A Guide to Budgeted Tree Search

Nathan R. Sturtevant
University of Alberta
Amii Fellow, CIFAR Chair

Malte Helmert
Universität Basel

Talk Overview

- Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*

Talk Overview

- Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*
- Companion work to original paper on IBEX

  (Helmert, Lattimore, Lelis, Orseau, Sturtevant, IJCAI 2019)

Talk Overview

- Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*
- Companion work to original paper on IBEX

  (Helmert, Lattimore, Lelis, Orseau, Sturtevant, IJCAI 2019)

- Why do we need BTS?

Talk Overview

- Budgeted Tree Search (BTS) is a new algorithm with better worst-case guarantees than IDA*
- Companion work to original paper on IBEX

  (Helmert, Lattimore, Lelis, Orseau, Sturtevant, IJCAI 2019)

- Why do we need BTS?
- How does BTS work?

Start

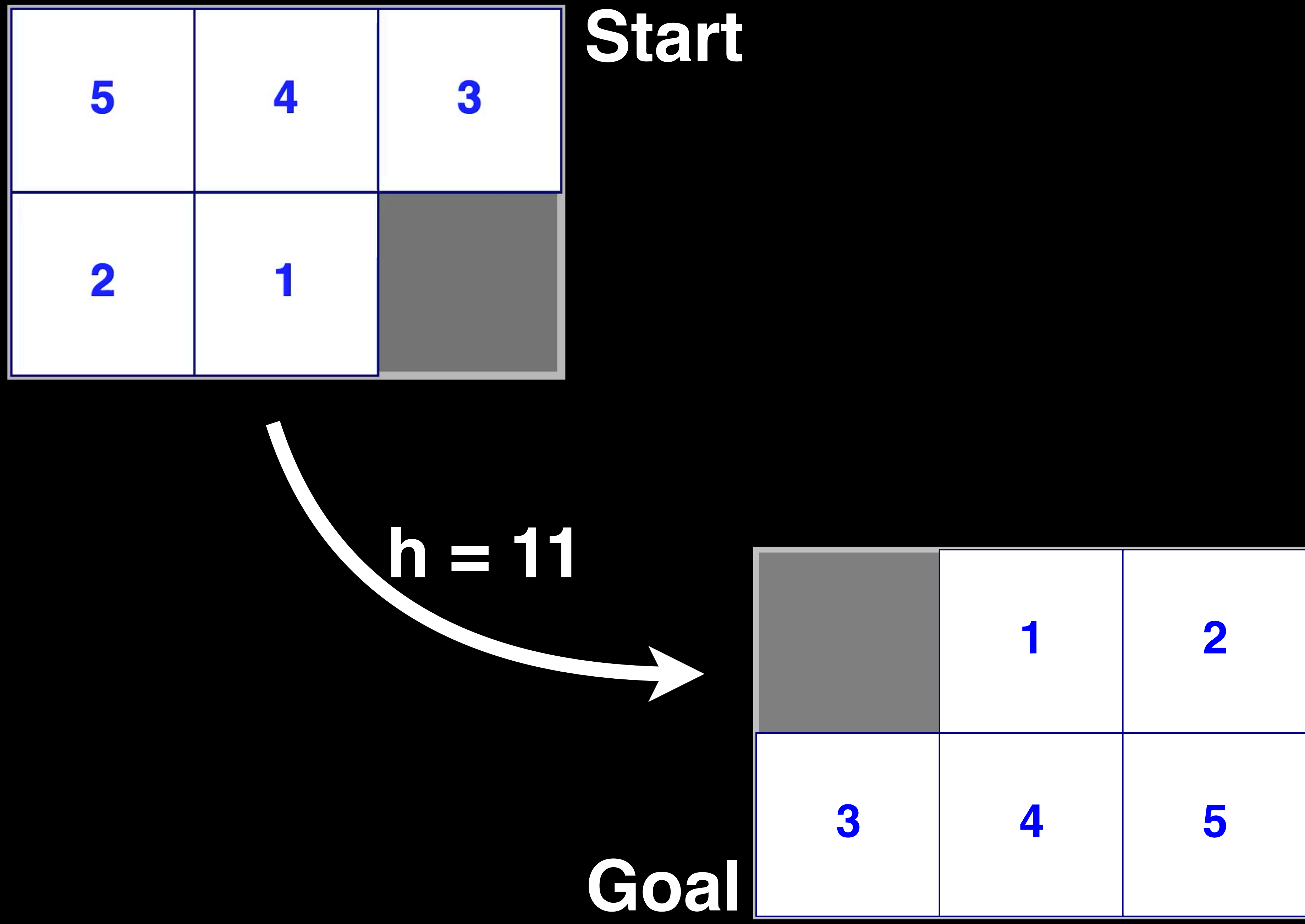
5	4	3
2	1	

Start

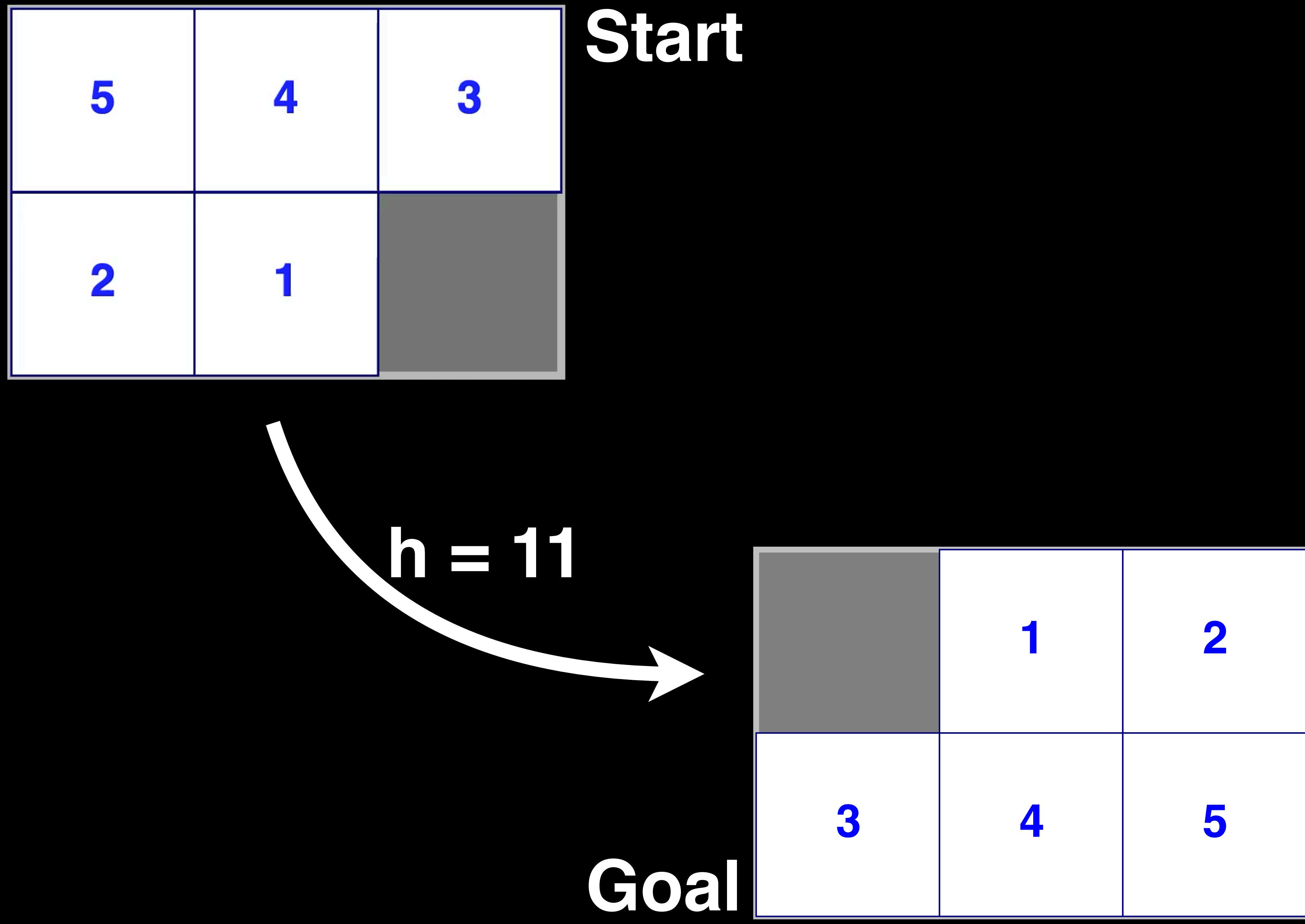
5	4	3
2	1	

Goal

	1	2
3	4	5



Start
5
4
3
2
1
h = 11
1
2
3
4
5
Goal


Start
5
4
3
2
1
h = 11
1
2
3
4
5
Goal

Search Problem

- Given:

Search Problem

- Given:
  - Start state

Search Problem

- Given:
  - Start state
  - Goal state

Search Problem

- Given:
  - Start state
  - Goal state
  - Successor function

Search Problem

- Given:
  - Start state
  - Goal state
  - Successor function
  - Cost function

Search Problem

- Given:
  - Start state
  - Goal state
  - Successor function
  - Cost function

← Defines implicit graph

Search Problem

- Given:
  - Start state
  - Goal state
  - Successor function
  - Cost function
  - Heuristic function

← Defines implicit graph

Search Problem

- Given:
  - Start state
  - Goal state
  - Successor function
  - Cost function

  ← Defines implicit graph
  - Heuristic function
- Find:

Search Problem

- Given:
  - Start state
  - Goal state
  - Successor function
  - Cost function

  ← Defines implicit graph

  - Heuristic function
- Find:
  - Optimal path between start/goal

Why do we need BTS?

