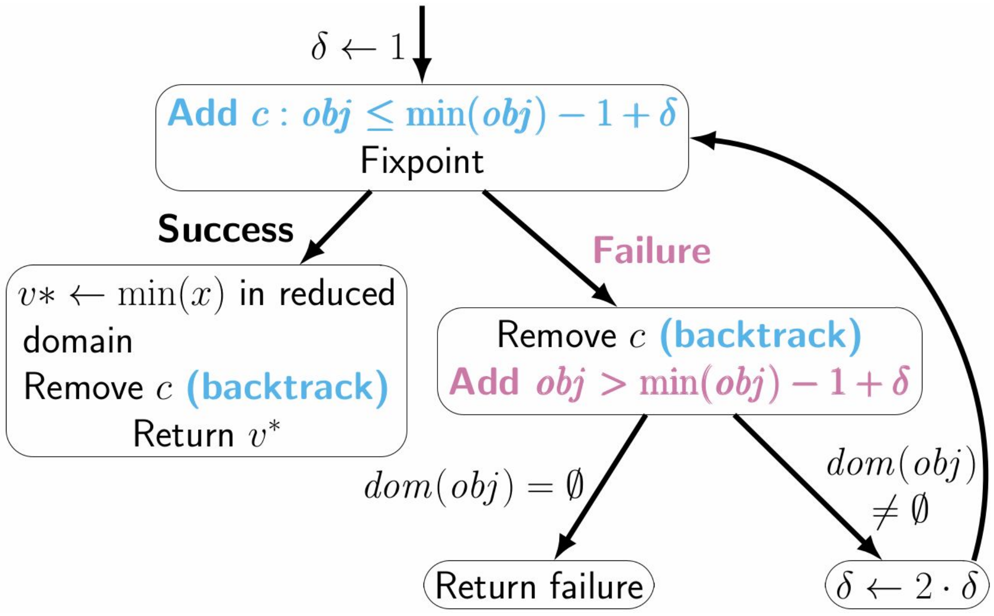
Bound-Impact Value Selection (BIVS)

- Look at every value v of the **branching variable** x
- What is the **impact on the objective** if x = v ?
- Return the value with the best impact on the objective
- ***Start from variable, look at objective***

Reverse Look-Ahead

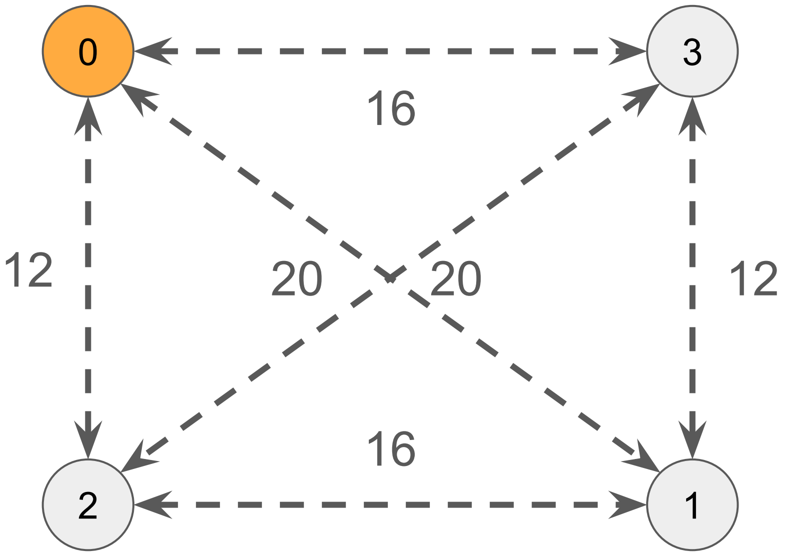
- Look at values v of the **objective**
- What is the **impact on the branching variable** if objective shrink to interval obj <= v ?
- Return value found when objective shrink to best interval
- ***Start from objective, look at variable***

Reverse Look-Ahead



- Look at values v of the **objective**
- What is the **impact on the branching variable** if objective shrink to interval obj <= v ?
- Return value found when objective shrink to best interval
- ***Start from objective, look at variable***

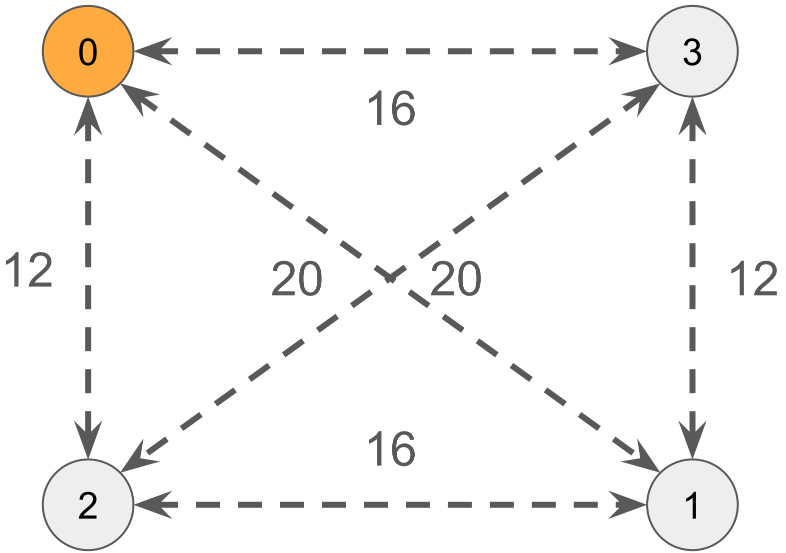
Reverse Look-Ahead



Cost = {48 .. 80}

48	49	50	51	52	53	54	55	56	57	58	..

Reverse Look-Ahead



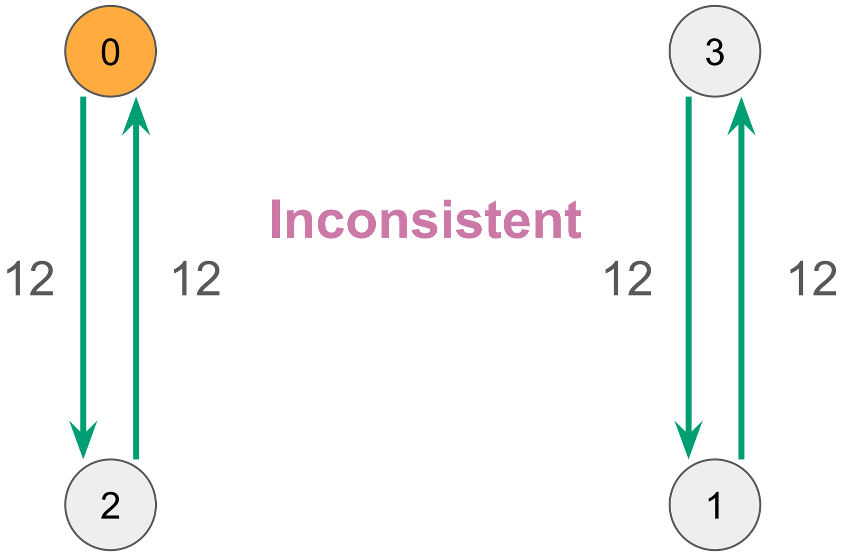
Cost = {48}

48	49	50	51	52	53	54	55	56	57	58	..

48

$$\delta = 1$$

Reverse Look-Ahead



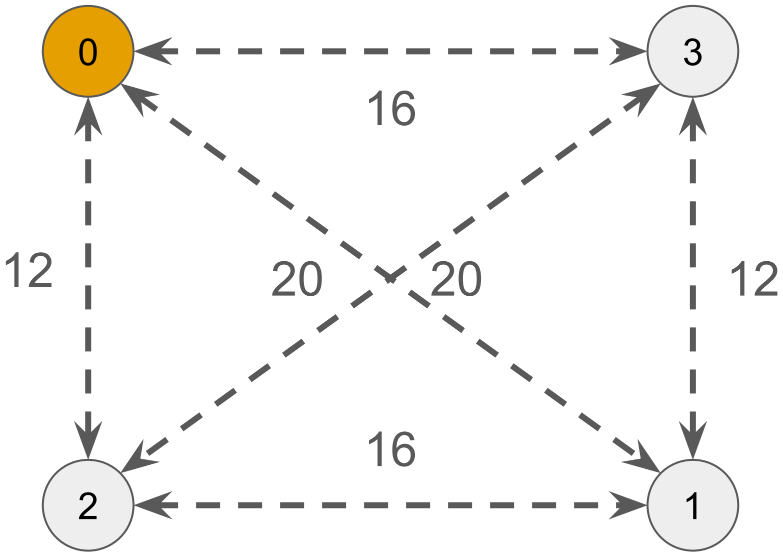
Cost = {48}

48	49	50	51	52	53	54	55	56	57	58	..

48

$$\delta = 1$$

Reverse Look-Ahead



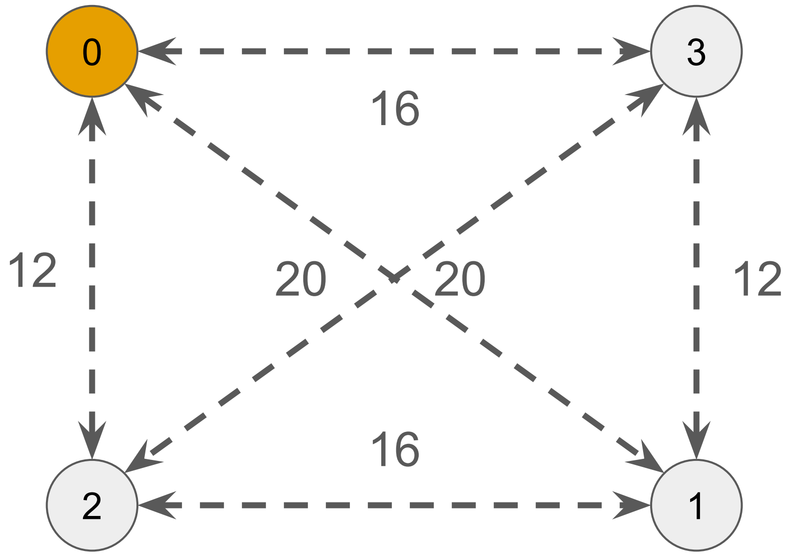
Cost = {**49** .. 80}

48	49	50	51	52	53	54	55	56	57	58	..