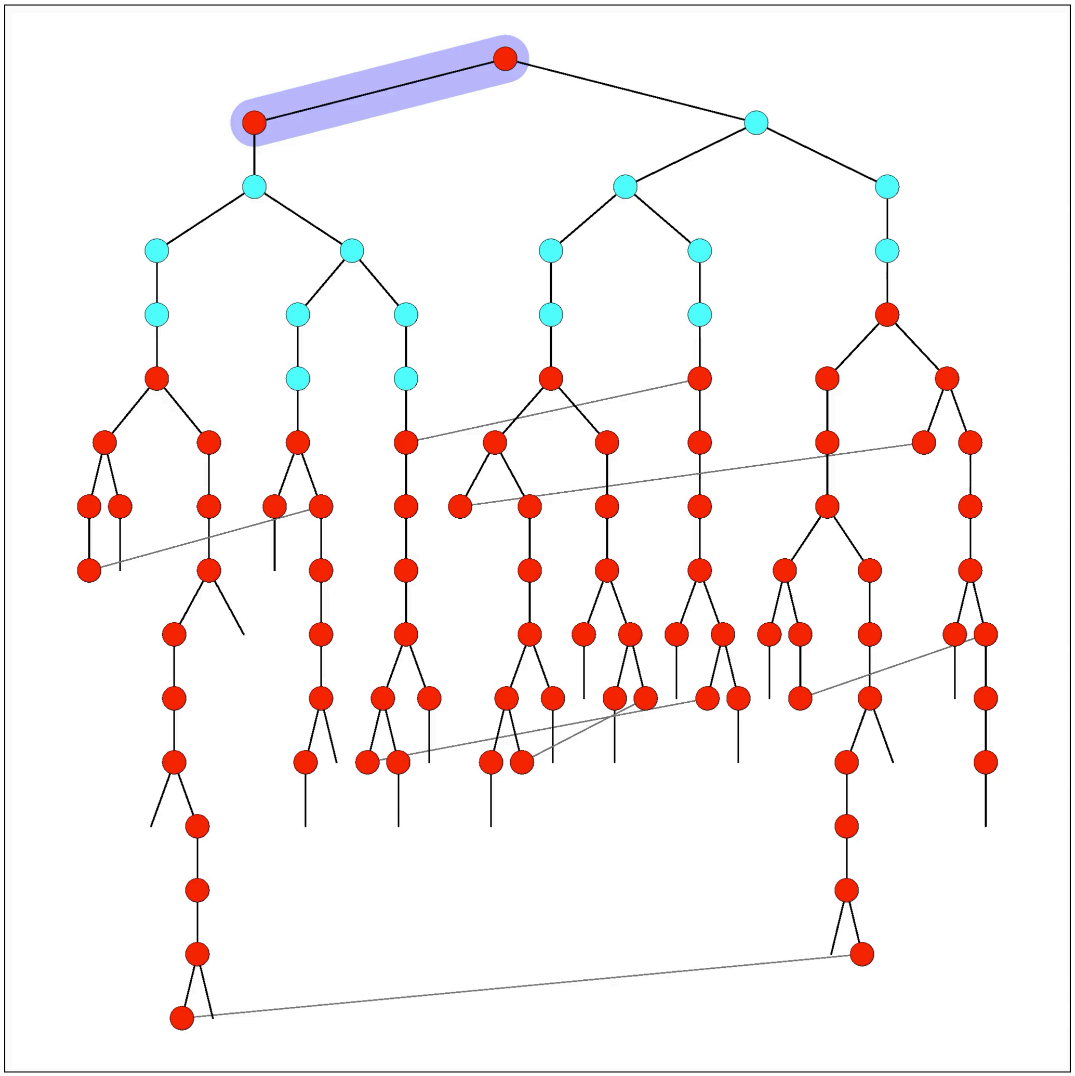
IDA* Refresher

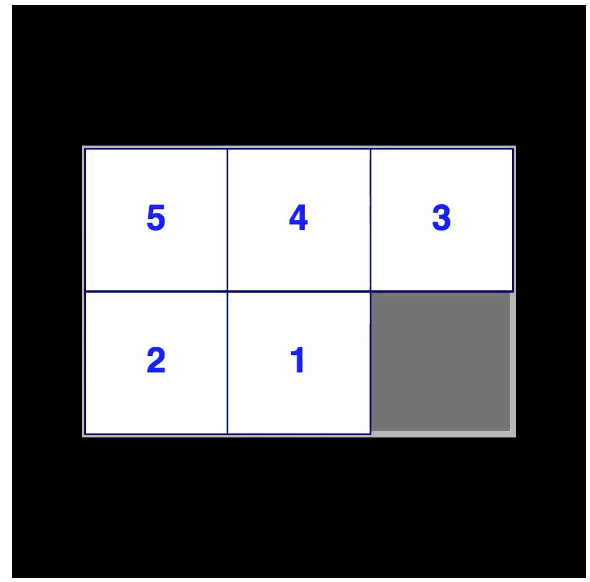
- IDA* does *iterative deepening* search on *f*-costs
  - $f(n) = g(n) + h(n)$

IDA* Refresher

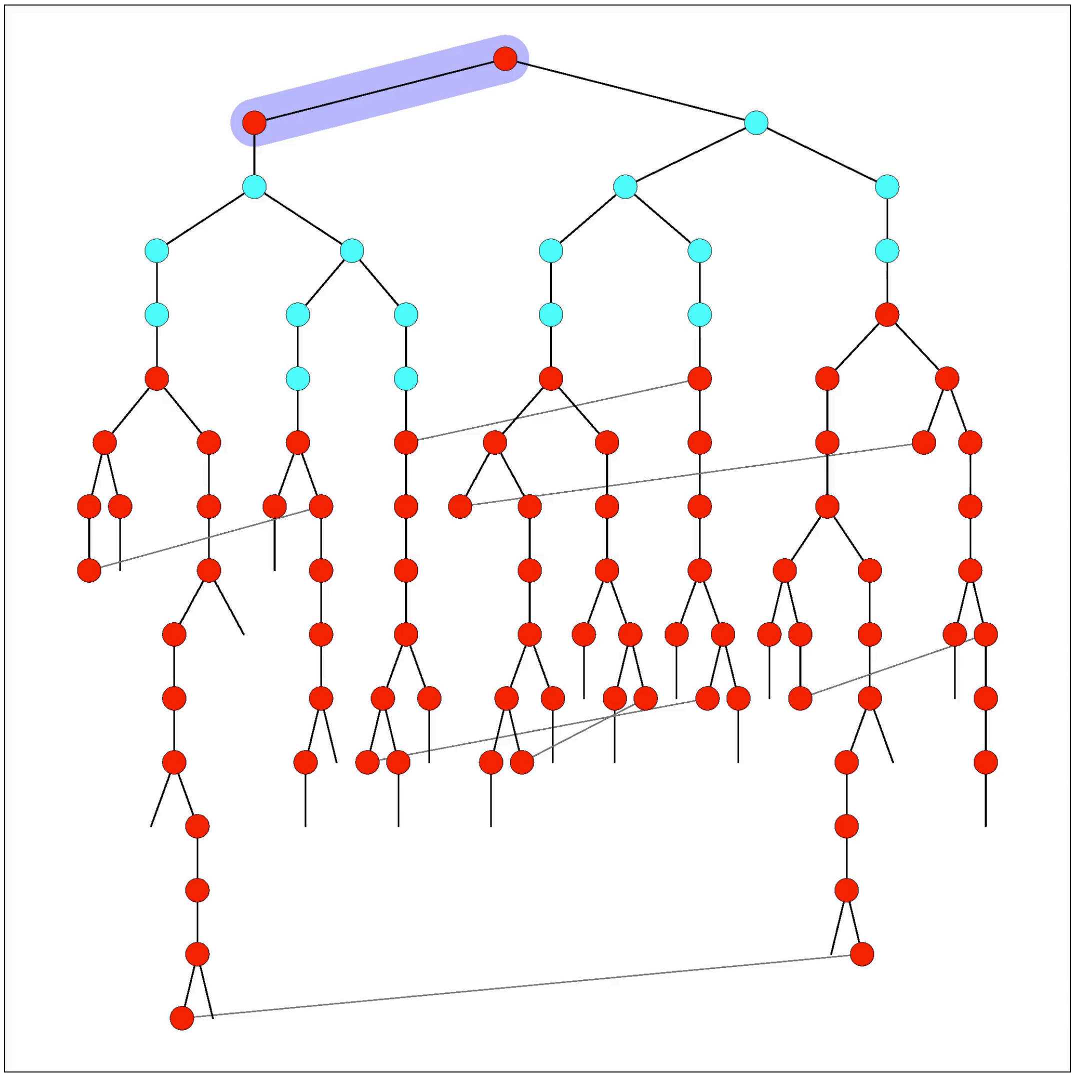
- IDA* does *iterative deepening* search on *f*-costs
  - $f(n) = g(n) + h(n)$
- Next iteration *f*-cost:
  - Smallest unexplored from previous iteration

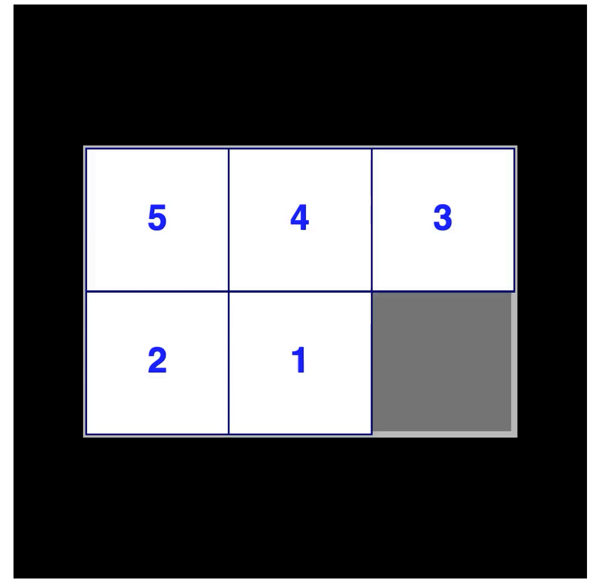
IDA* - Unit Costs



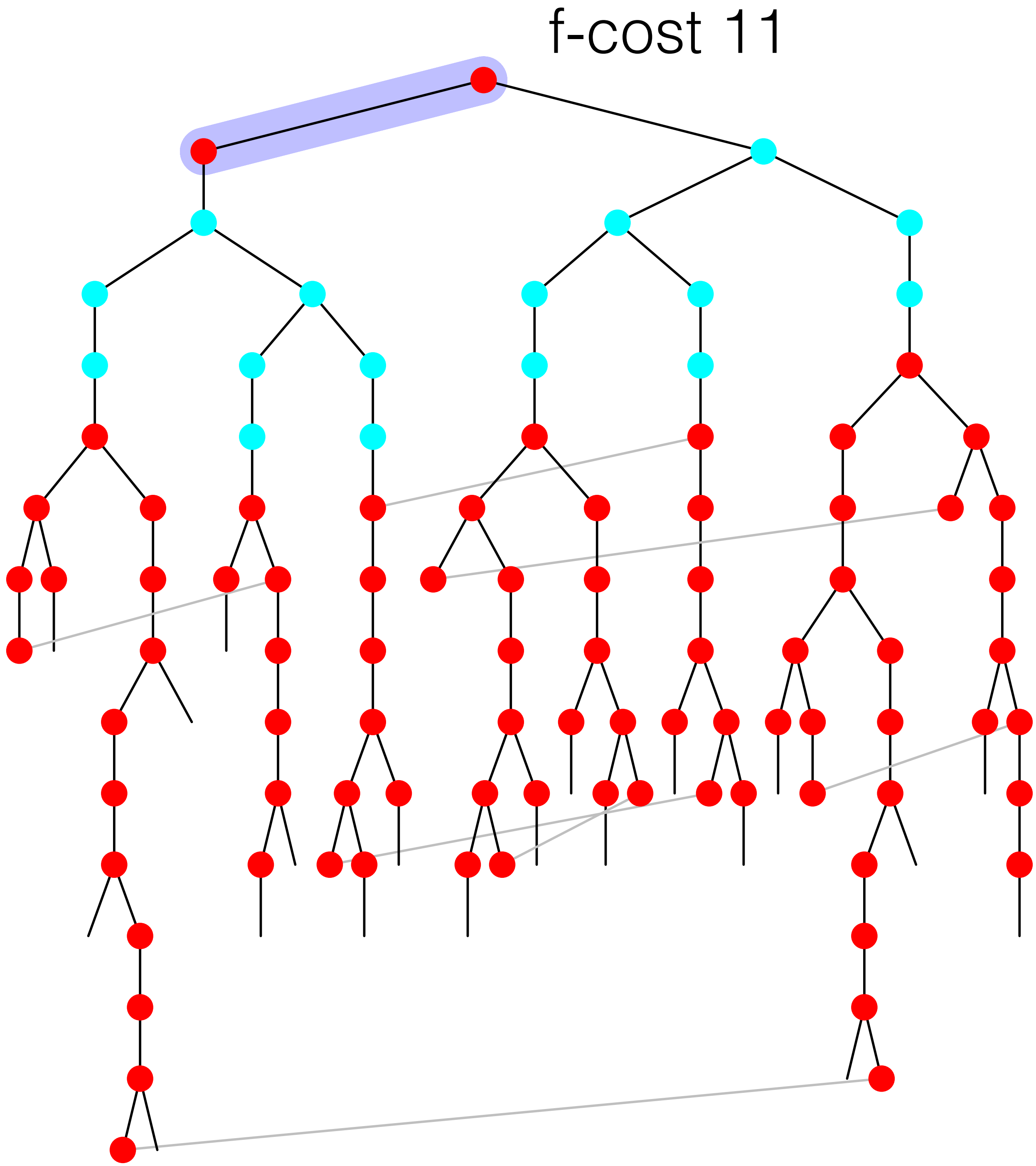


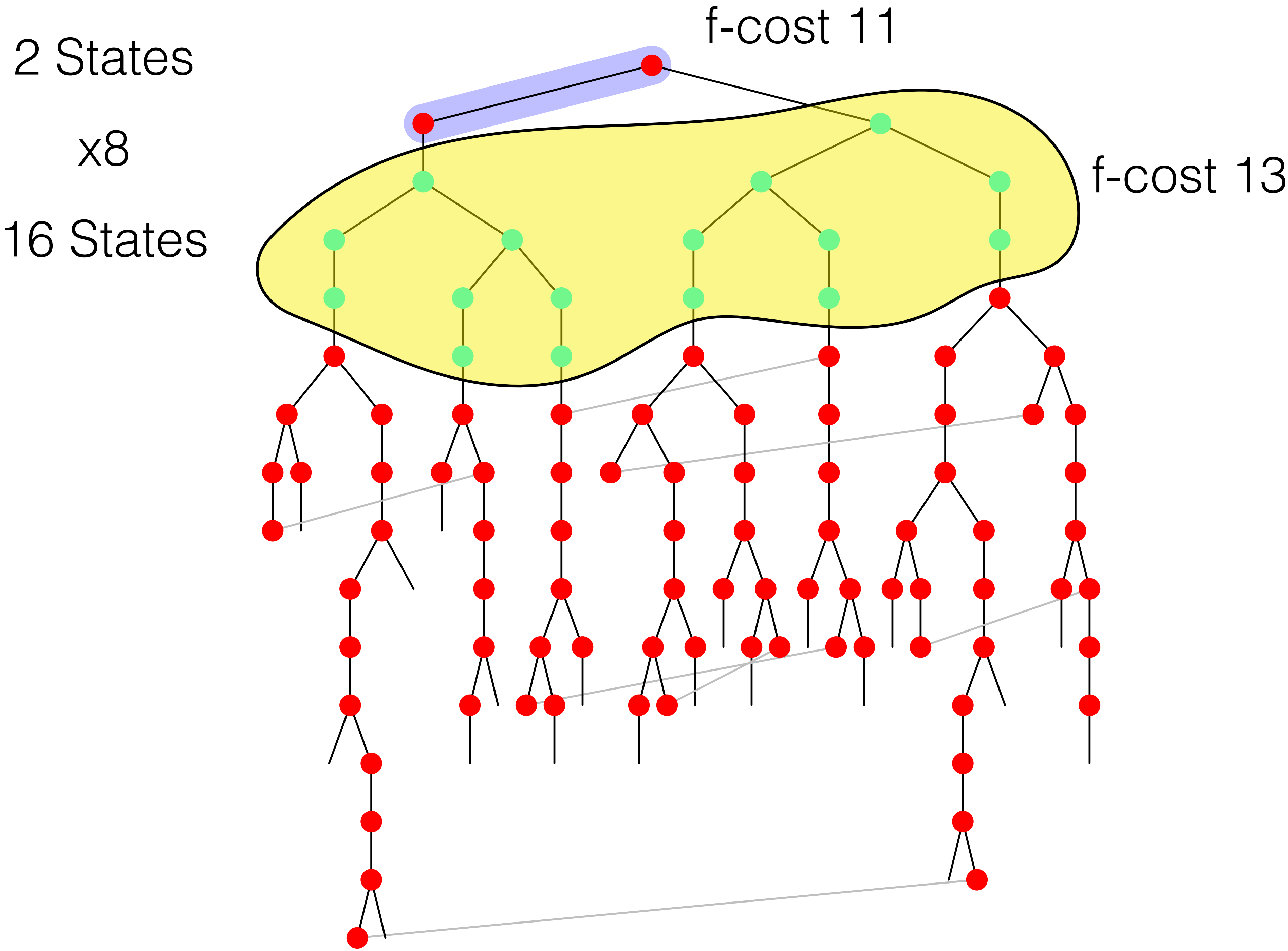
IDA* - Unit Costs

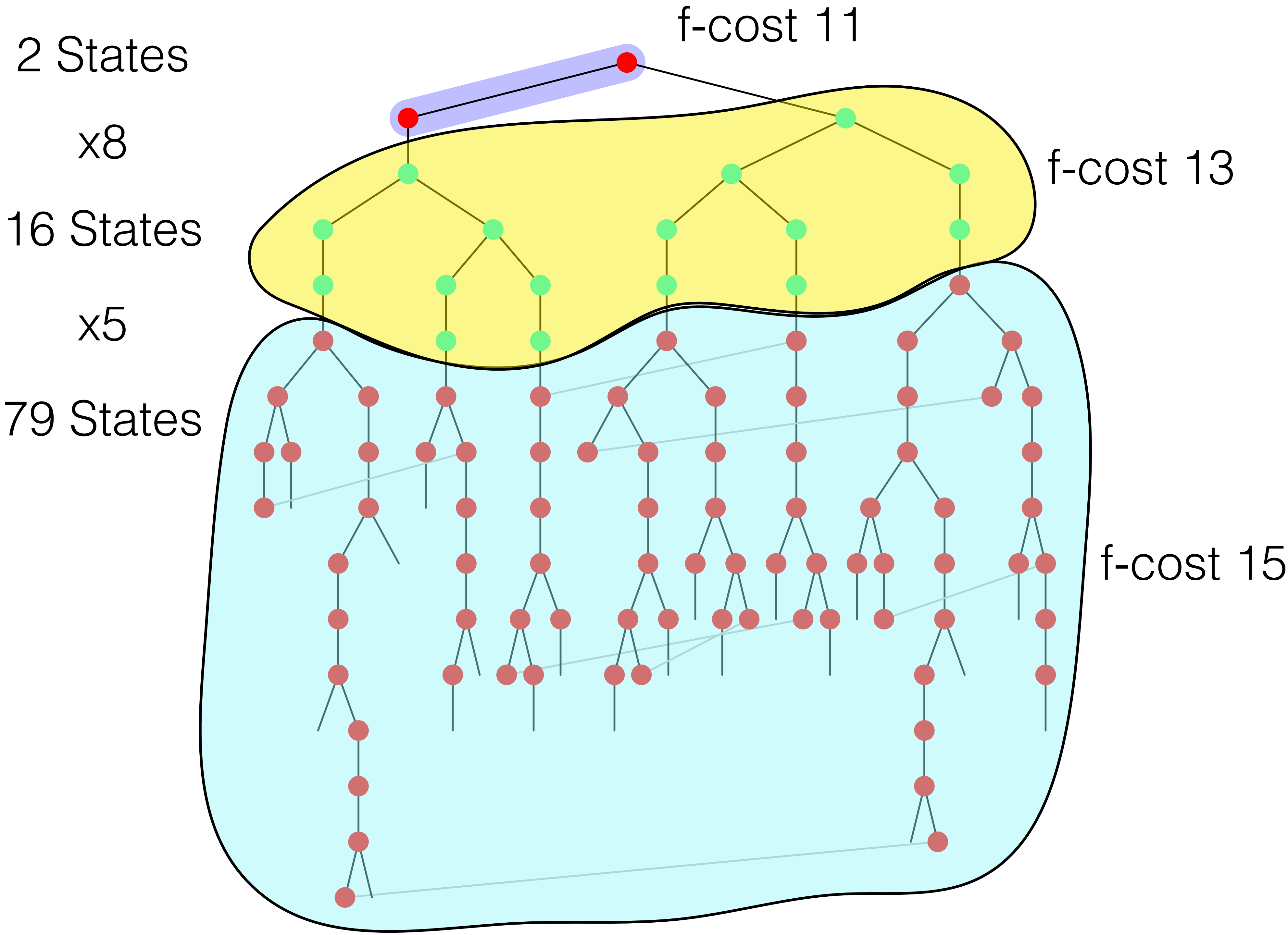




2 States




2 States
x8
16 States
f-cost 11
f-cost 13


f-cost 11
2 States
x8
f-cost 13
16 States
x5
79 States
f-cost 15

IDA* Worst Case

IDA* Worst Case

- *f*-cost layers grow exponentially

IDA* Worst Case

- *f*-cost layers grow exponentially
  - 1

IDA* Worst Case

- *f*-cost layers grow exponentially
  - $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$

IDA* Worst Case

- *f*-cost layers grow exponentially
  - $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
- What if *f*-cost layers grew linearly?

IDA* Worst Case

- *f*-cost layers grow exponentially
  - $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
- What if *f*-cost layers grew linearly?
  - 1

IDA* Worst Case

- *f*-cost layers grow exponentially
  - $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
- What if *f*-cost layers grew linearly?
  - $1 + 2 + 3 + 4 + \ldots + b^d \approx (b^d)^2$