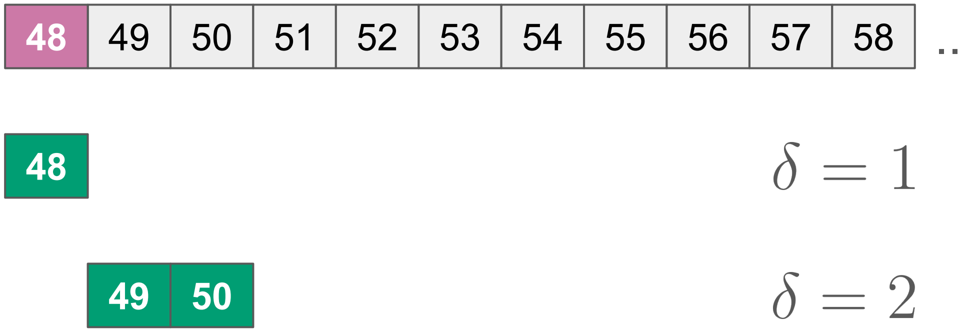
48

$$\delta = 1$$

Reverse Look-Ahead



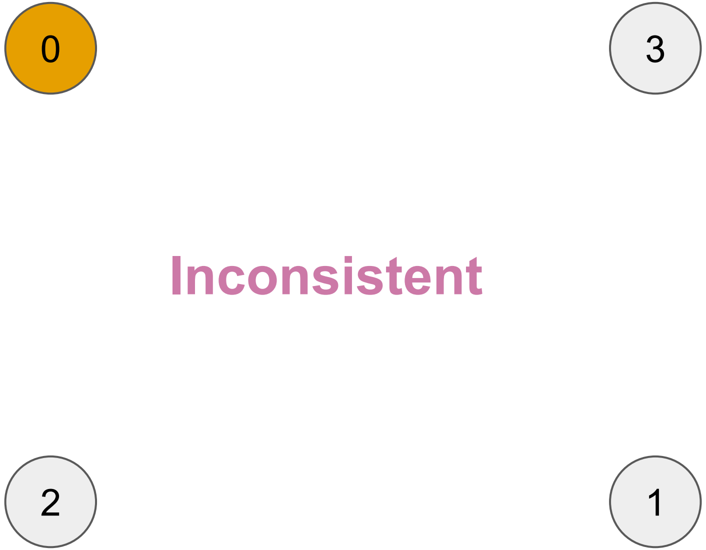
Cost = {49, 50}



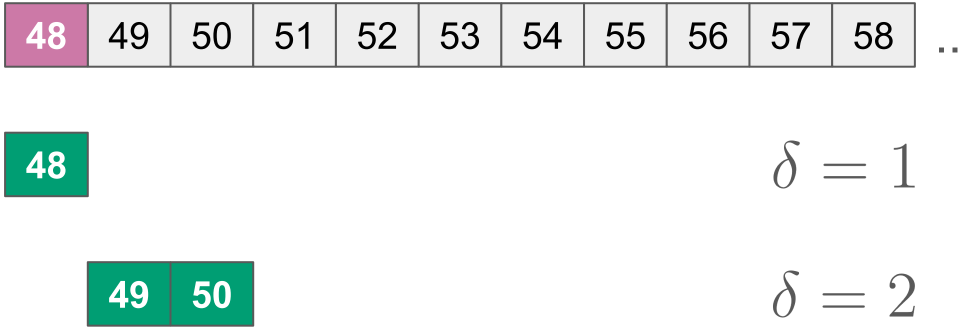
$\delta = 1$

$\delta = 2$

Reverse Look-Ahead



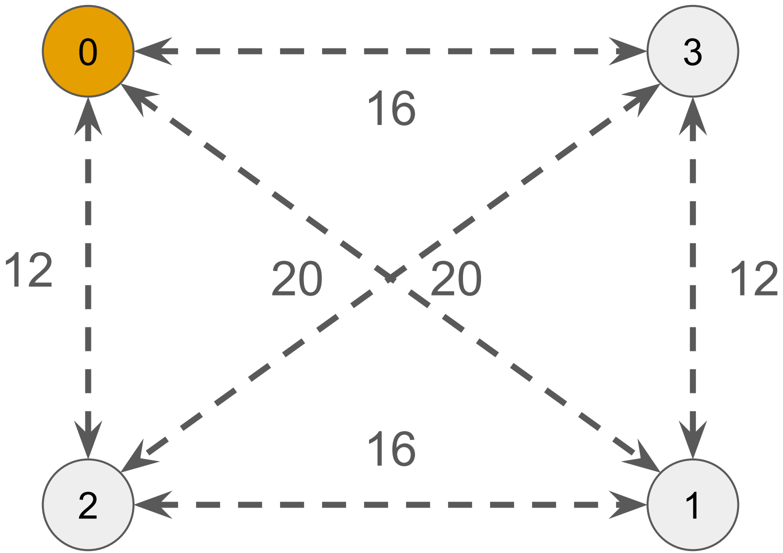
Cost = {49, 50}



$\delta = 1$

$\delta = 2$

Reverse Look-Ahead



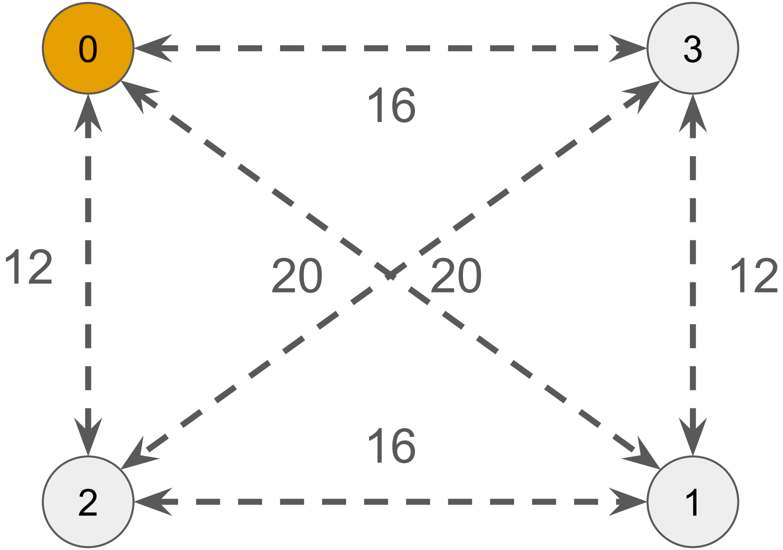
Cost = {**51** .. 80}

48	**49**	**50**	51	52	53	54	55	56	57	58	..

48 $\delta = 1$

49 **50** $\delta = 2$

Reverse Look-Ahead



Cost = {51 .. 54}

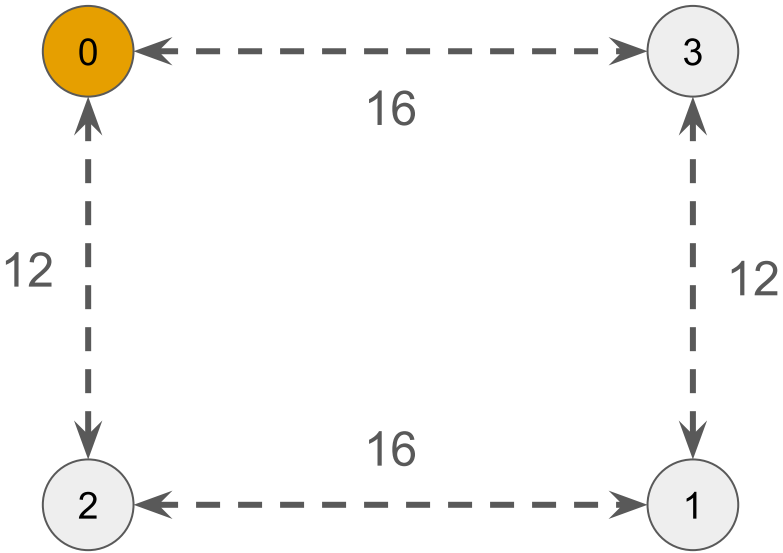
48	49	50	51	52	53	54	55	56	57	58	..

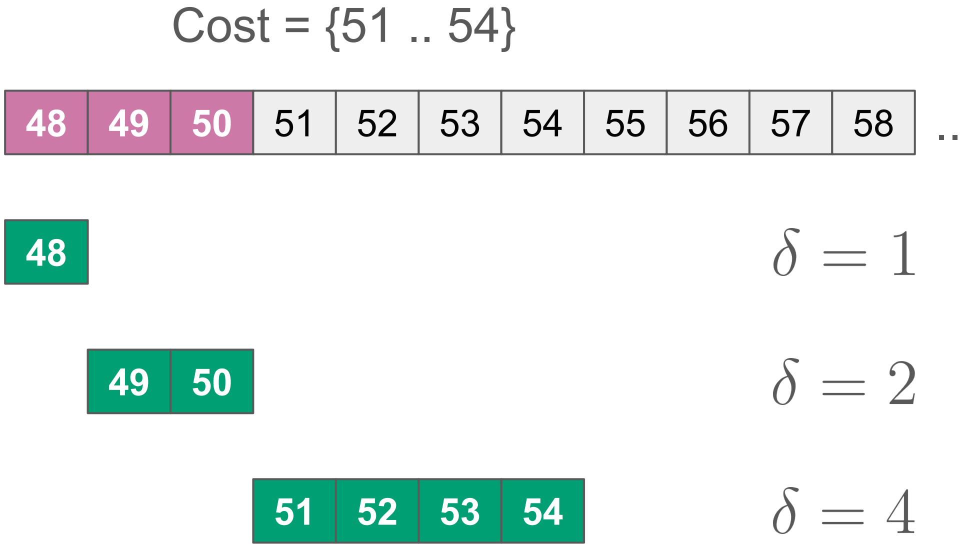
48 $\delta = 1$

49 50 $\delta = 2$

51 52 53 54 $\delta = 4$

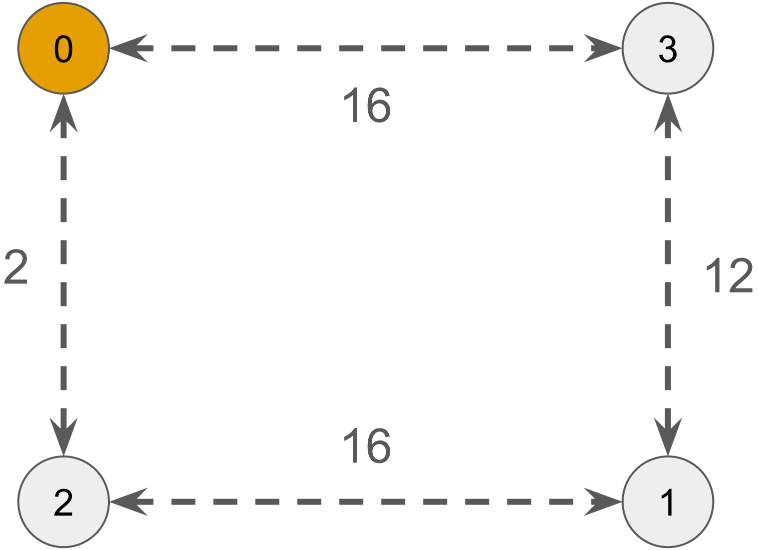
Reverse Look-Ahead



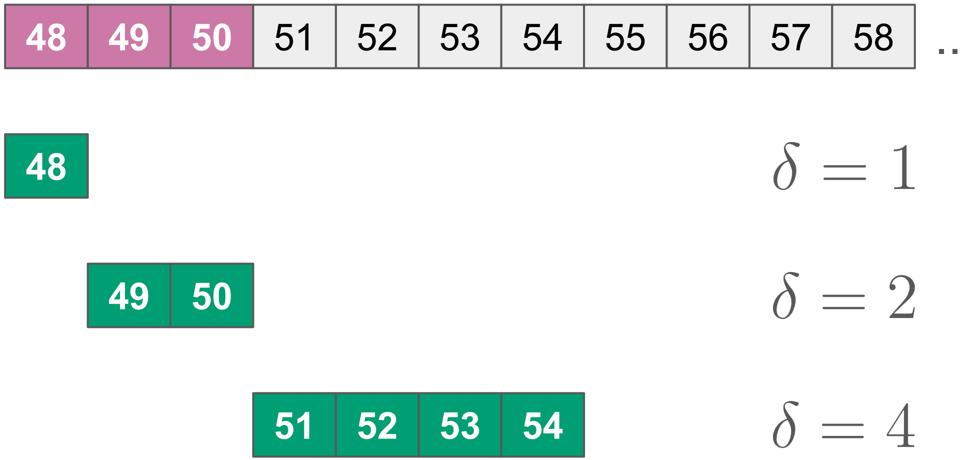


Reverse Look-Ahead

{2, 3} □ select value 2



Cost = {51 .. 54}



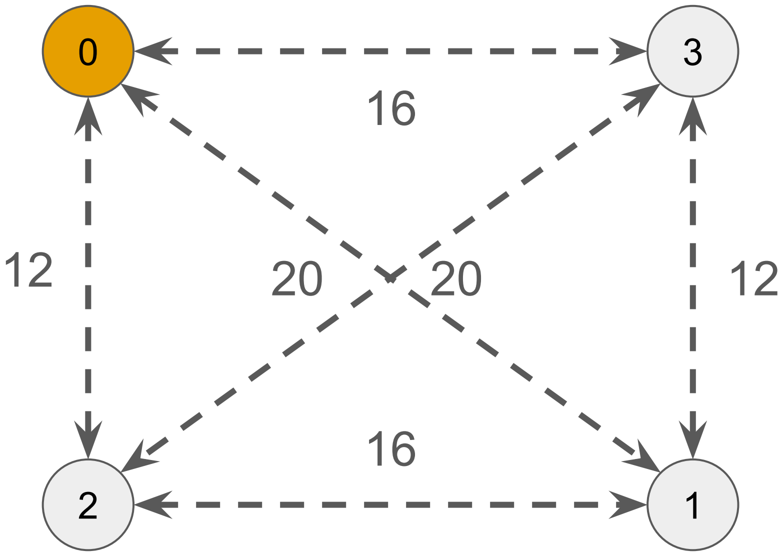
$\delta = 1$

$\delta = 2$

$\delta = 4$

Reverse Look-Ahead

{1, **2**, 3} □ **select value 2**



Cost = {**51** .. 80}

48 49 50 51 52 53 54 55 56 57 58 ..

48 $\delta = 1$

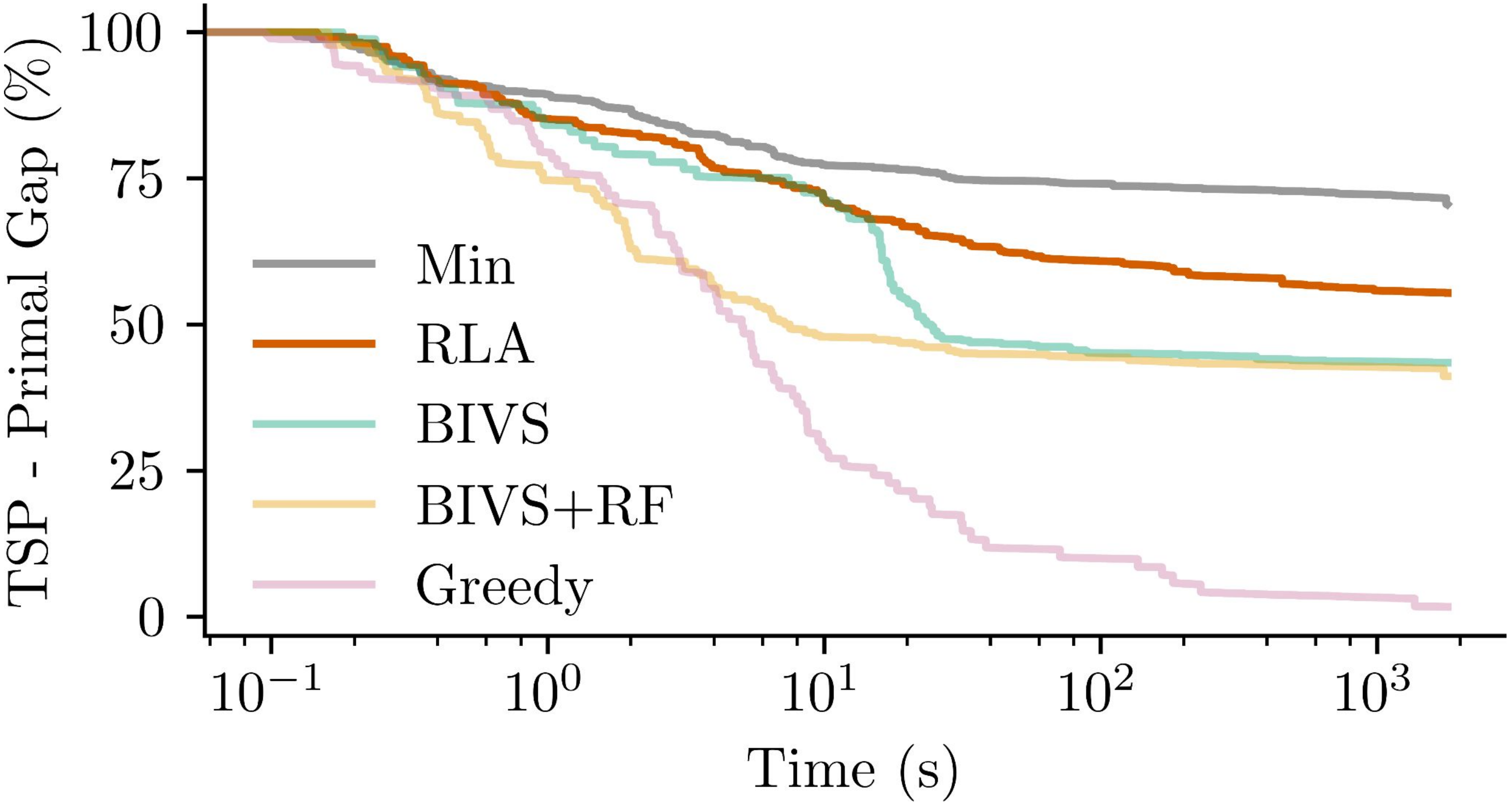
49 50 $\delta = 2$

51 52 53 54 $\delta = 4$

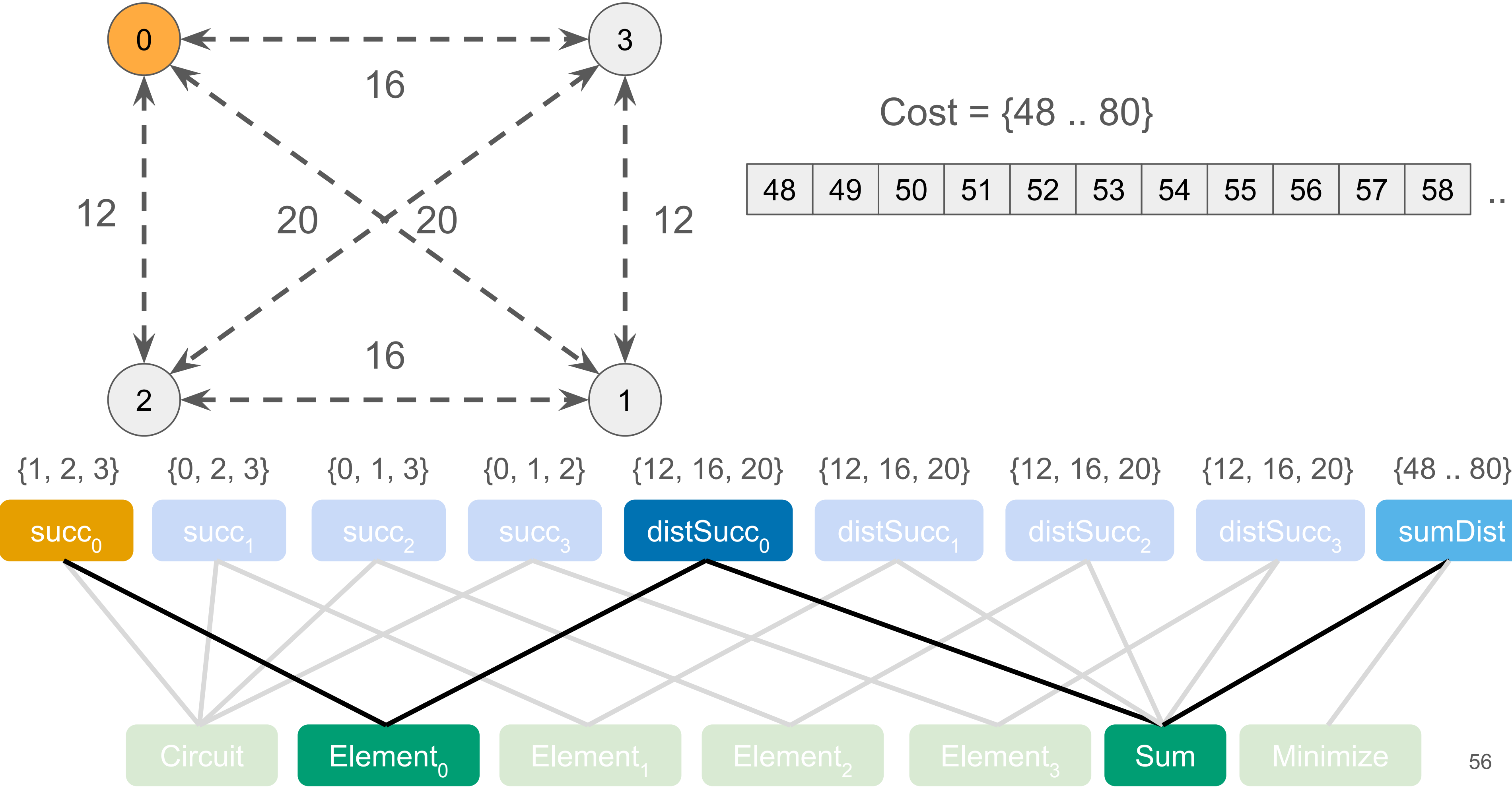
Some considerations about Reverse Look-Ahead (RLA)

- Different exploration than BIVS
  - BIVS with and without RLA might not select the same value
- Different from MinDom
  - Value is picked in reduced domain, not initial one
- May deduce bounds on objective at each selection
- Cost not always smaller than BIVS
  - $\mathcal{O}\left(\log_2(|dom(obj)|) \cdot \mathcal{F}\right)$ compared to $\mathcal{O}\left(|dom(x)| \cdot \mathcal{F}\right)$
- Similar to destructive lower bound in scheduling
  - But used as a value selector instead

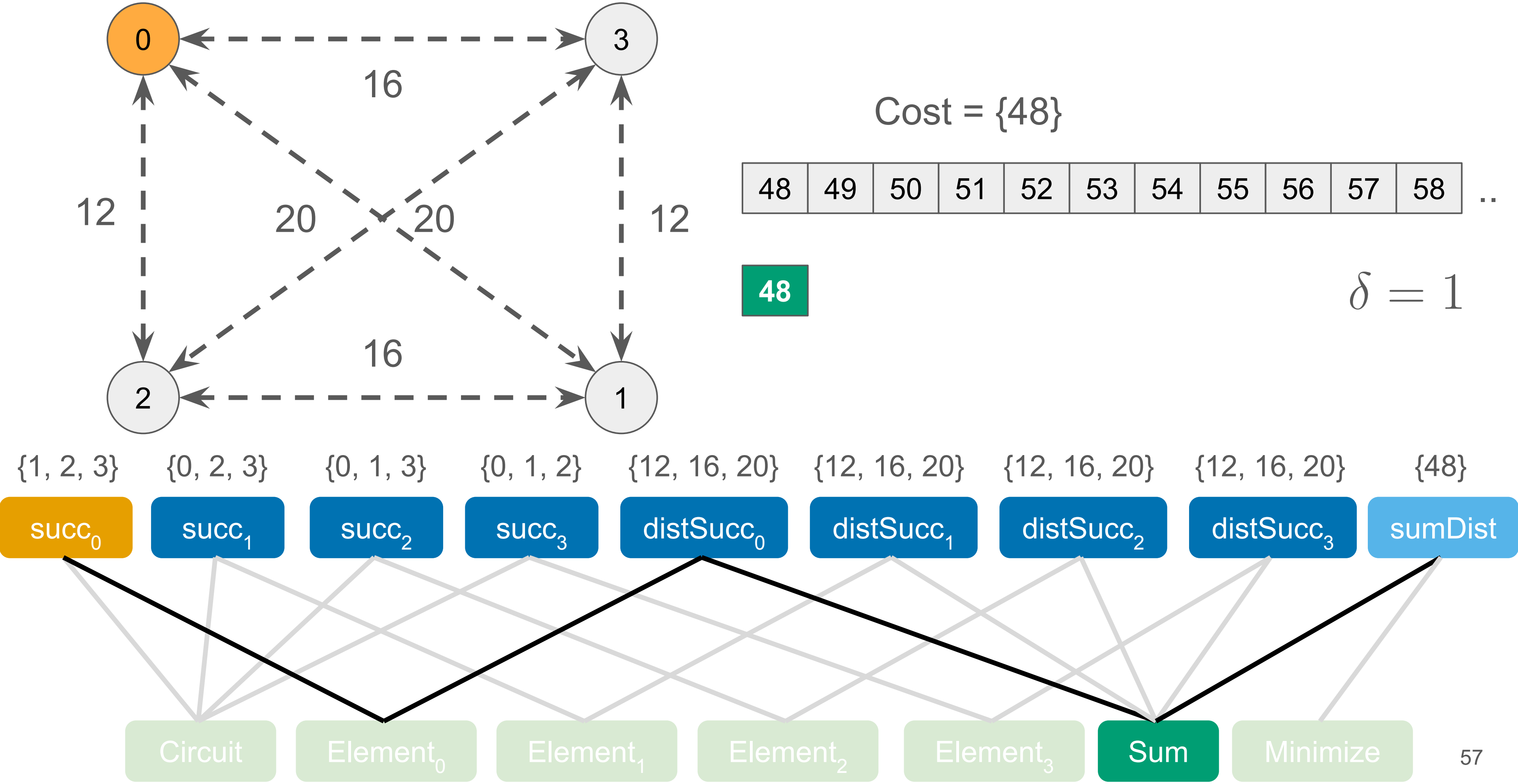
Are we there yet?