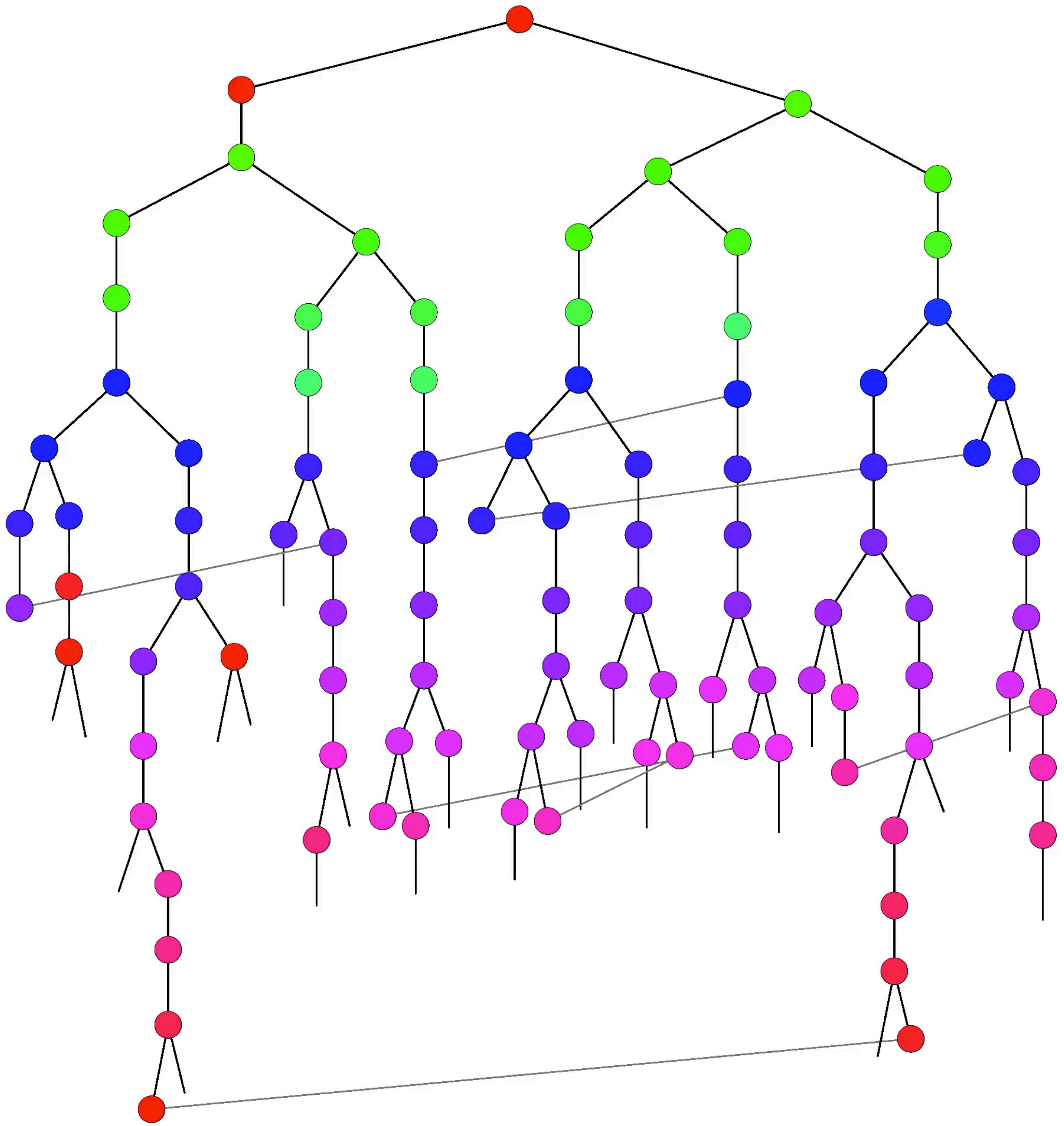
IDA* Worst Case

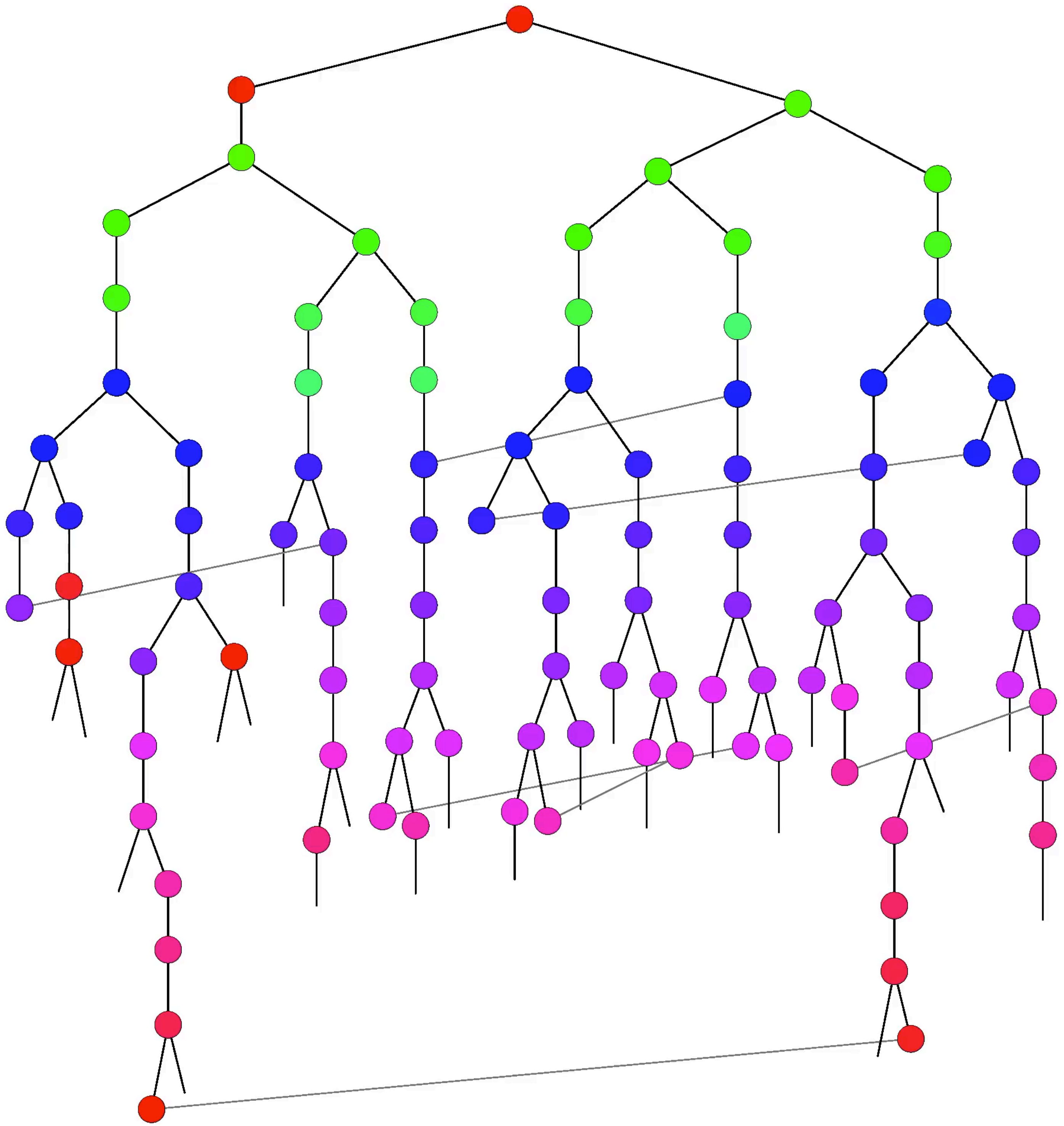
- *f*-cost layers grow exponentially
  - $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
- What if *f*-cost layers grew linearly?
  - $1 + 2 + 3 + 4 + \ldots + b^d \approx (b^d)^2$
- Happens with non-unit edge costs:
  - STP: Cost of moving tile t: $\dfrac{t+2}{t+1}$

IDA* Worst Case

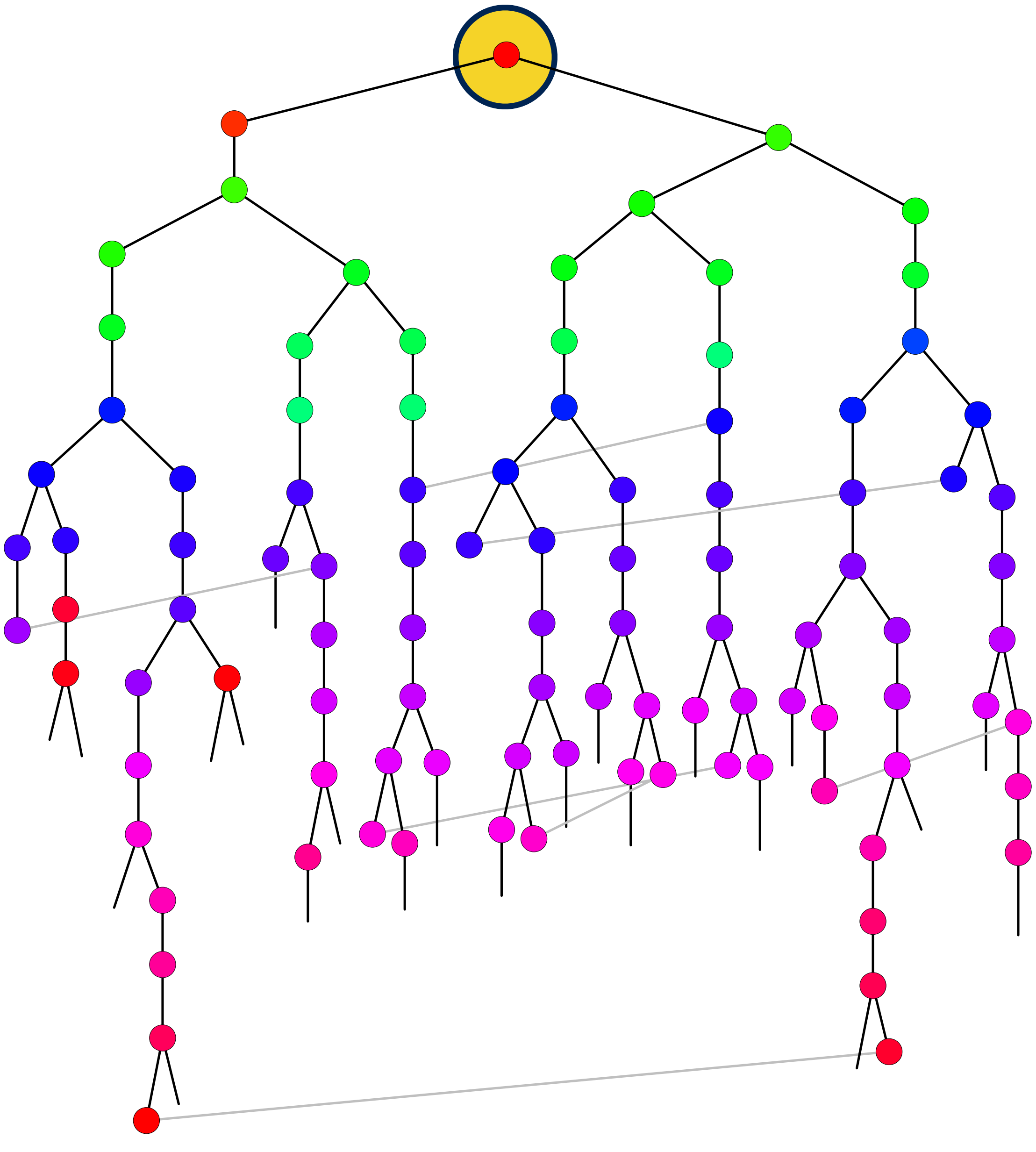
- *f*-cost layers grow exponentially
  - $1 + b + b^2 + b^3 + \ldots + b^d \approx b^d$
- What if *f*-cost layers grew linearly?
  - $1 + 2 + 3 + 4 + \ldots + b^d \approx (b^d)^2$
- Happens with non-unit edge costs:
  - STP: Cost of moving tile t: $\dfrac{t+2}{t+1}$