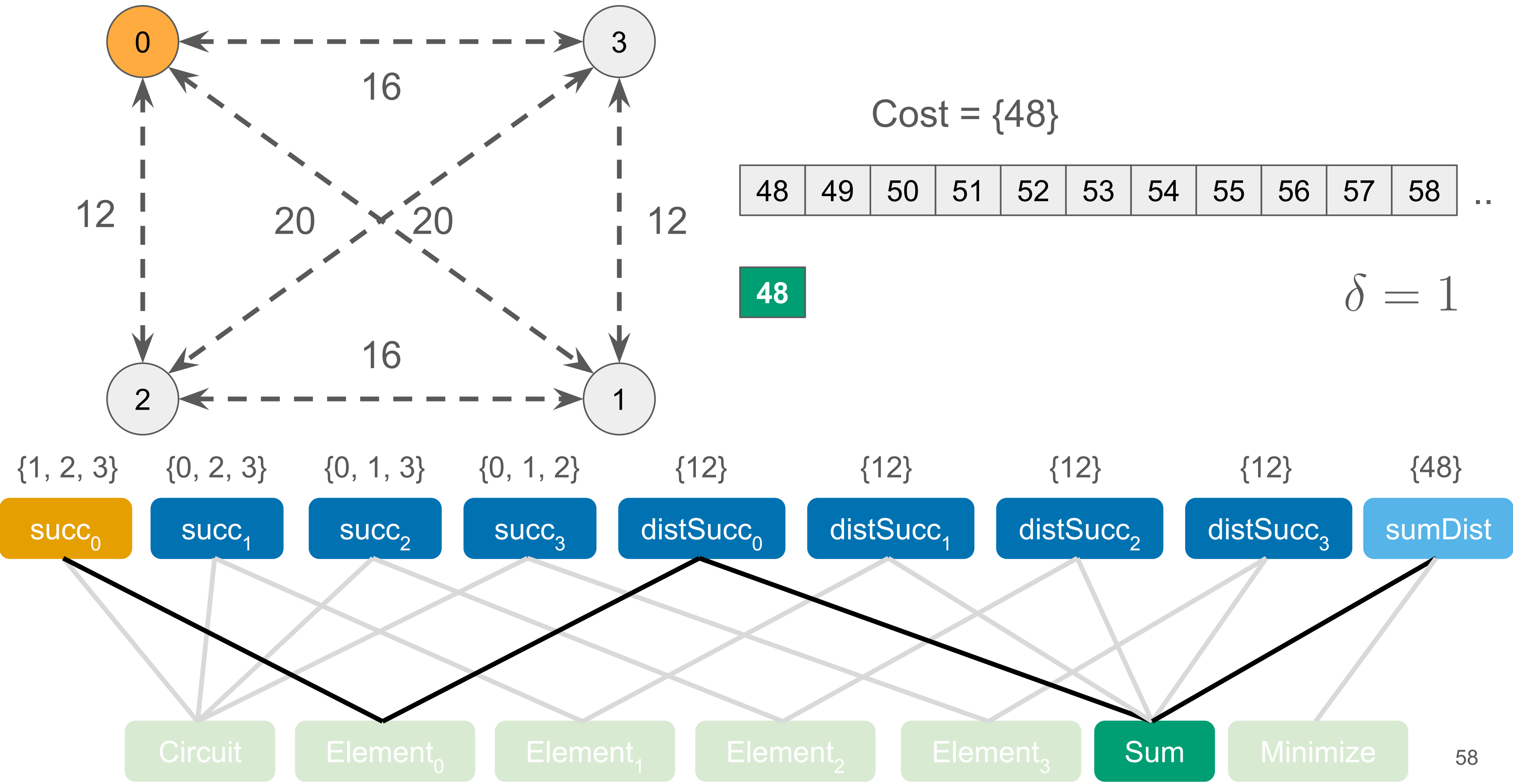


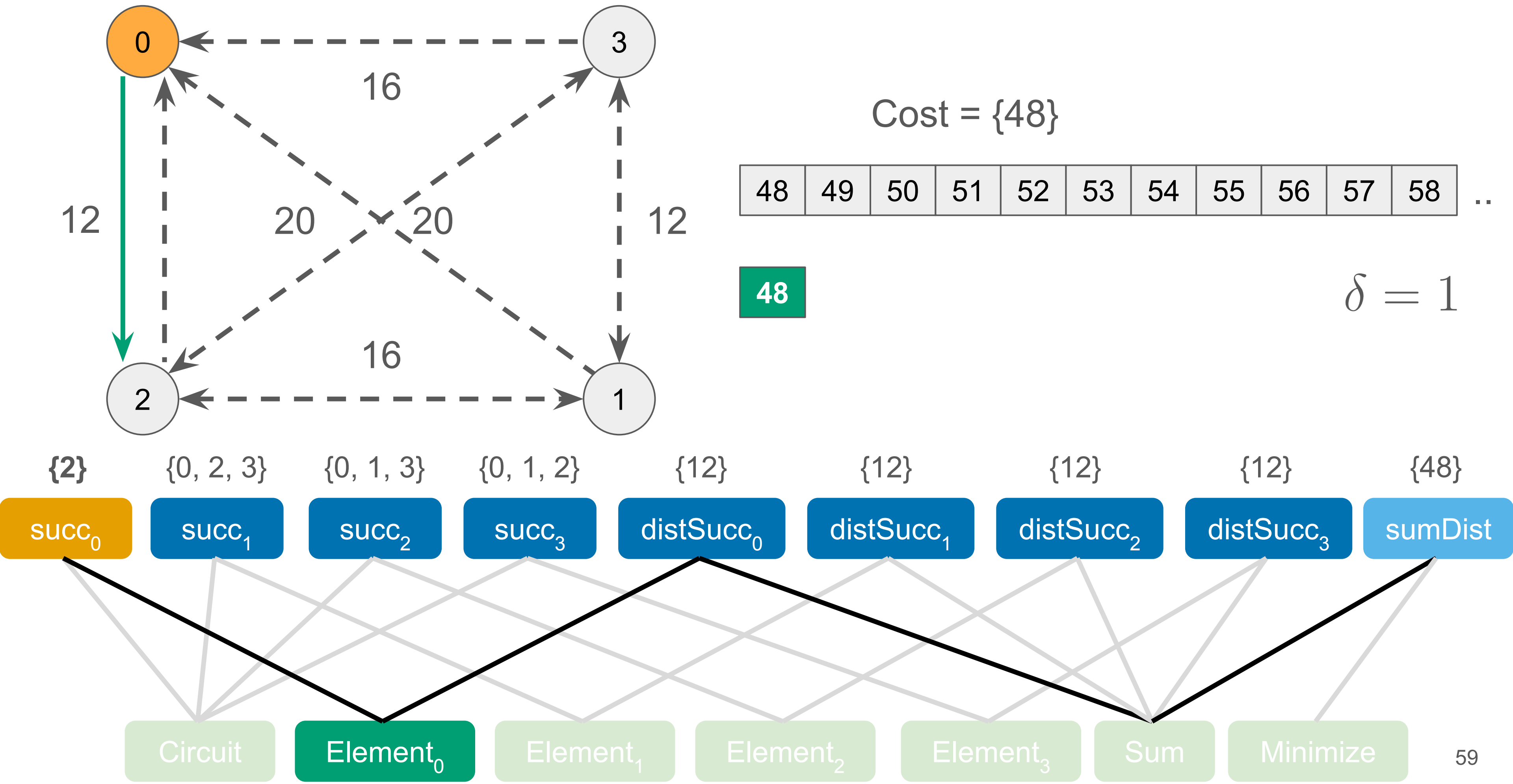
Combining the two methods



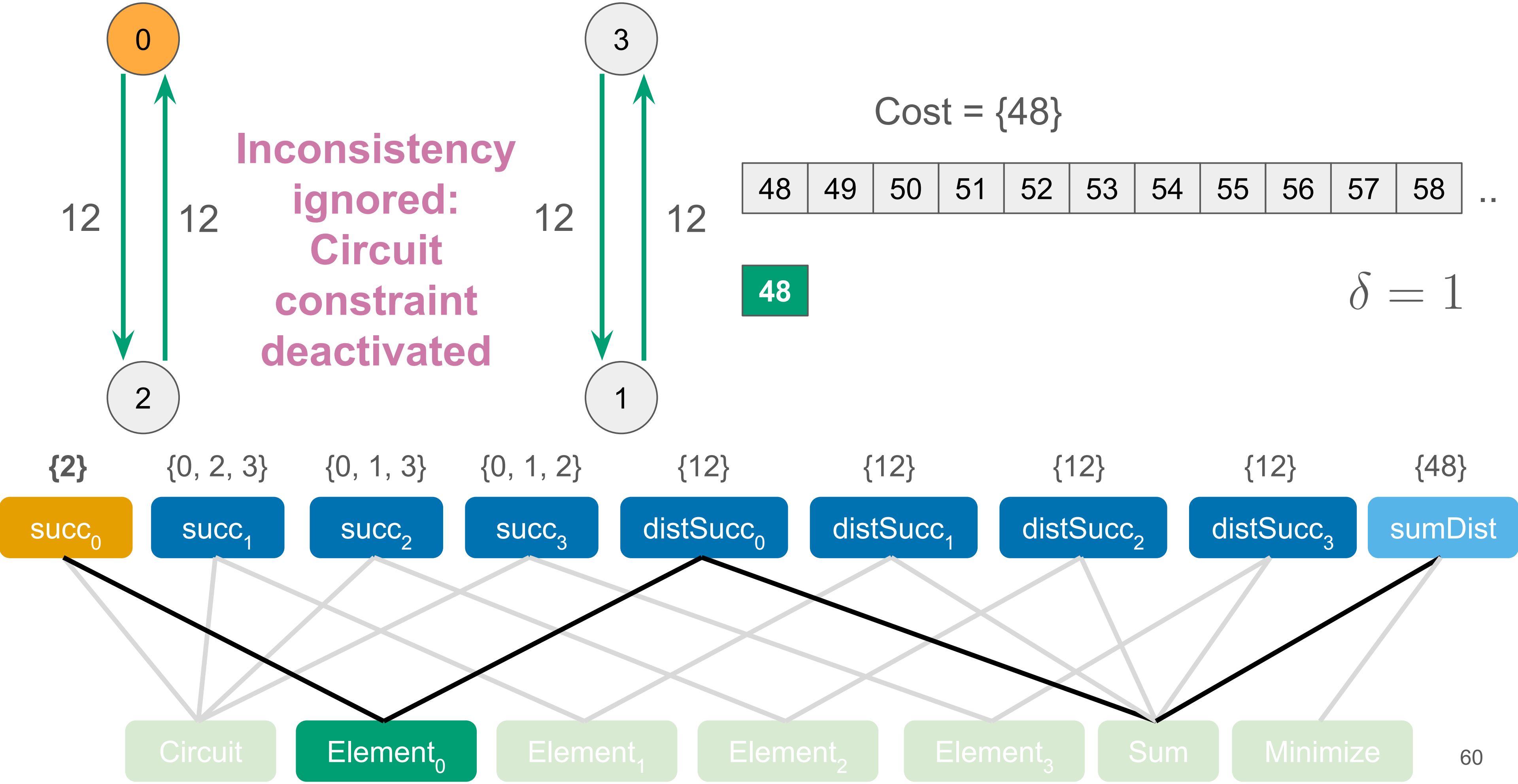
0
3
2
1
16
12
20
20
12
16
Cost = {48 .. 80}
48 49 50 51 52 53 54 55 56 57 58 ..
{1, 2, 3}
{0, 2, 3}
{0, 1, 3}
{0, 1, 2}
{12, 16, 20}
{12, 16, 20}
{12, 16, 20}
{12, 16, 20}
{48 .. 80}
succ0
succ1
succ2
succ3
distSucc0
distSucc1
distSucc2
distSucc3
sumDist
Circuit
Element0
Element1
Element2
Element3
Sum
Minimize