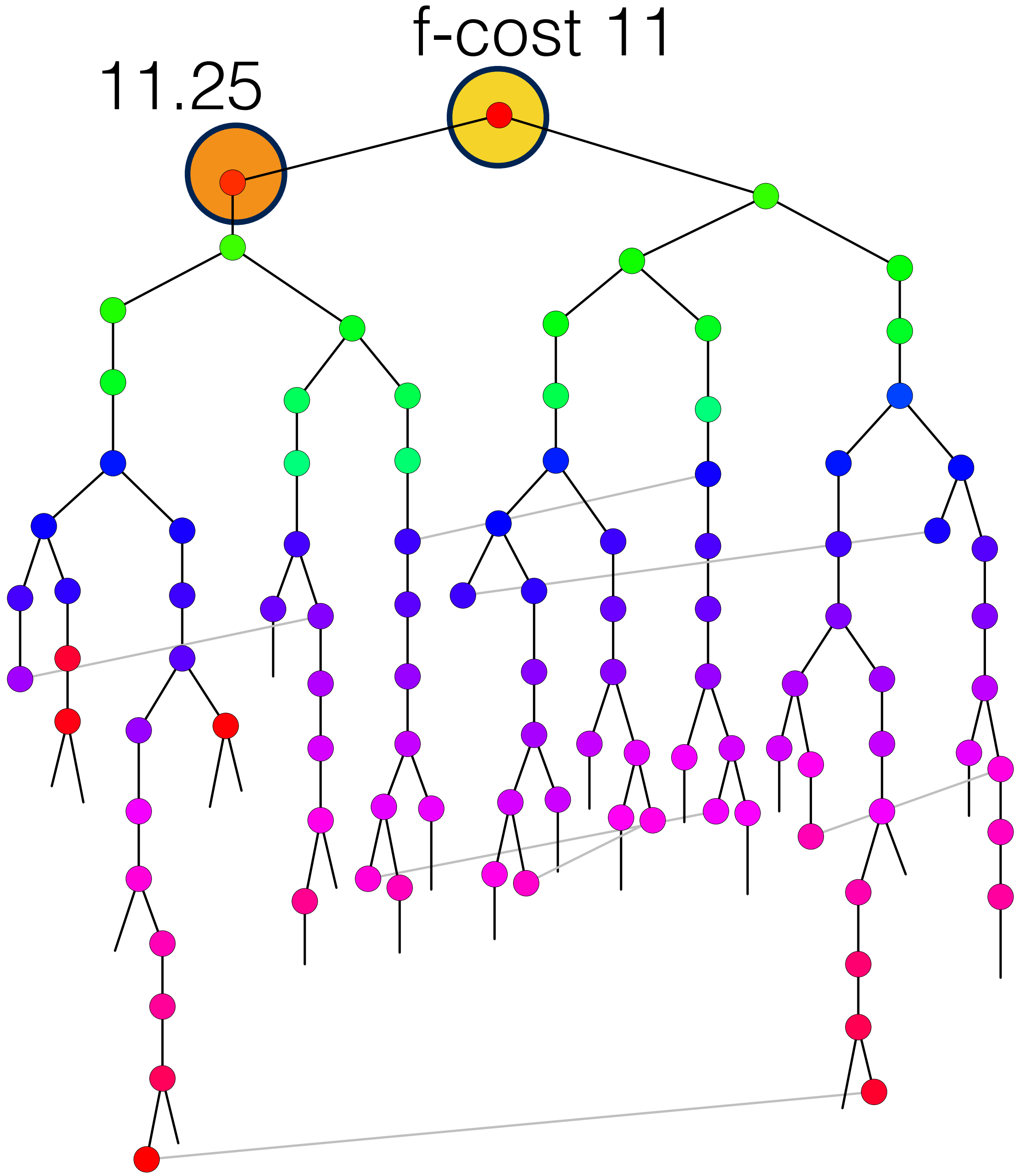
Tile	Cost
1	$\frac{1+2}{1+1} = 1.5$
3	$\frac{3+2}{3+1} = 1.25$
7	$\frac{7+2}{7+1} = 1.125$
9	$\frac{9+2}{9+1} = 1.1$



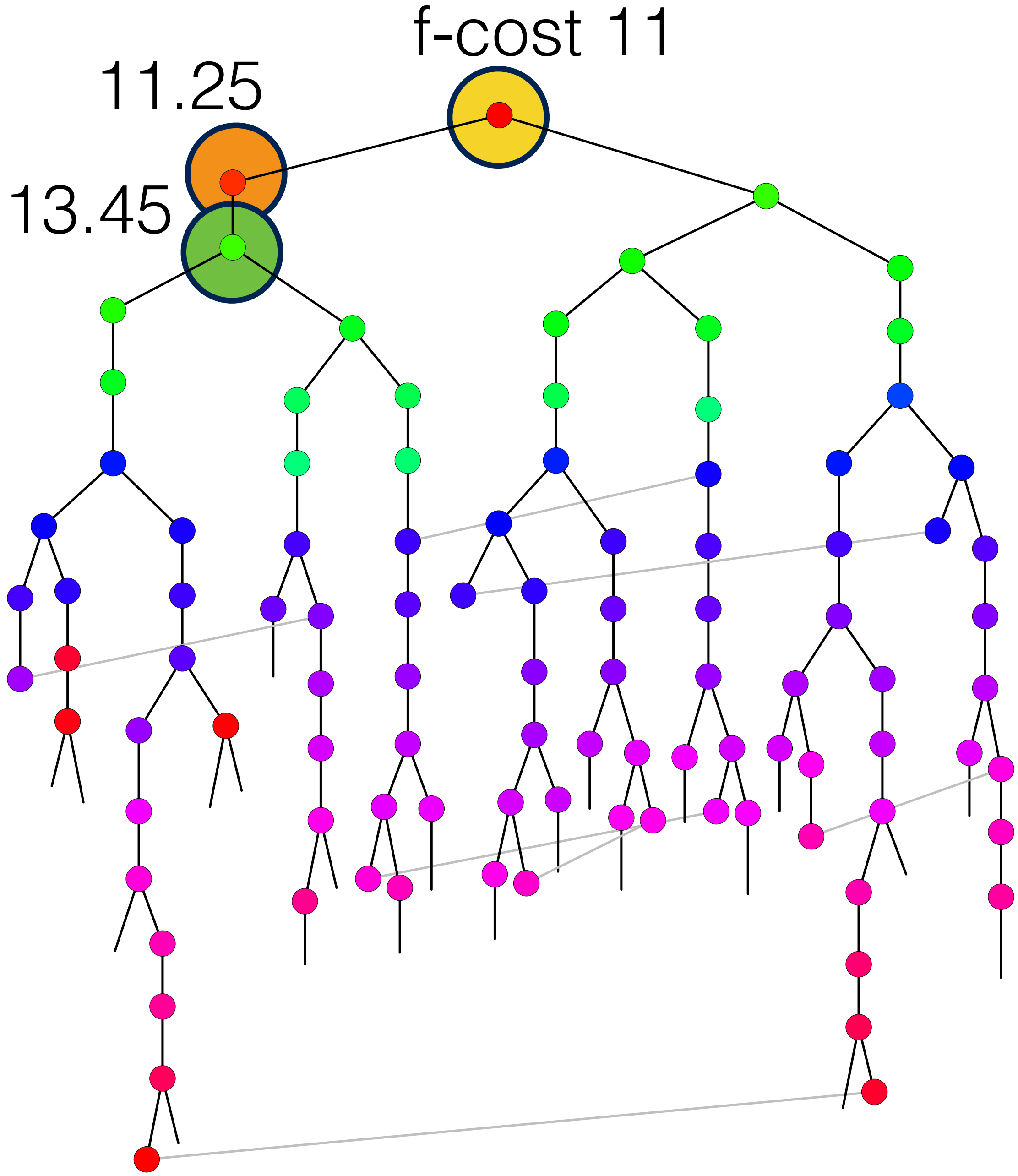


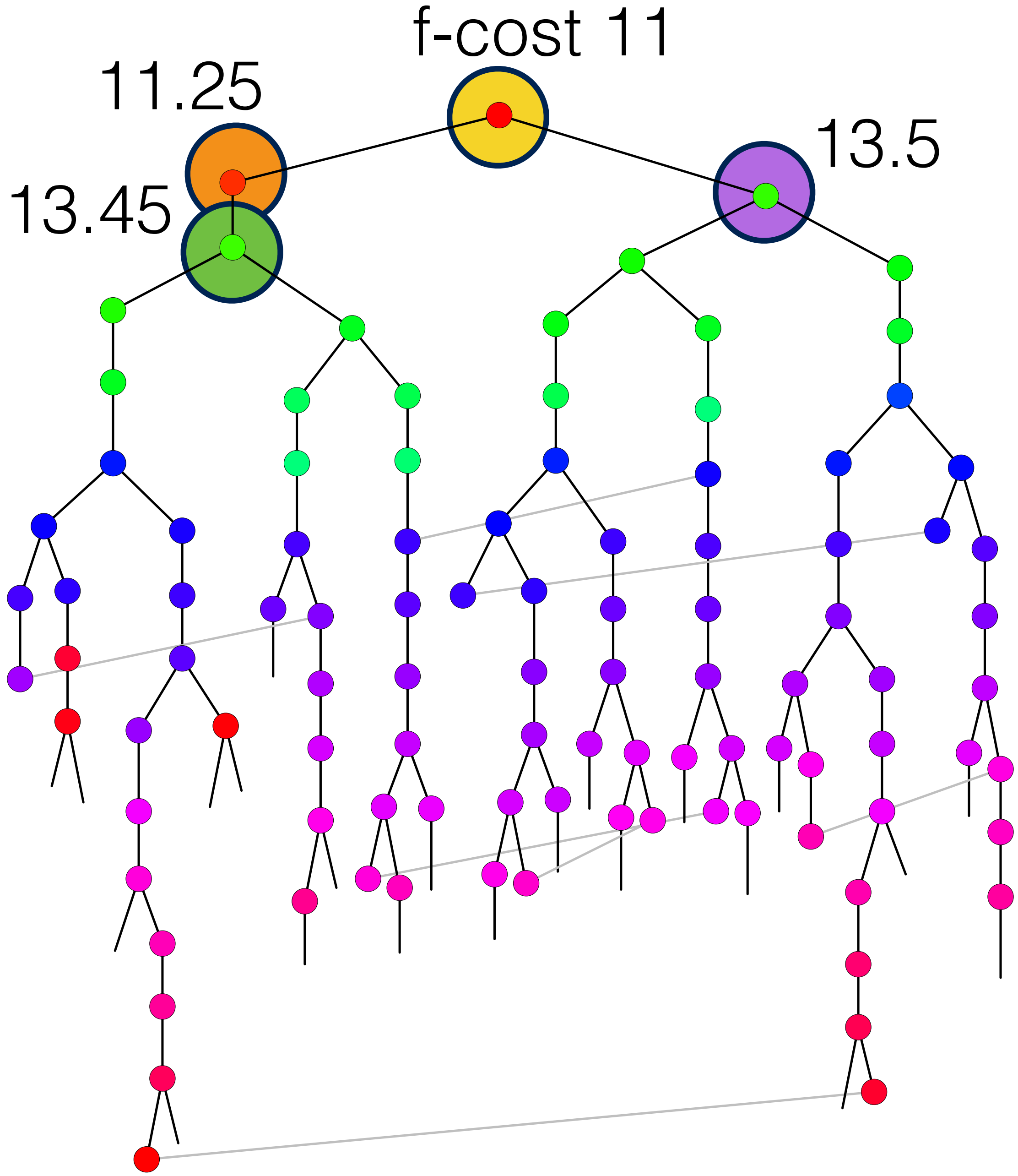
f-cost 11



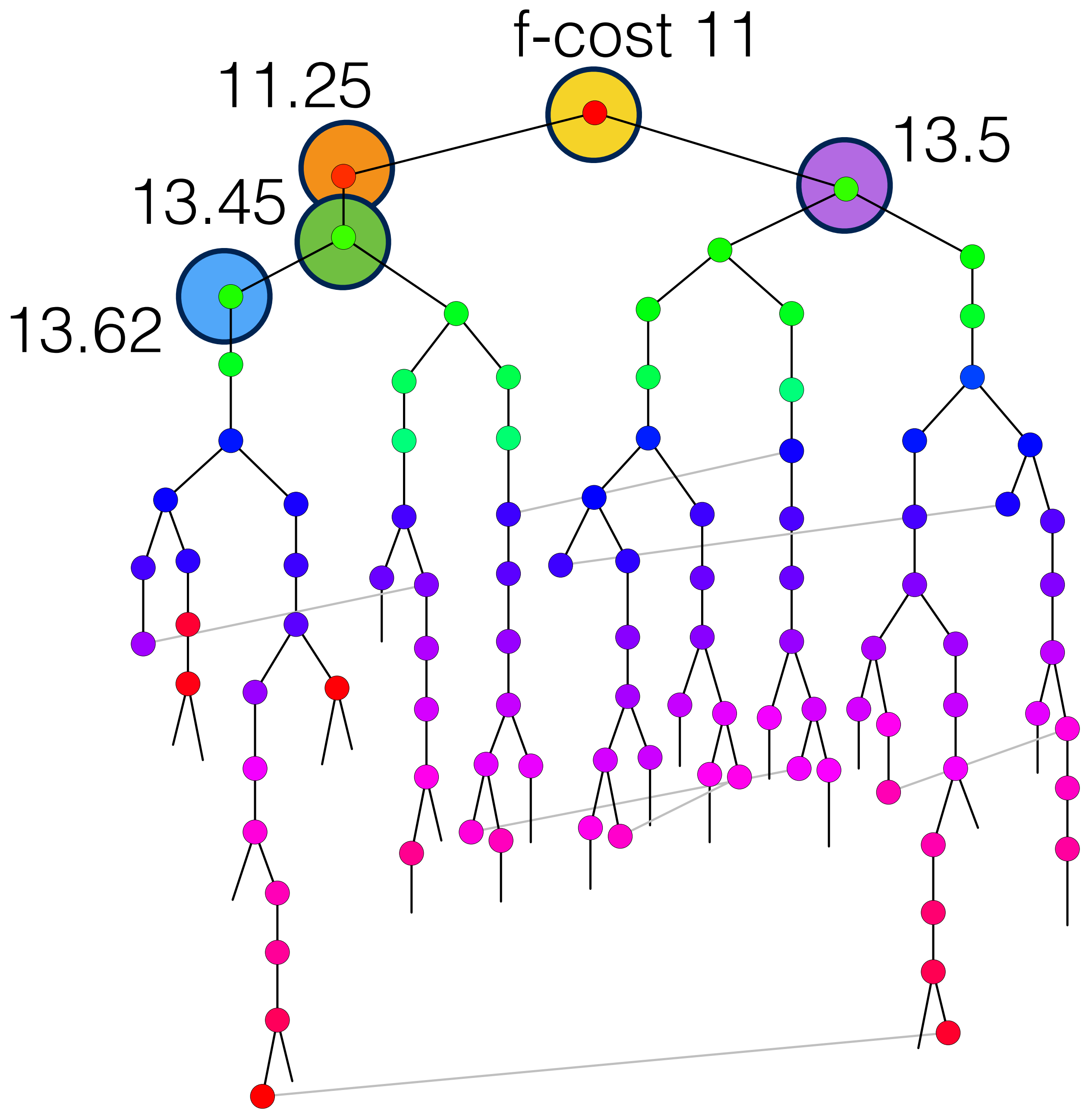


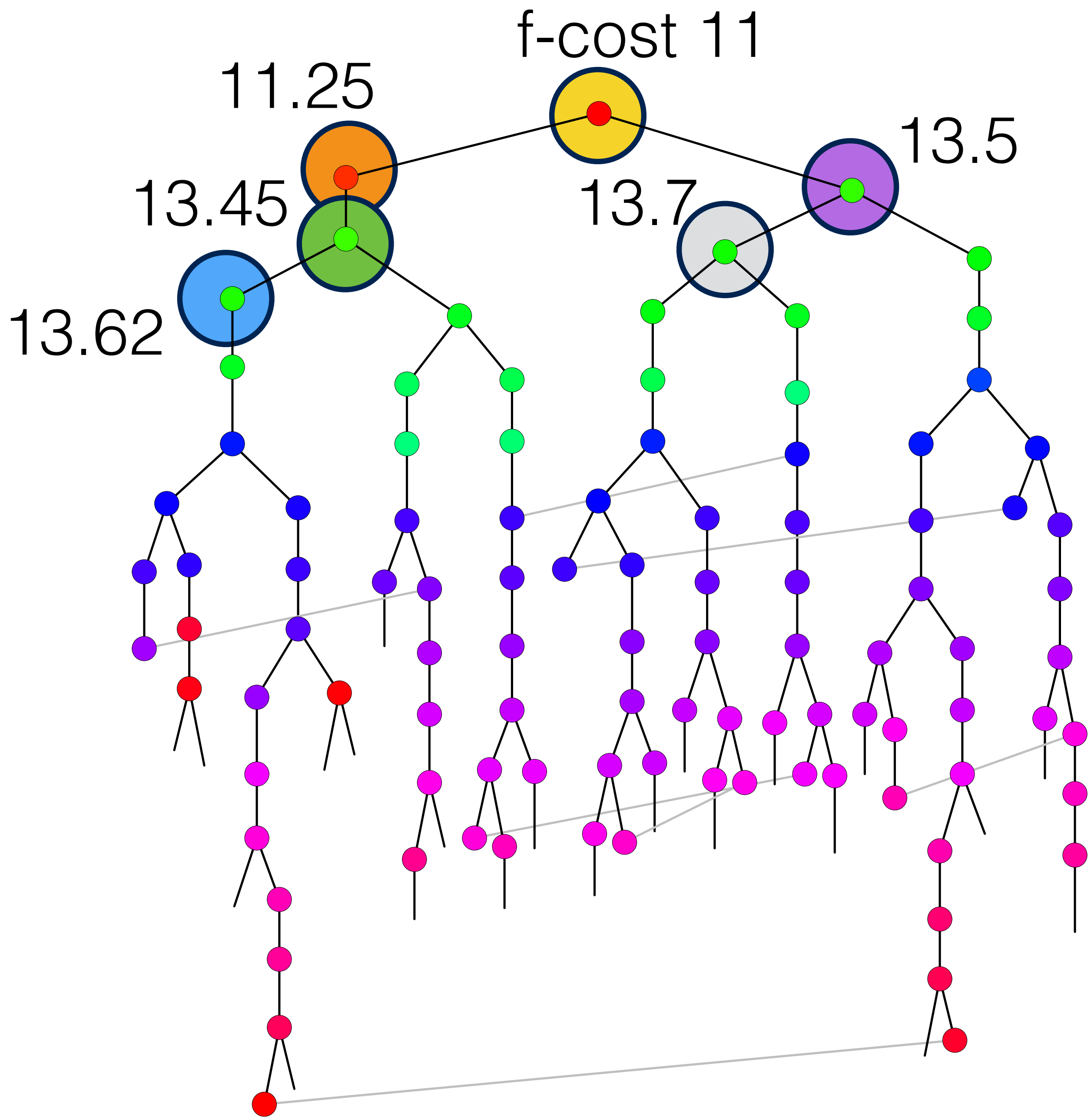
f-cost 11
11.25

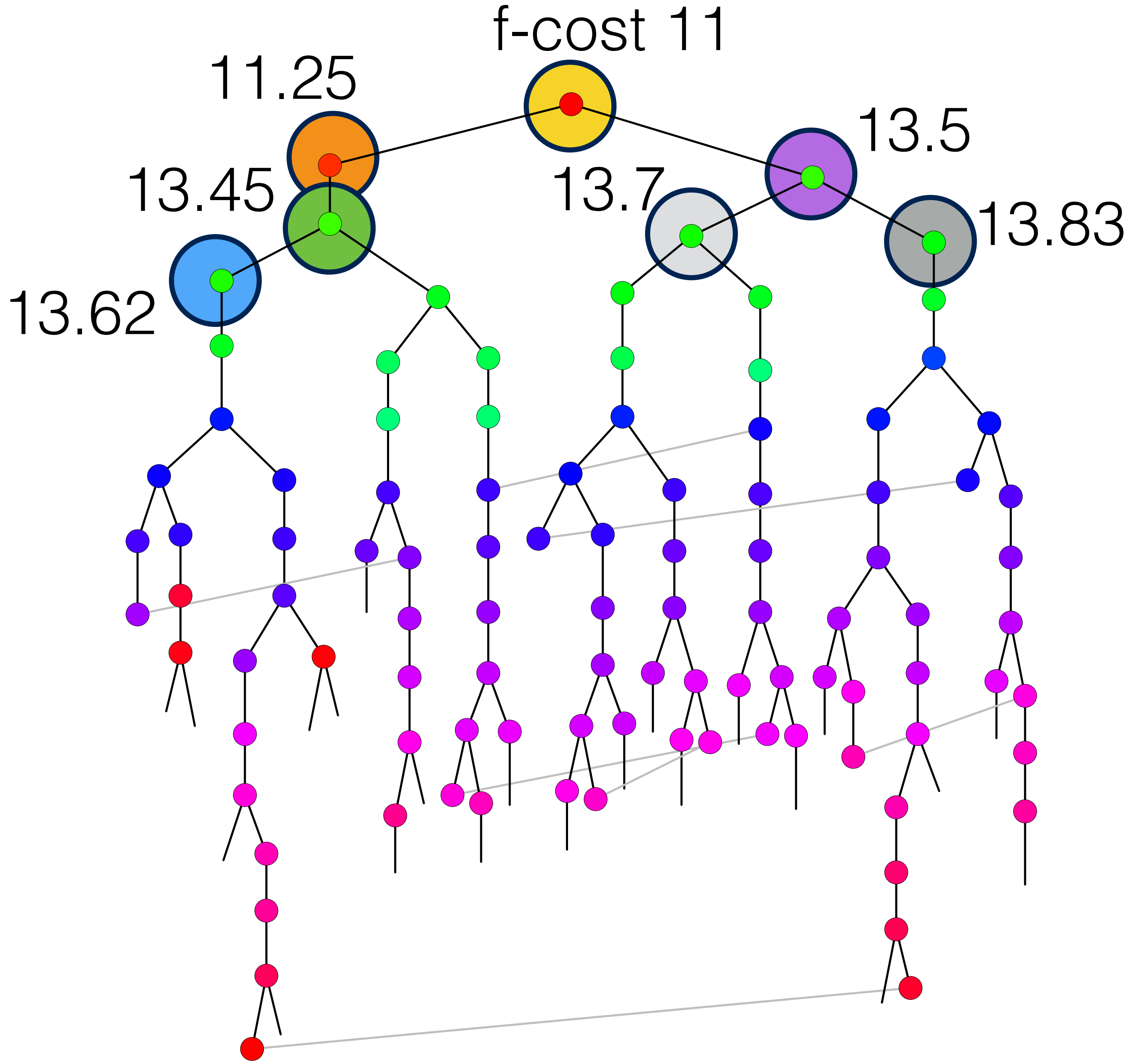

f-cost 11
11.25
13.45

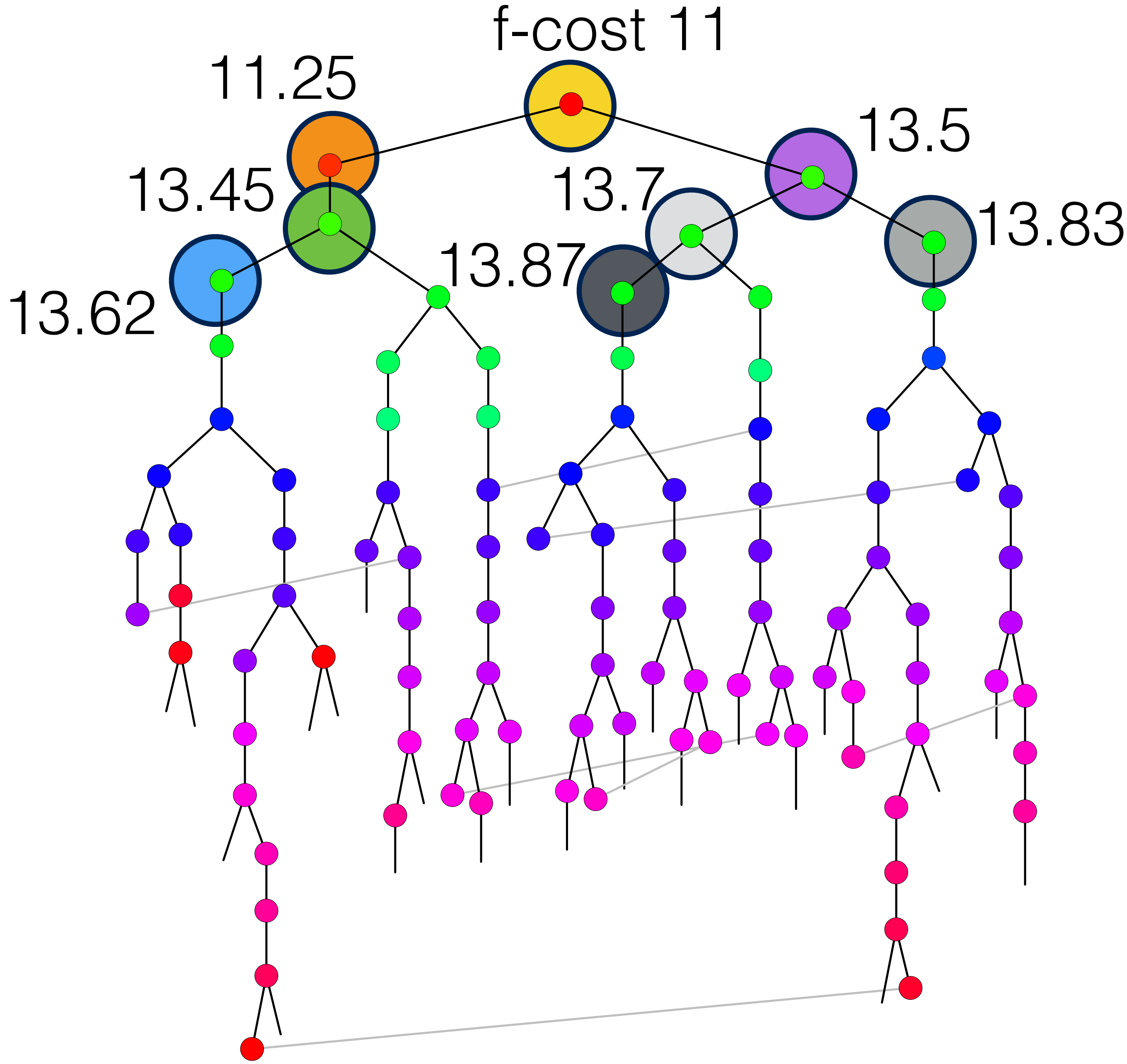


f-cost 11
11.25
13.45
13.5


f-cost 11
11.25
13.5
13.45
13.62


f-cost 11
11.25
13.5
13.45
13.7
13.62


f-cost 11
11.25
13.5
13.45
13.7
13.83
13.62


f-cost 11
11.25
13.5
13.45
13.7
13.83
13.87
13.62

Why do we need BTS?

If the nodes in each iteration
do not grow exponentially

Getting the next bound

Getting the next bound

- Can be conservative:
  - IDA* (Korf, 1985)

Getting the next bound

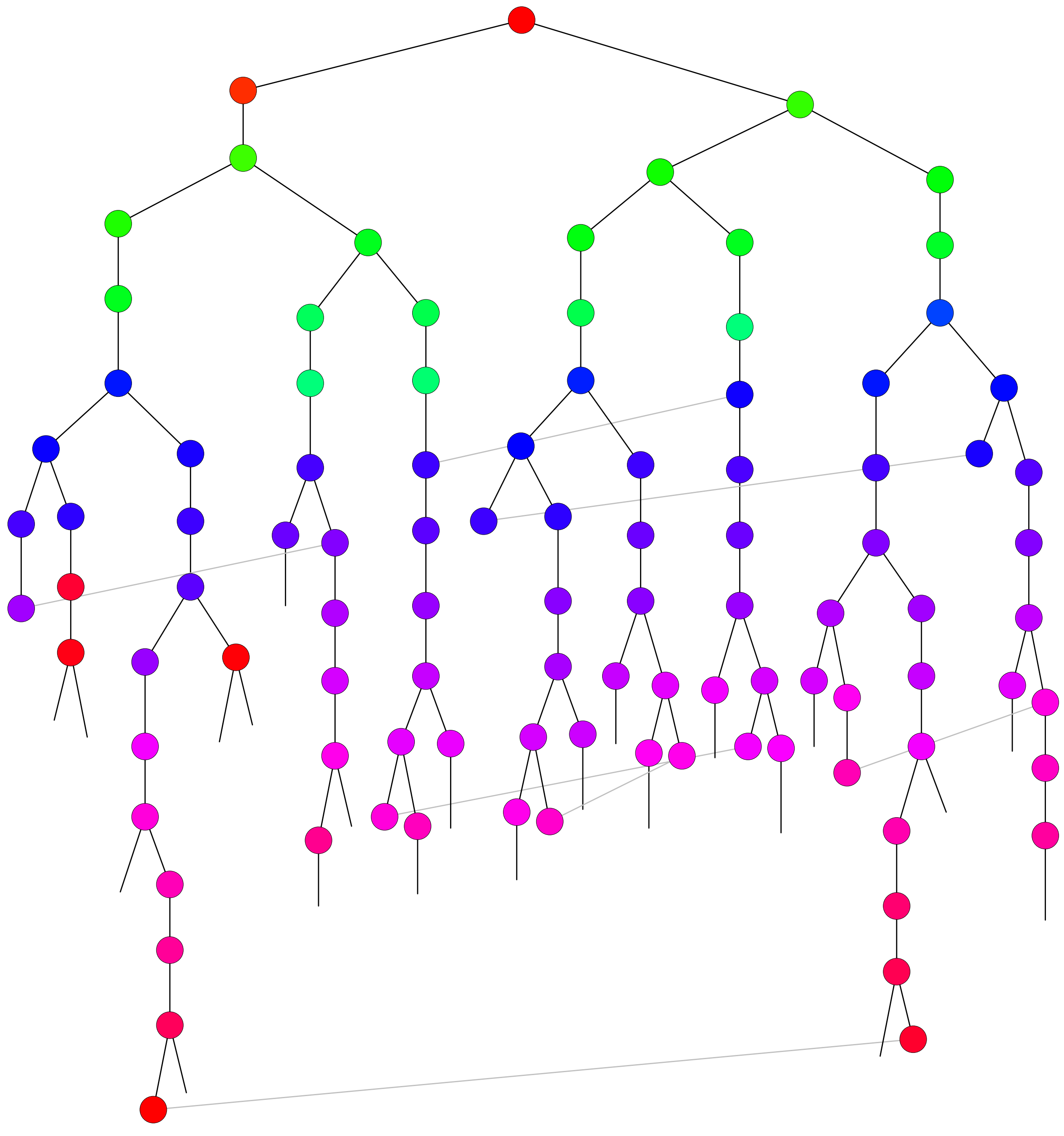