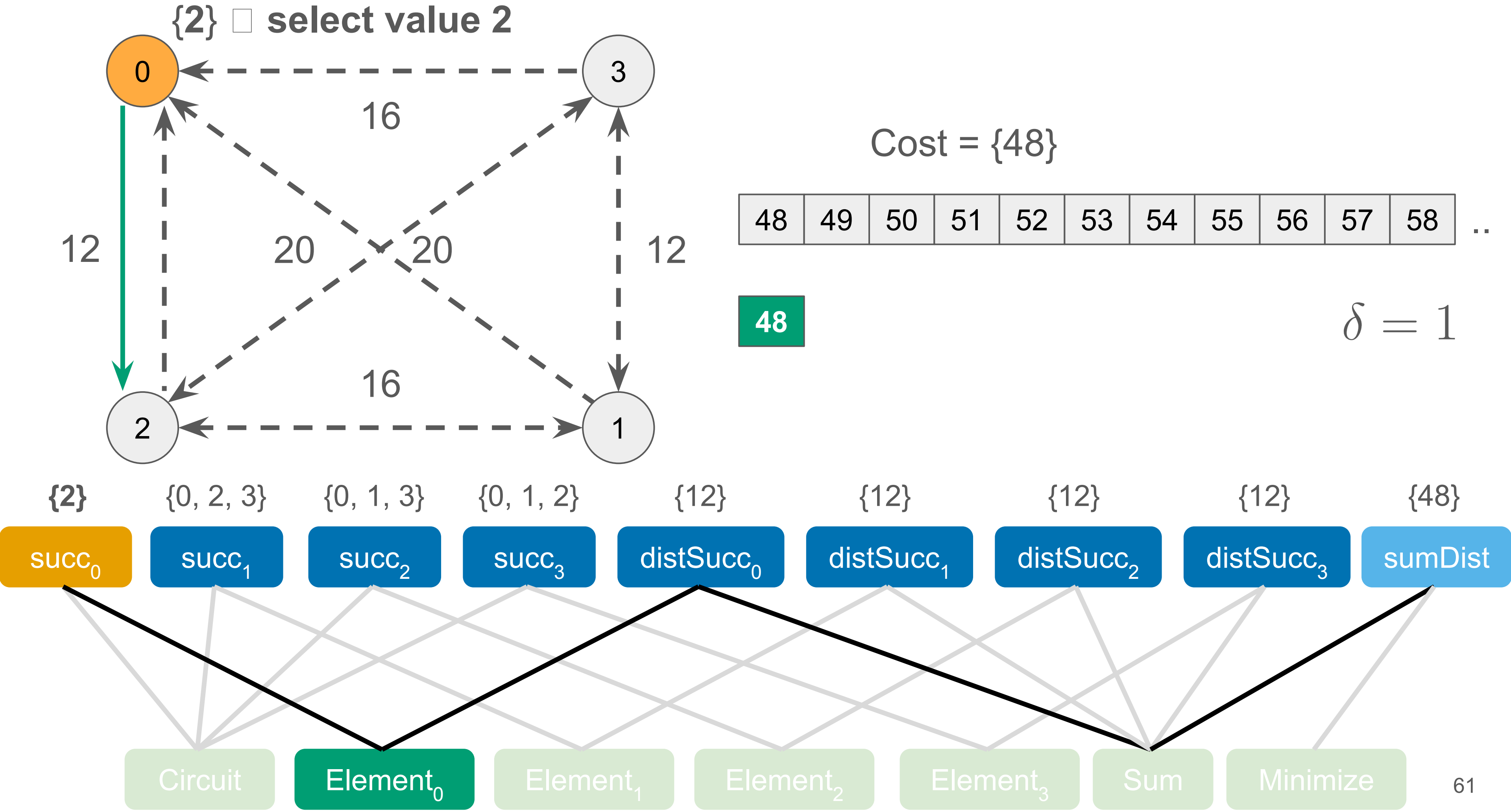

0
3
2
1
16
12
20
20
12
16
Cost = {48}
48
49
50
51
52
53
54
55
56
57
58
..
48
$\delta = 1$
{1, 2, 3}
{0, 2, 3}
{0, 1, 3}
{0, 1, 2}
{12, 16, 20}
{12, 16, 20}
{12, 16, 20}
{12, 16, 20}
{48}
$succ_0$
$succ_1$
$succ_2$
$succ_3$
$distSucc_0$
$distSucc_1$
$distSucc_2$
$distSucc_3$
sumDist
Circuit
$Element_0$
$Element_1$
$Element_2$
$Element_3$
Sum
Minimize


0
3
2
1
16
12
20
20
12
16
Cost = {48}
48
49
50
51
52
53
54
55
56
57
58
..
48
$\delta = 1$
{1, 2, 3}
{0, 2, 3}
{0, 1, 3}
{0, 1, 2}
{12}
{12}
{12}
{12}
{48}
$succ_0$
$succ_1$
$succ_2$
$succ_3$
$distSucc_0$
$distSucc_1$
$distSucc_2$
$distSucc_3$
sumDist
Circuit
$Element_0$
$Element_1$
$Element_2$
$Element_3$
Sum
Minimize



0
3
2
1
16
12
20
20
12
16
Cost = {48}
48 49 50 51 52 53 54 55 56 57 58 ..
48
$\delta = 1$
{2}
{0, 2, 3}
{0, 1, 3}
{0, 1, 2}
{12}
{12}
{12}
{12}
{48}
$succ_0$
$succ_1$
$succ_2$
$succ_3$
$distSucc_0$
$distSucc_1$
$distSucc_2$
$distSucc_3$
sumDist
Circuit
$Element_0$
$Element_1$
$Element_2$
$Element_3$
Sum
Minimize

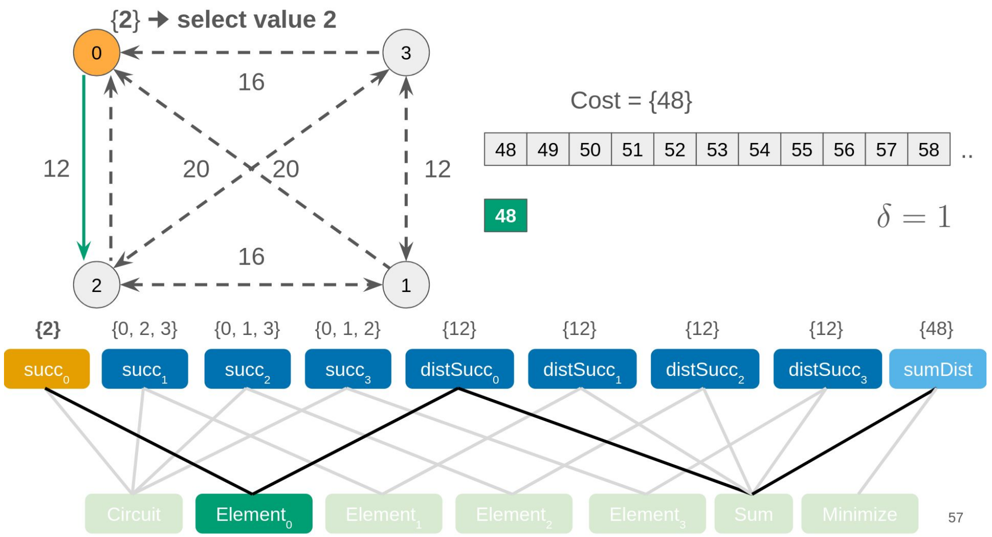

0
3
2
1
12
12
12
12
Inconsistency ignored: Circuit constraint deactivated
Cost = {48}
48 49 50 51 52 53 54 55 56 57 58 ..
48
$\delta = 1$
{2}
{0, 2, 3}
{0, 1, 3}
{0, 1, 2}
{12}
{12}
{12}
{12}
{48}
$succ_0$
$succ_1$
$succ_2$
$succ_3$
$distSucc_0$
$distSucc_1$
$distSucc_2$
$distSucc_3$
sumDist
Circuit
$Element_0$
$Element_1$
$Element_2$
$Element_3$
Sum
Minimize



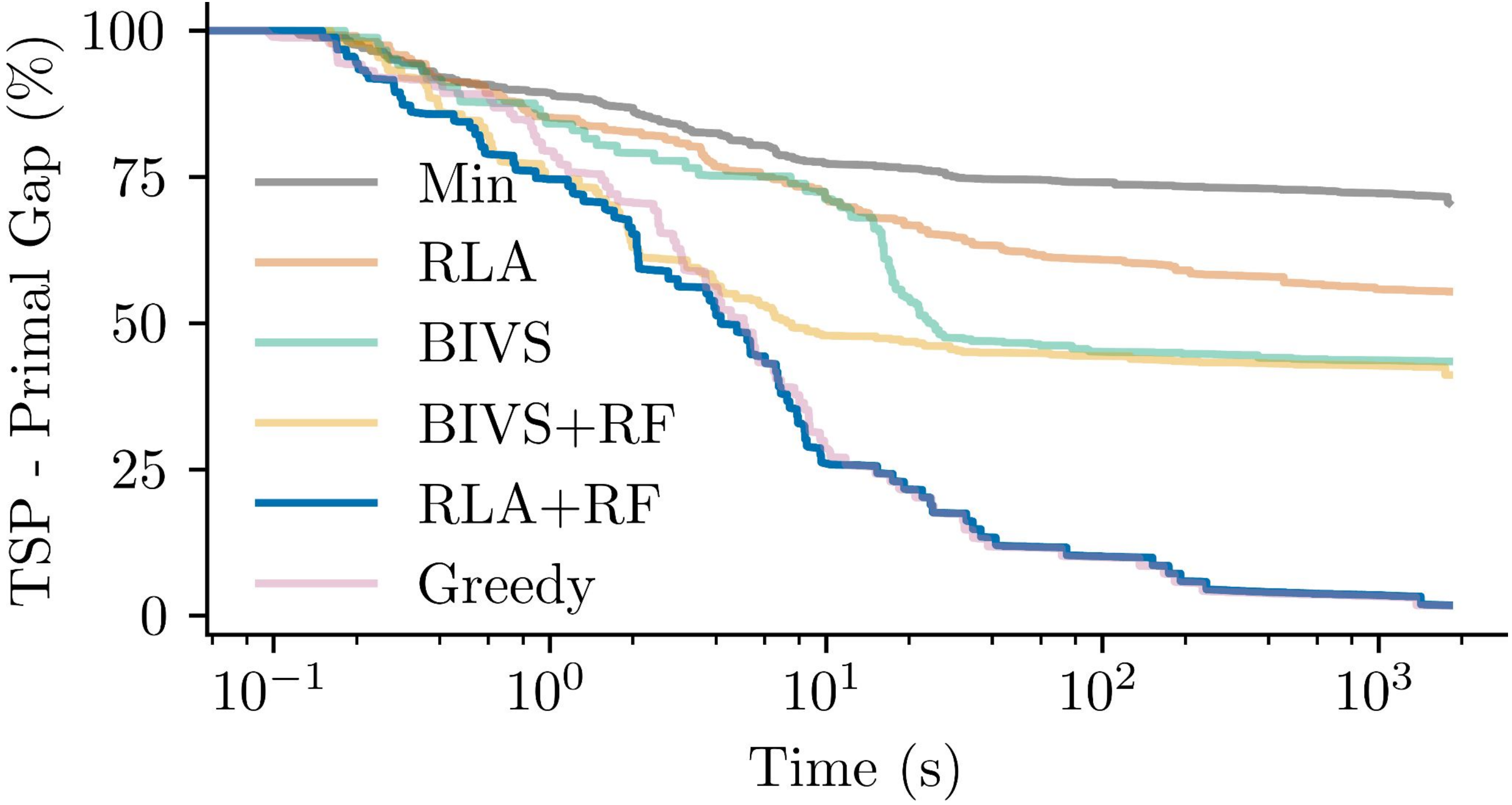
{2} □ select value 2
0
3
2
1
16
12
20
20
12
16
Cost = {48}
48 49 50 51 52 53 54 55 56 57 58 ..
48
$\delta = 1$
{2}
{0, 2, 3}
{0, 1, 3}
{0, 1, 2}
{12}
{12}
{12}
{12}
{48}
$succ_0$
$succ_1$
$succ_2$
$succ_3$
$distSucc_0$
$distSucc_1$
$distSucc_2$
$distSucc_3$
sumDist
Circuit
$Element_0$
$Element_1$
$Element_2$
$Element_3$
Sum
Minimize

RLA + RF

- Faster than BIVS+RF and RLA
- Still slower than MinDom (obviously) but more informed
- Deduces less bounds than RLA
  - Less iterations can be expected
  - Fixpoint not as strong
  - Example with TSP: no bounds deduced
- On the TSP: requires only 2 constraints propagation (cost: $\tilde{\mathcal{F}}$)
  - 1 Sum and 1 Element, no matter the instance size
  - RLA+RF cost: $\mathcal{O}(\tilde{\mathcal{F}})$
  - BIVS+RF cost: $\mathcal{O}(|dom(x)| \cdot \tilde{\mathcal{F}})$



Same performances as greedy!

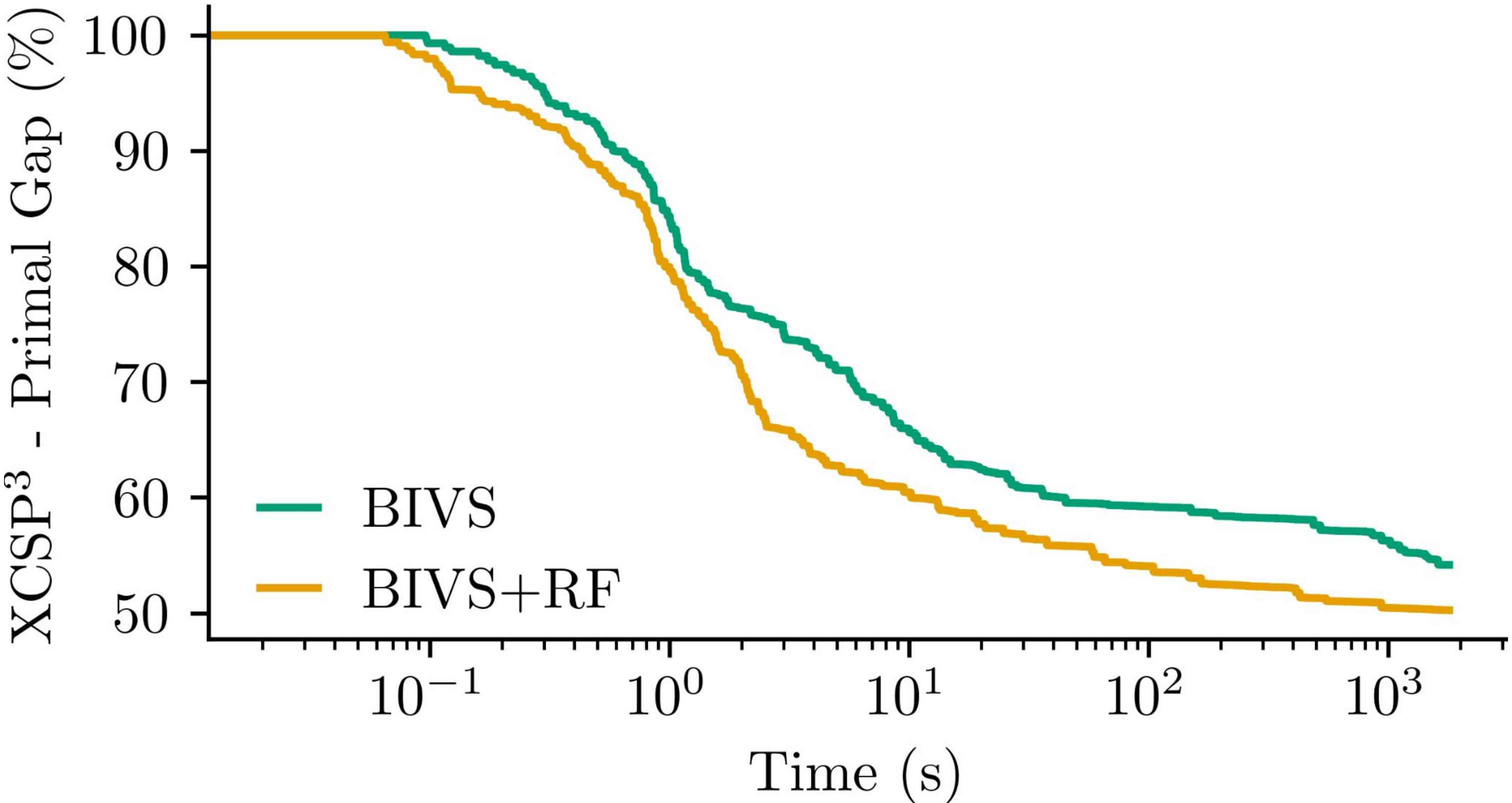


What about other problems?

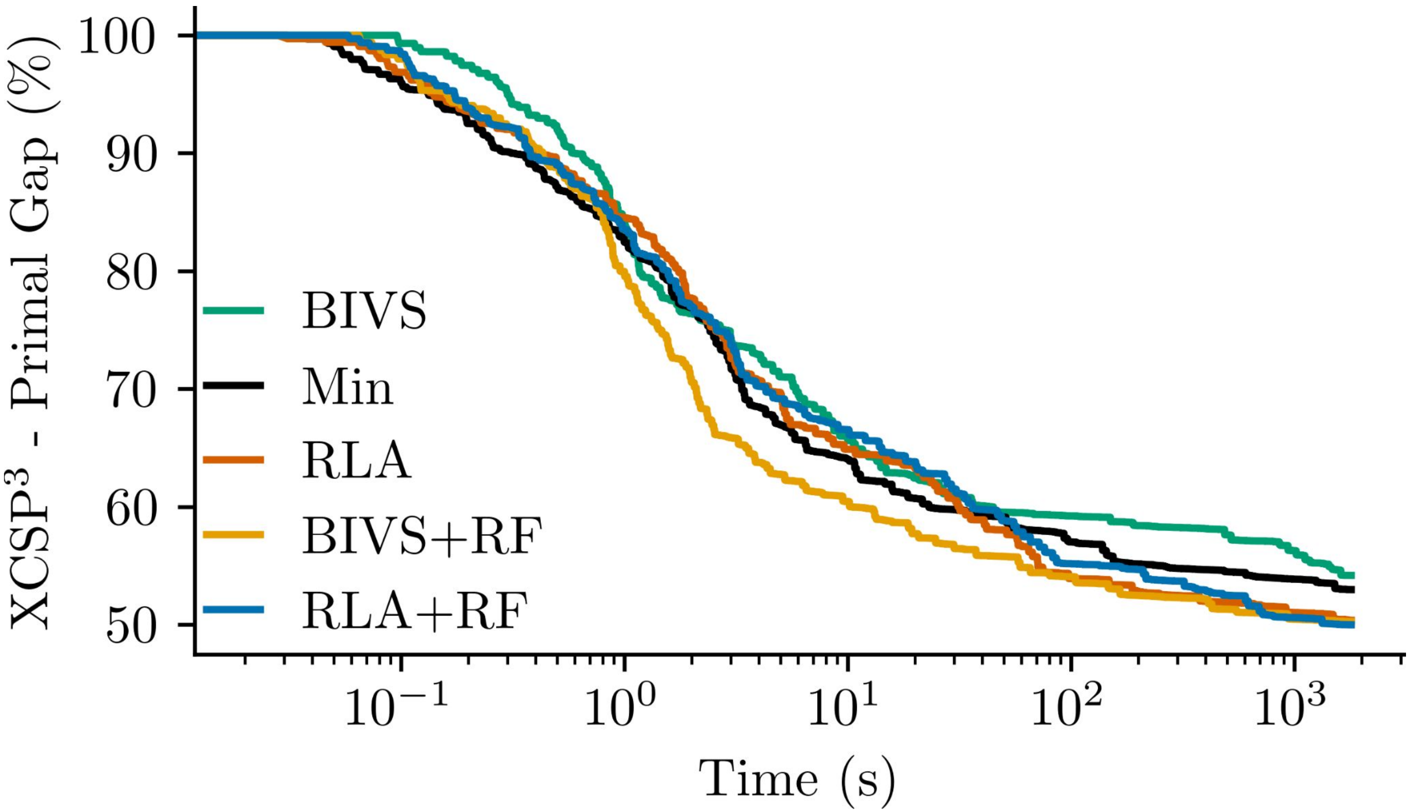
XCSP^3

- Competition from 2023 considered
  - Instances too voluminous discarded
  - 18 problems, 232 instances
  - Various problems: scheduling, routing, assignments, …
- Black-Box optimization
- Variable Selection: DomWDeg + Last Conflict
- Timeout of 30 minutes

Experiments on XCSP^3



Experiments on XCSP^3



Problem (#instances)	Min	BIVS	BIVS+RF	RLA	RLA+RF
AircraftAssemblyLine (20)	95.00 (1)	**90.00 (2)**	100.00 (0)	95.00 (1)	95.00 (1)
CarpetCutting (20)	61.29 (9)	57.17 (9)	65.48 (8)	62.67 (8)	**54.48 (11)**
GBACP (20)	78.86 **(11)**	**61.43** (8)	79.69 (10)	69.54 (9)	85.51 (9)
GeneralizedMKP (15)	49.74 **(15)**	**6.78** (14)	6.81 (14)	26.57 (12)	20.26 (13)
HCPizza (10)	**33.56** (10)	34.91 (10)	34.43 (10)	34.67 (10)	34.62 (10)
Hsp (18)	**0.00 (18)**	5.56 (17)	5.56 (17)	5.56 (17)	**0.00 (18)**
KMedian (15)	50.84 (8)	57.32 (7)	**43.96 (9)**	56.01 (7)	50.84 (8)
KidneyExchange (14)	44.63 **(14)**	85.71 (3)	51.11 (10)	**43.68** (13)	52.79 (10)
LargeScaleScheduling (9)	66.76 **(6)**	66.83 **(6)**	66.83 **(6)**	45.56 (5)	**34.44 (6)**
PSP1 (8)	100.00 (0)	100.00 (0)	**87.50 (1)**	100.00 (0)	100.00 (0)
PSP2 (8)	**87.50** (1)	**87.50** (1)	87.62 (1)	**87.50** (1)	87.66 (1)
ProgressiveParty (7)	57.14 (3)	57.14 (3)	57.14 (3)	57.14 (3)	57.14 (3)
RIP (12)	5.06 (12)	6.31 (12)	4.59 (12)	**3.24** (12)	4.78 (12)
Rulemining (9)	100.00 (0)	100.00 (0)	100.00 (0)	100.00 (0)	100.00 (0)
SREFLP (15)	7.27 (15)	**3.08** (15)	7.44 (15)	8.78 (15)	7.72 (15)
Sonet (16)	**1.40** (16)	2.28 (16)	3.42 (16)	2.42 (16)	3.15 (16)
TSPTW1 (8)	87.81 **(1)**	100.00 (0)	87.84 **(1)**	87.81 **(1)**	**87.50 (1)**
TSPTW2 (8)	**75.19 (2)**	87.50 (1)	**75.19 (2)**	75.42 **(2)**	75.24 **(2)**
All (232)	52.02 **(142)**	51.04 (124)	50.23 (135)	49.85 (132)	**49.52** (136)

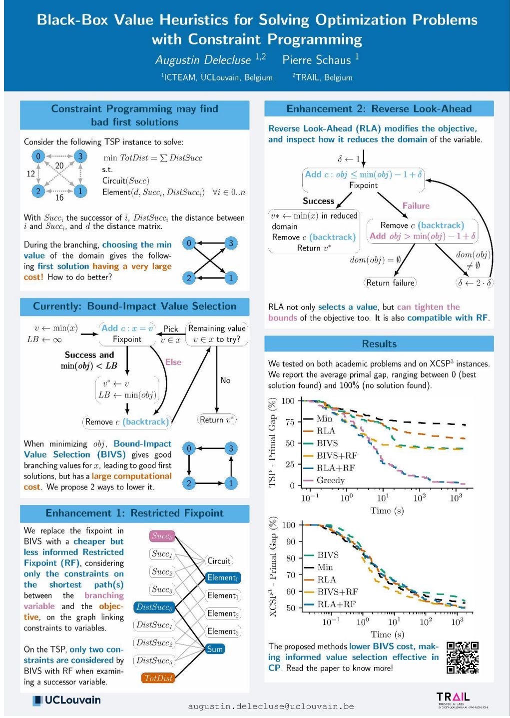
Gap (#instances with a feasible solution found)

Wrapping up the results

- RF adds value on both BIVS and RLA
- MinDom finds more feasible solutions
  - Hypothesis: more nodes in the search tree are explored, giving variable selection more opportunity to learn
- RLA+RF gives the best gaps on average
- Best heuristic is problem-dependent
  - Result also observed on papers about variable selection heuristics

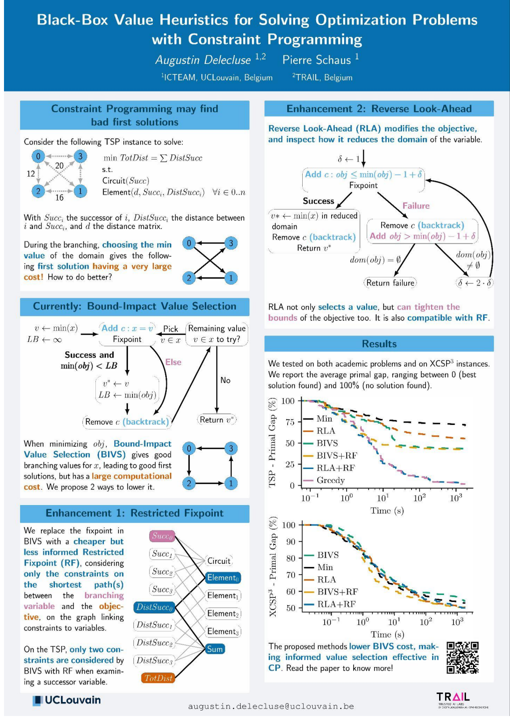
Conclusion

- BIVS is an efficient but costly heuristic for selecting branching values
- We presented 2 methods to lower its cost
    - **Restricted Fixpoint**, considering a subset of constraints
    - **Reverse Look-Ahead**, shrinking the objective and observing the impact on branching variable
- Both methods are easy to implement
- They improved the performances (in both speed and objective values)
- Could become default value selection in CP solvers
- Algorithms, source code and more experiments available in the paper