- Can be conservative:
  - IDA* (Korf, 1985)
- Can try to build a predictor based on past:
  - IDA^*_{CR} (Sarkar et al, 1990)
  - IDA^*_{IM} (Burns & Ruml, 2013)

Getting the next bound

- Can be conservative:
  - IDA* (Korf, 1985)
- Can try to build a predictor based on past:
  - IDA*_{CR} (Sarkar et al, 1990)
  - IDA*_{IM} (Burns & Ruml, 2013)
- Can model the state space growth:
  - EDA* (Sharon et al, 2014)

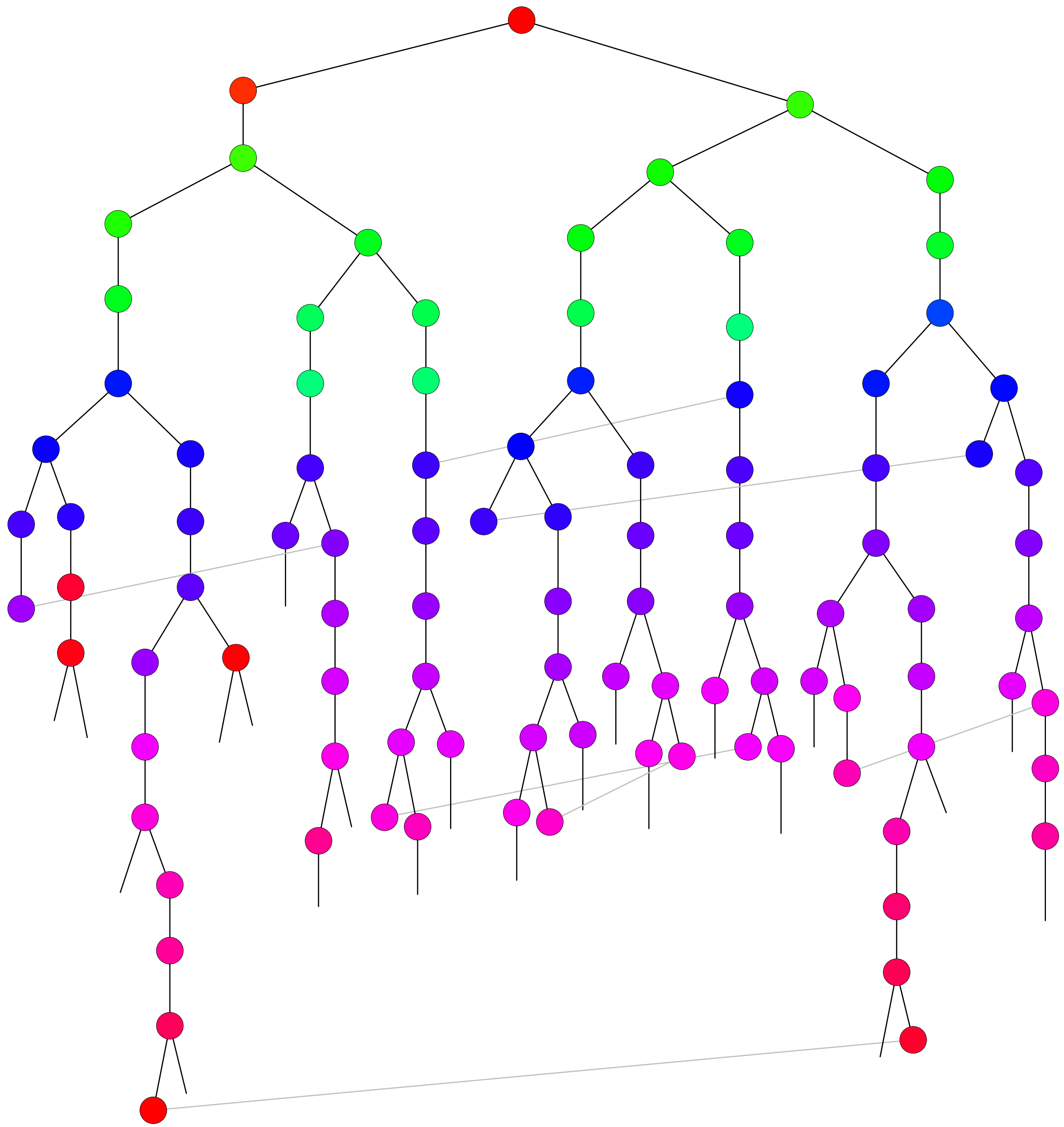
Getting the next bound

- Can be conservative:
  - IDA* (Korf, 1985)
- Can try to build a predictor based on past:
  - IDA^*_{CR} (Sarkar et al, 1990)
  - IDA^*_{IM} (Burns & Ruml, 2013)
- Can model the state space growth:
  - EDA* (Sharon et al, 2014)
- Want to guarantee exponential growth in expansions