Conclusion

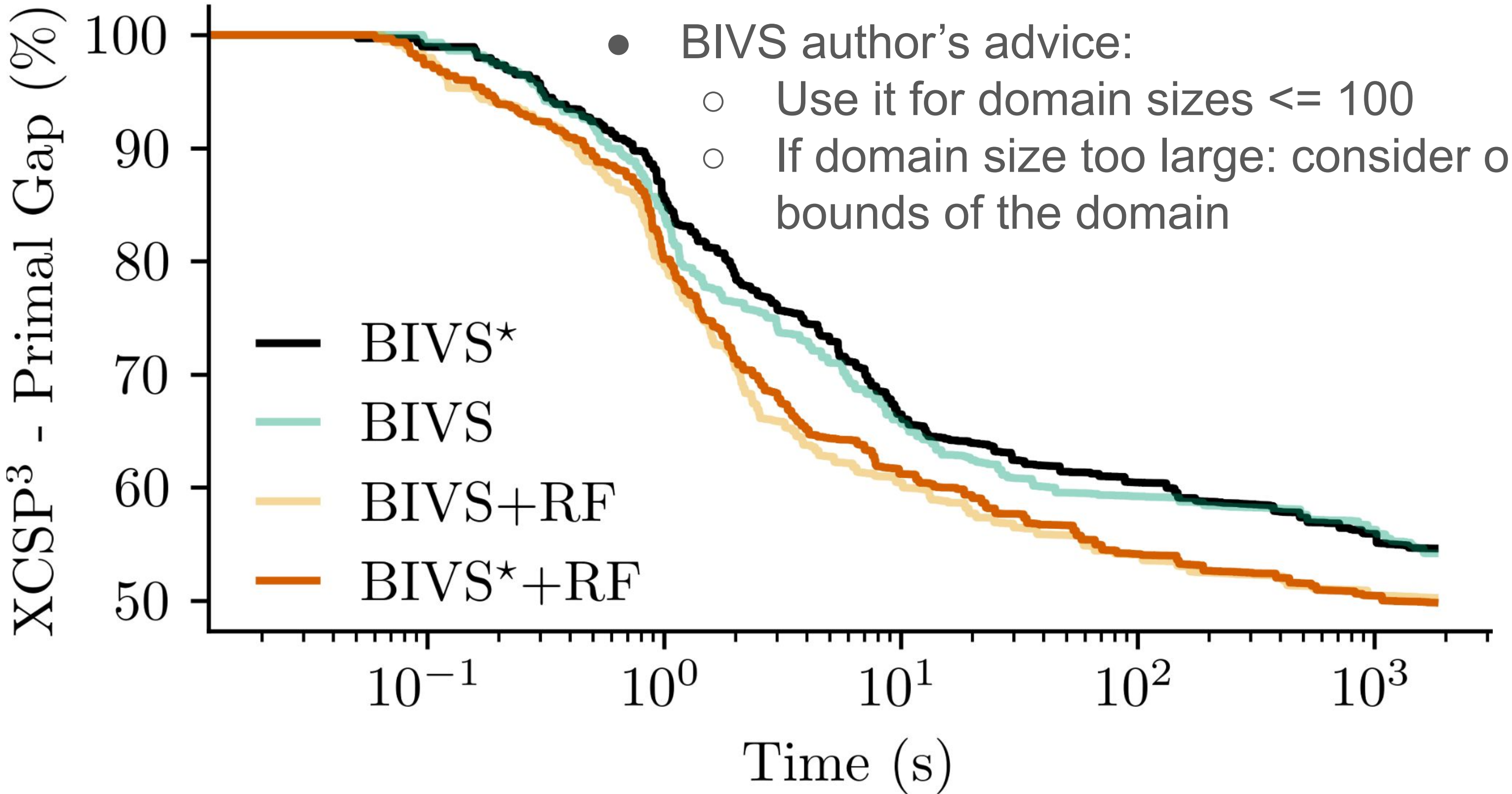
- BIVS is an efficient but costly heuristic for selecting branching values
- We presented 2 methods to lower its cost
    - **Restricted Fixpoint**, considering a subset of constraints
    - **Reverse Look-Ahead**, shrinking the objective and observing the impact on branching variable
- Both methods are easy to implement
- They improved the performances (in both speed and objective values)
- Could become default value selection in CP solvers
- Algorithms, source code and more experiments available in the paper



Thanks for your attention!

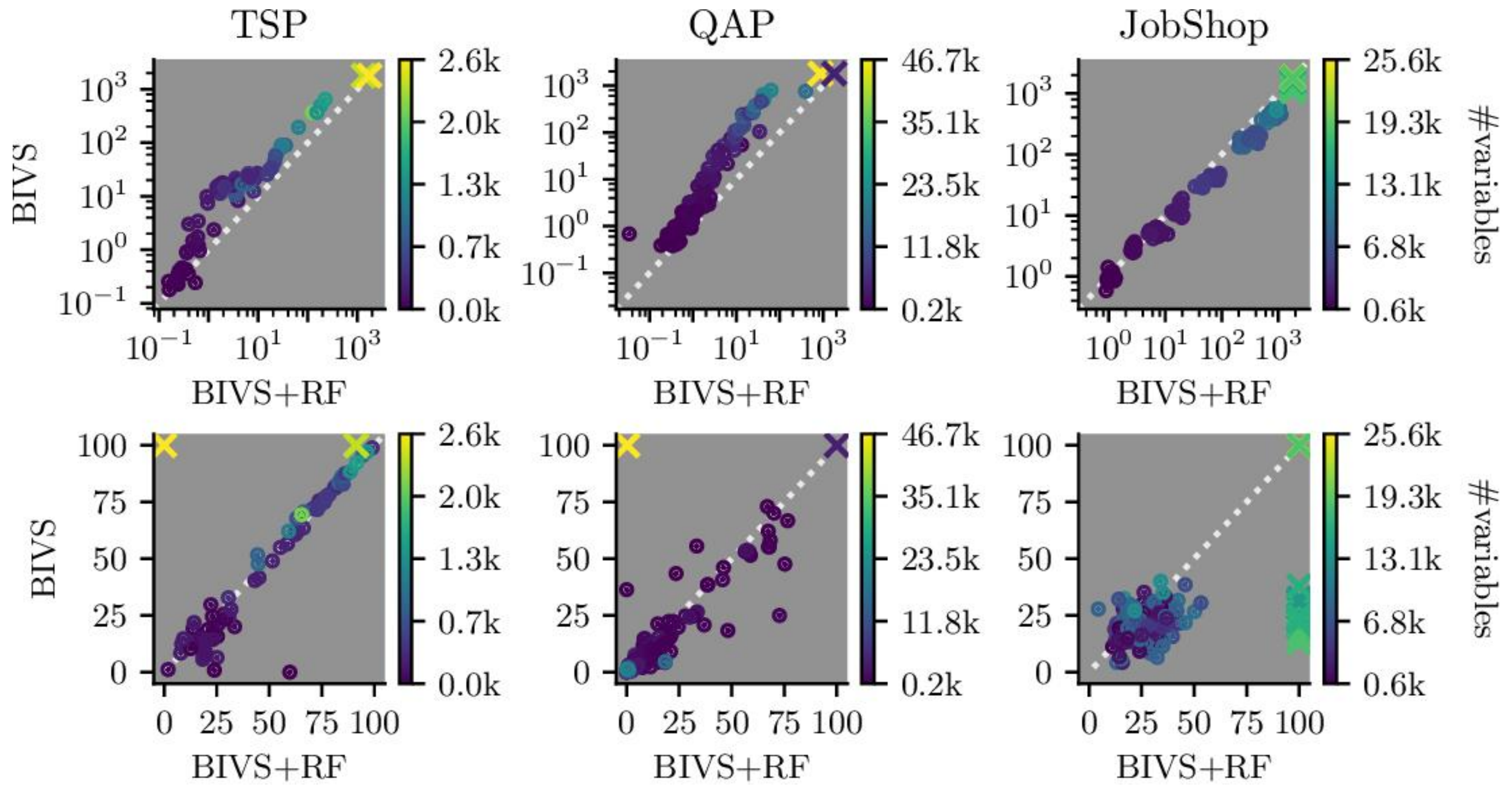
Experiments on XCSP^3

- BIVS author's advice:
  - Use it for domain sizes <= 100
  - If domain size too large: consider only the bounds of the domain



Runtime (s)

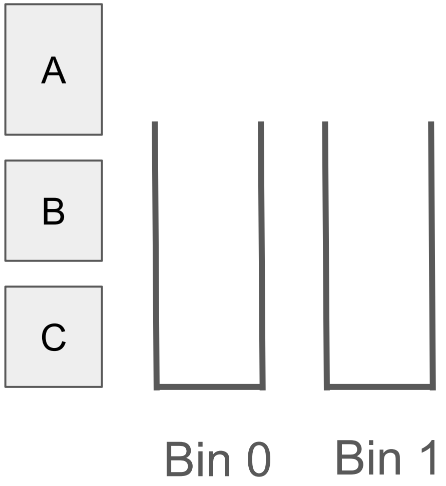
Primal gap



Differences with ranking values before the search

- Consider BinPacking problem
- Minimize maximum load

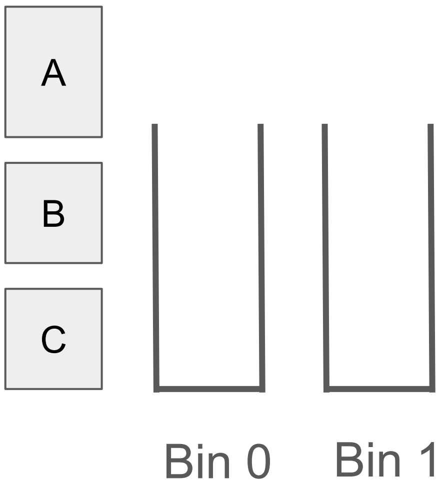
Initial ranking: all bins have same values

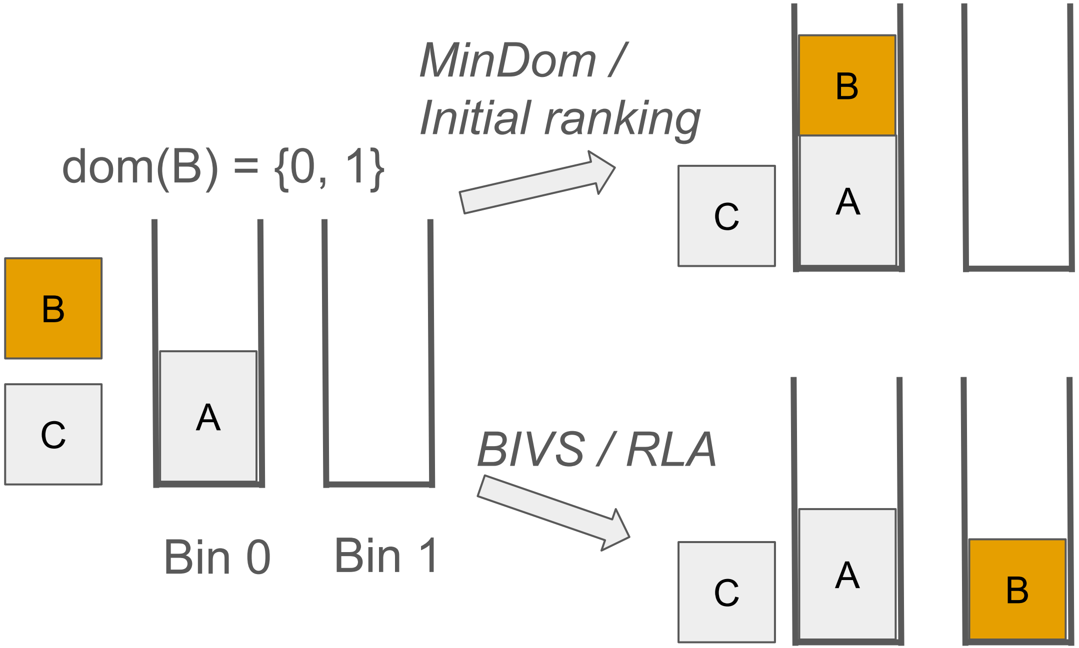


Differences with ranking values before the search

- Consider BinPacking problem
- Minimize maximum load

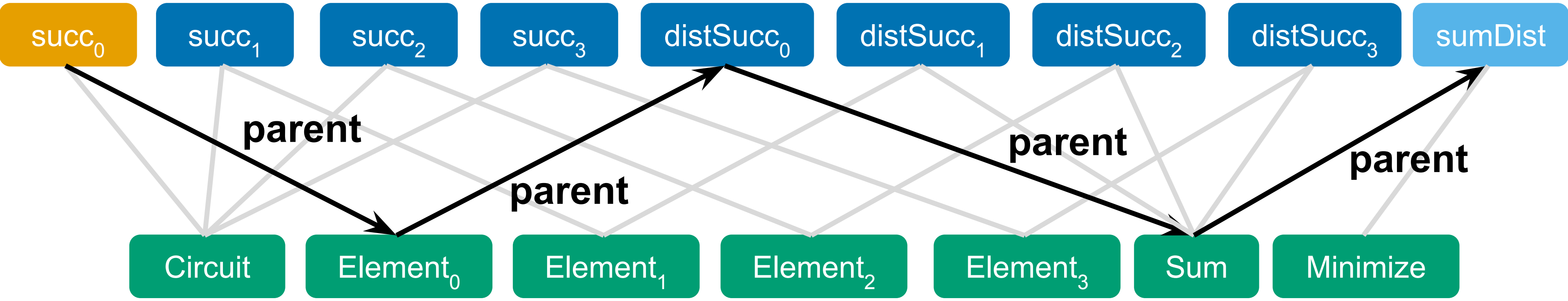
Initial ranking: all bins have same values





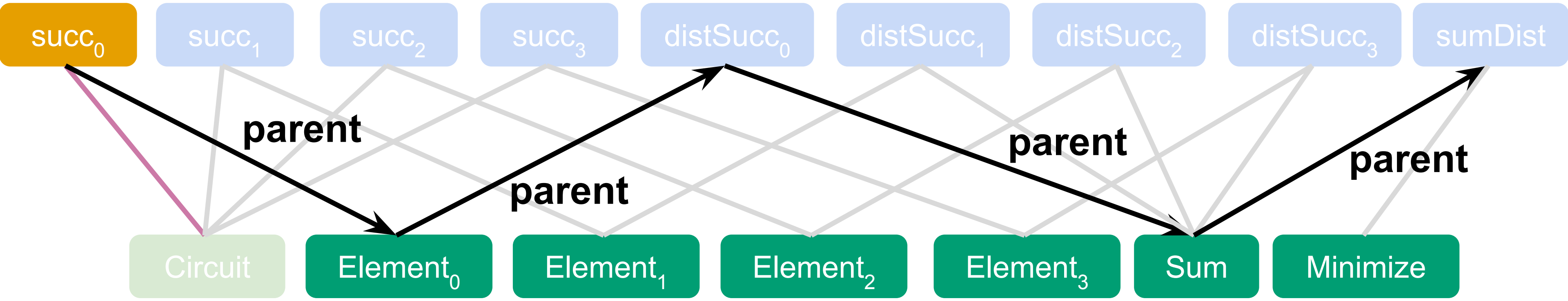
How to run a Restricted Fixpoint in a solver

- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
  - Temporary, deactivation removed upon backtrack



How to run a Restricted Fixpoint in a solver

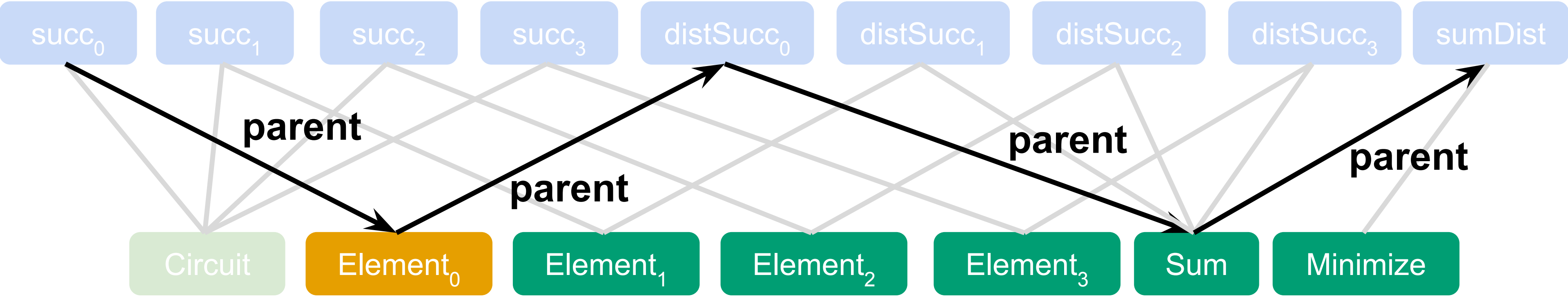
- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
  - Temporary, deactivation removed upon backtrack
- Starting from branching variable, deactivate in the scope constraints that are not parent and not already marked



In scope and deactivated

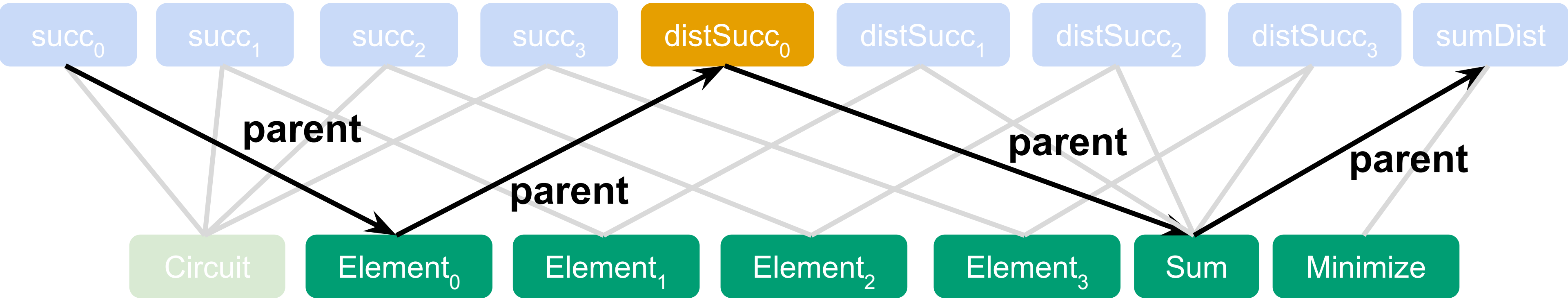
How to run a Restricted Fixpoint in a solver

- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
  - Temporary, deactivation removed upon backtrack
- Starting from branching variable, deactivate in the scope constraints that are not parent and not already marked



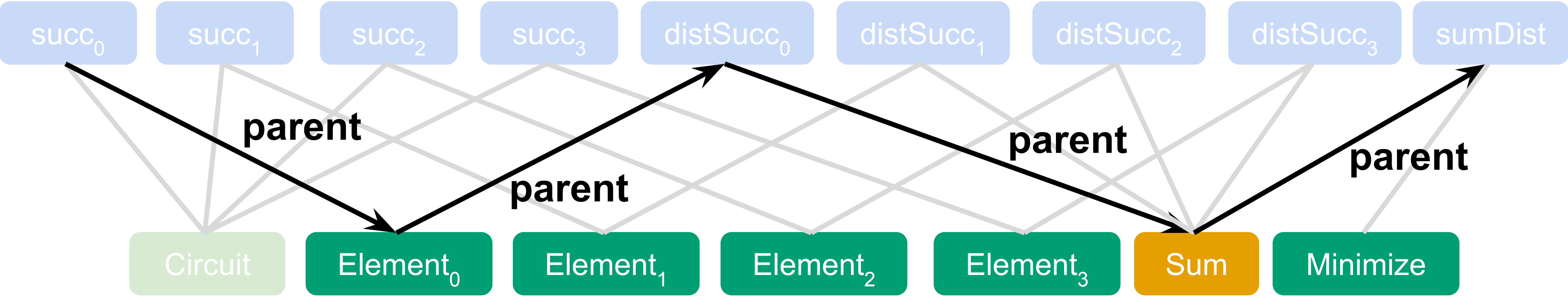
How to run a Restricted Fixpoint in a solver

- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
  - Temporary, deactivation removed upon backtrack
- Starting from branching variable, deactivate in the scope constraints that are not parent and not already marked



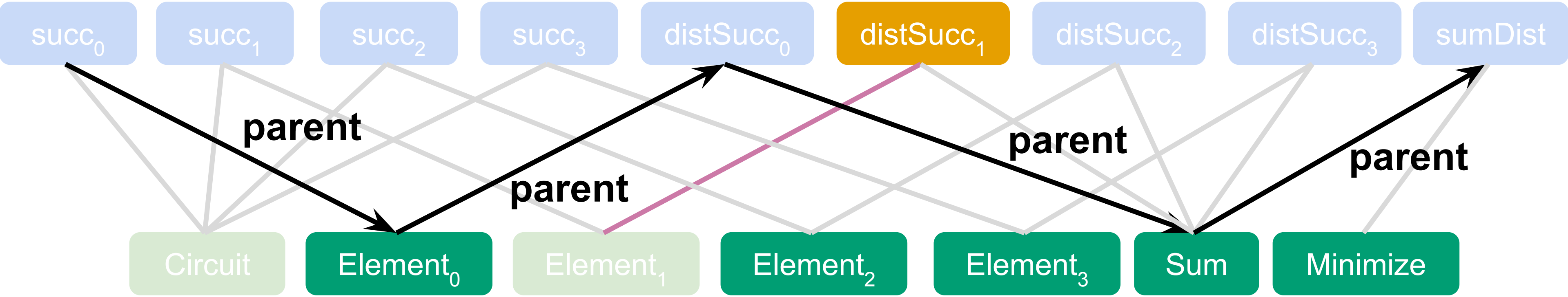
How to run a Restricted Fixpoint in a solver

- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
  - Temporary, deactivation removed upon backtrack
- Starting from branching variable, deactivate in the scope constraints that are not parent and not already marked



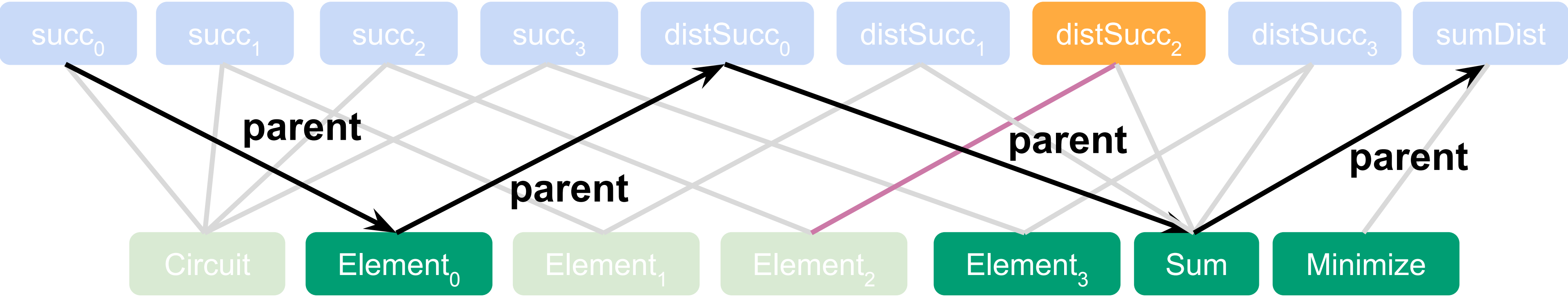
How to run a Restricted Fixpoint in a solver

- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
  - Temporary, deactivation removed upon backtrack
- Starting from branching variable, deactivate in the scope constraints that are not parent and not already marked



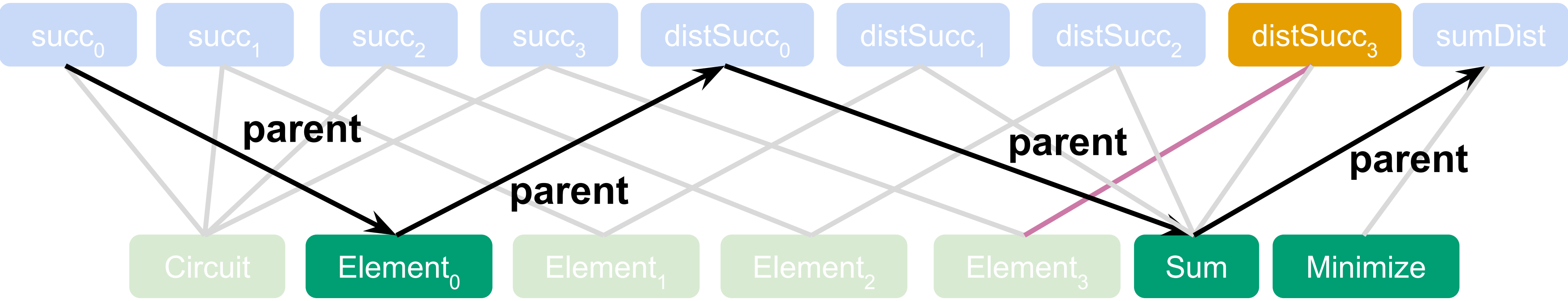
How to run a Restricted Fixpoint in a solver

- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
    - Temporary, deactivation removed upon backtrack
- Starting from branching variable, deactivate in the scope constraints that are not parent and not already marked



How to run a Restricted Fixpoint in a solver

- Example in Choco-Solver
- setPassive() method from propagator: deactivate its propagation
  - Temporary, deactivation removed upon backtrack
- Starting from branching variable, deactivate in the scope constraints that are not parent and not already marked



How to run a Restricted Fixpoint in a solver

- This is but one way to implement it
- Others are possible
  - For instance propagating “manually” the constraints (require to implement another fixpoint method, taking as input the constraints to consider)
- Advantage of this one:
  - Can tell if shortest paths are still valid when parsing them
  - No need to implement a new fixpoint method, no weird hidden interactions with solver