$f = 11$

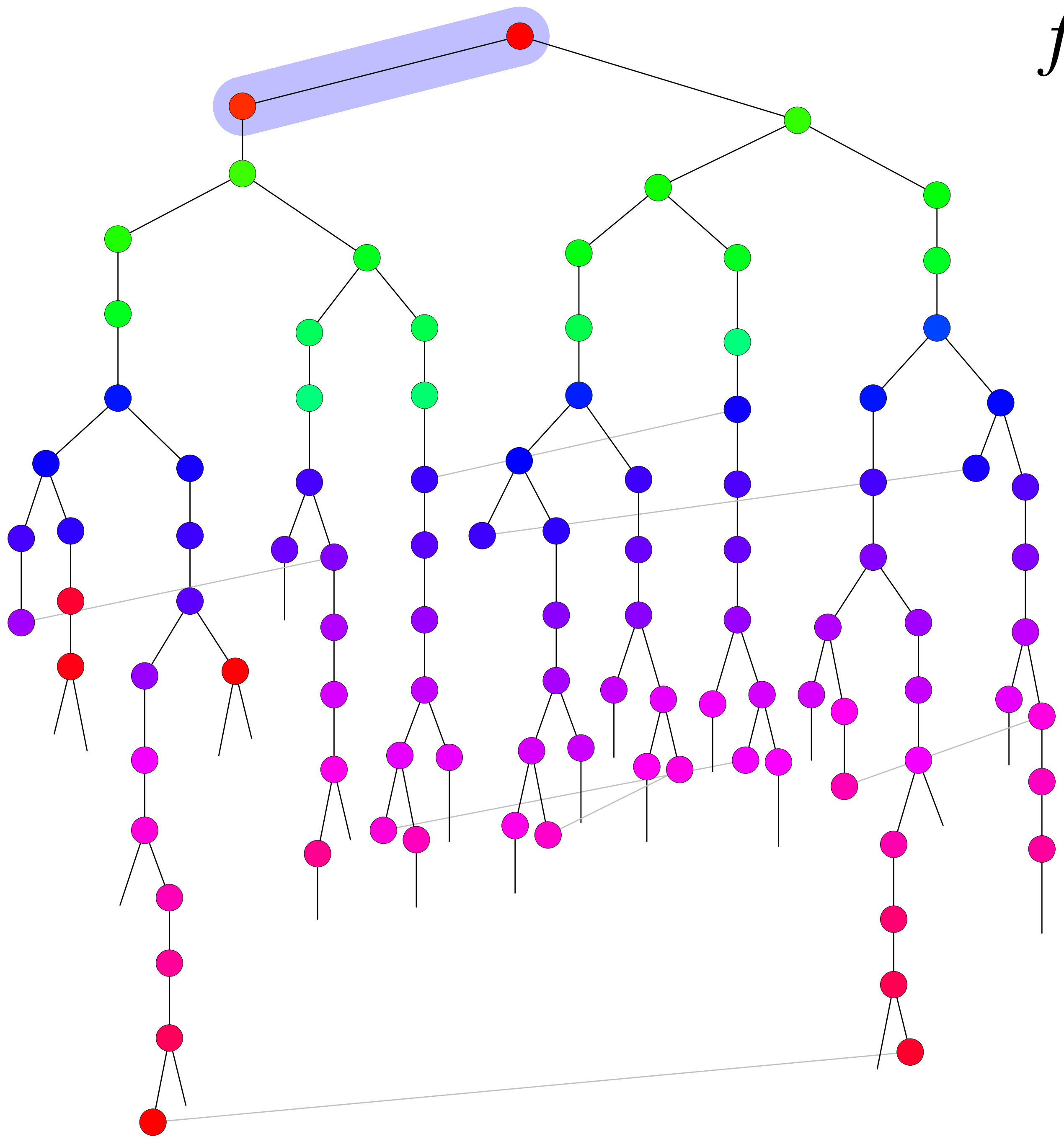
1 node



$f = 11$

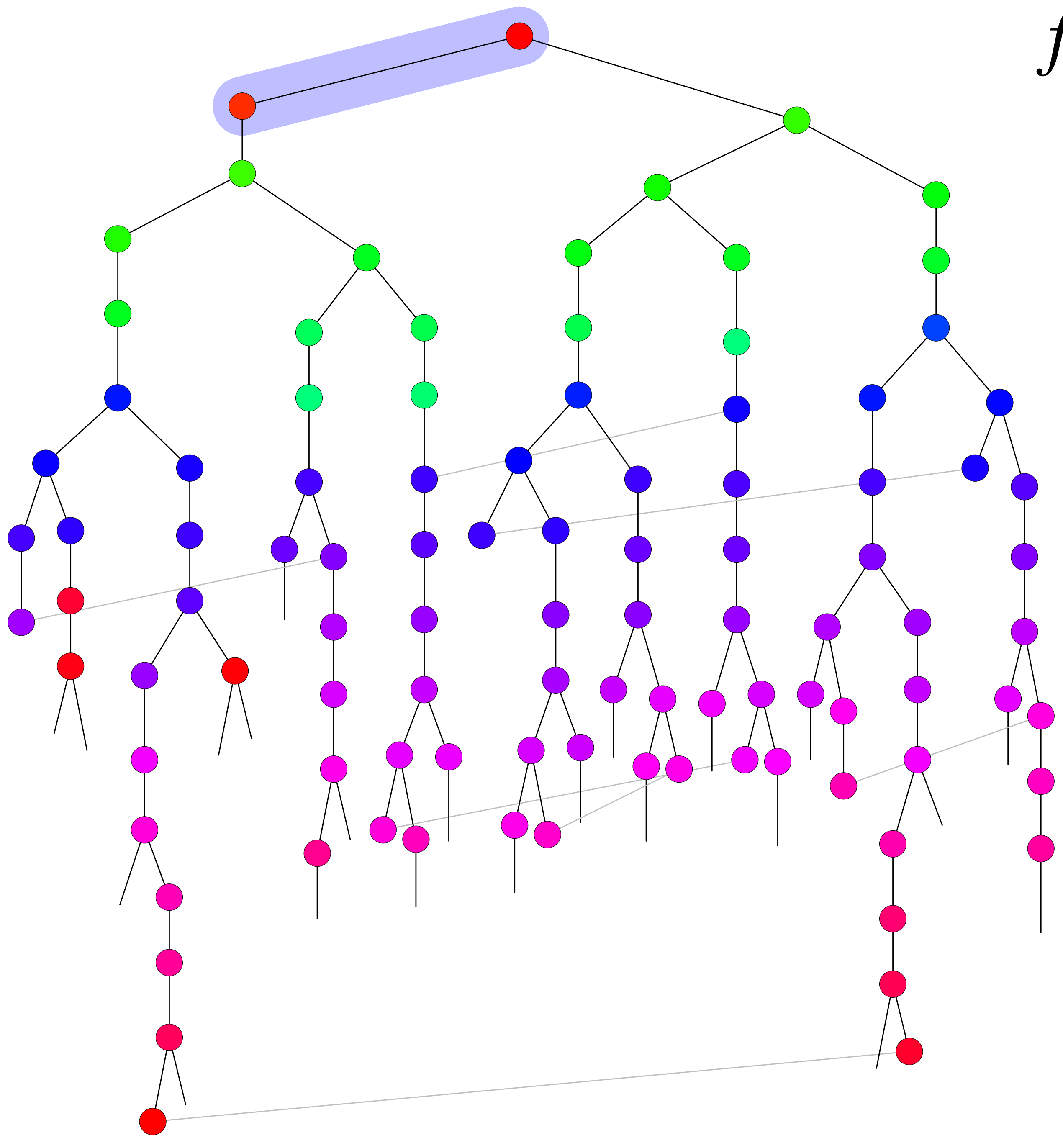
1 node

[2, 8]



$f = 11.25$

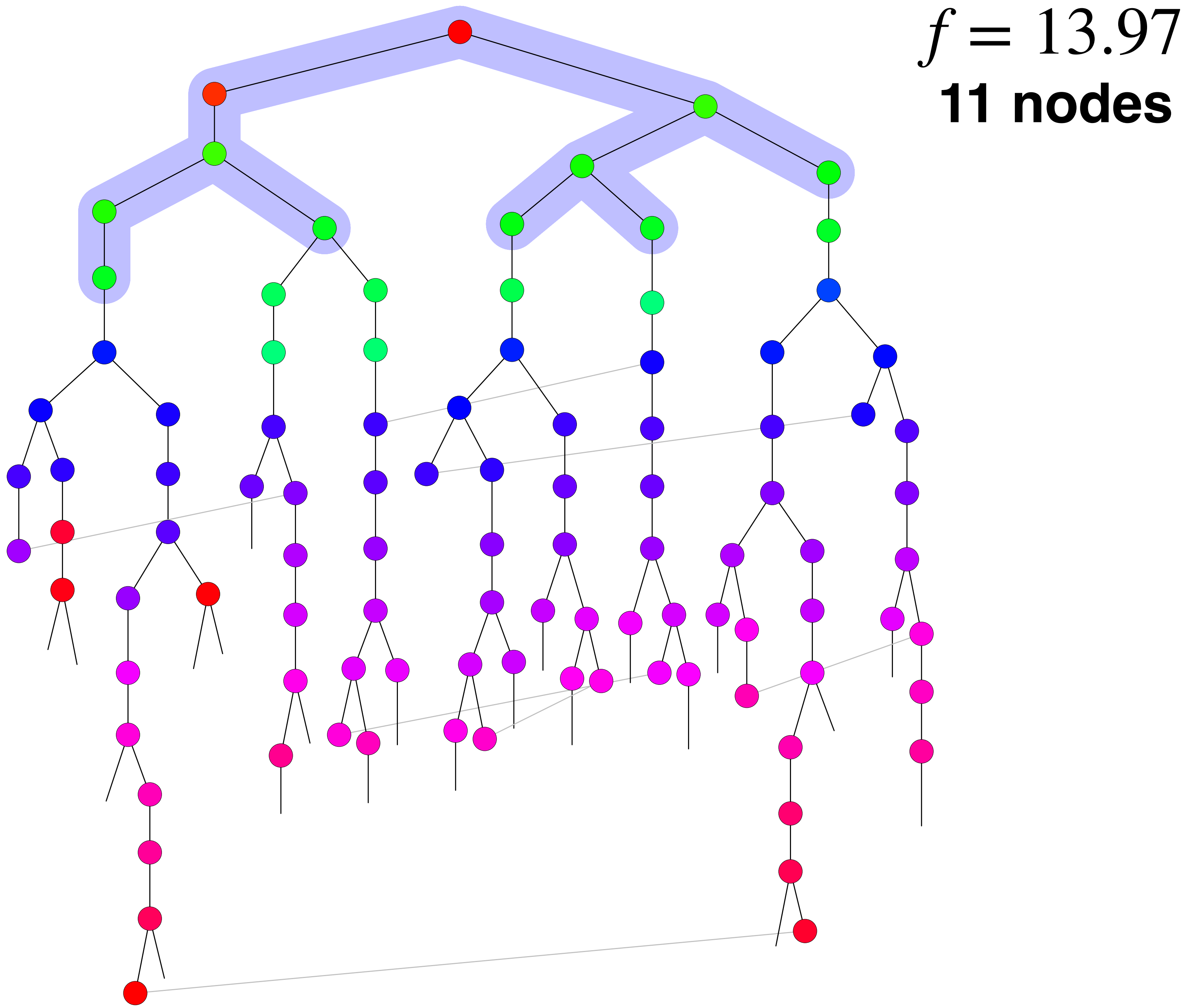
2 nodes

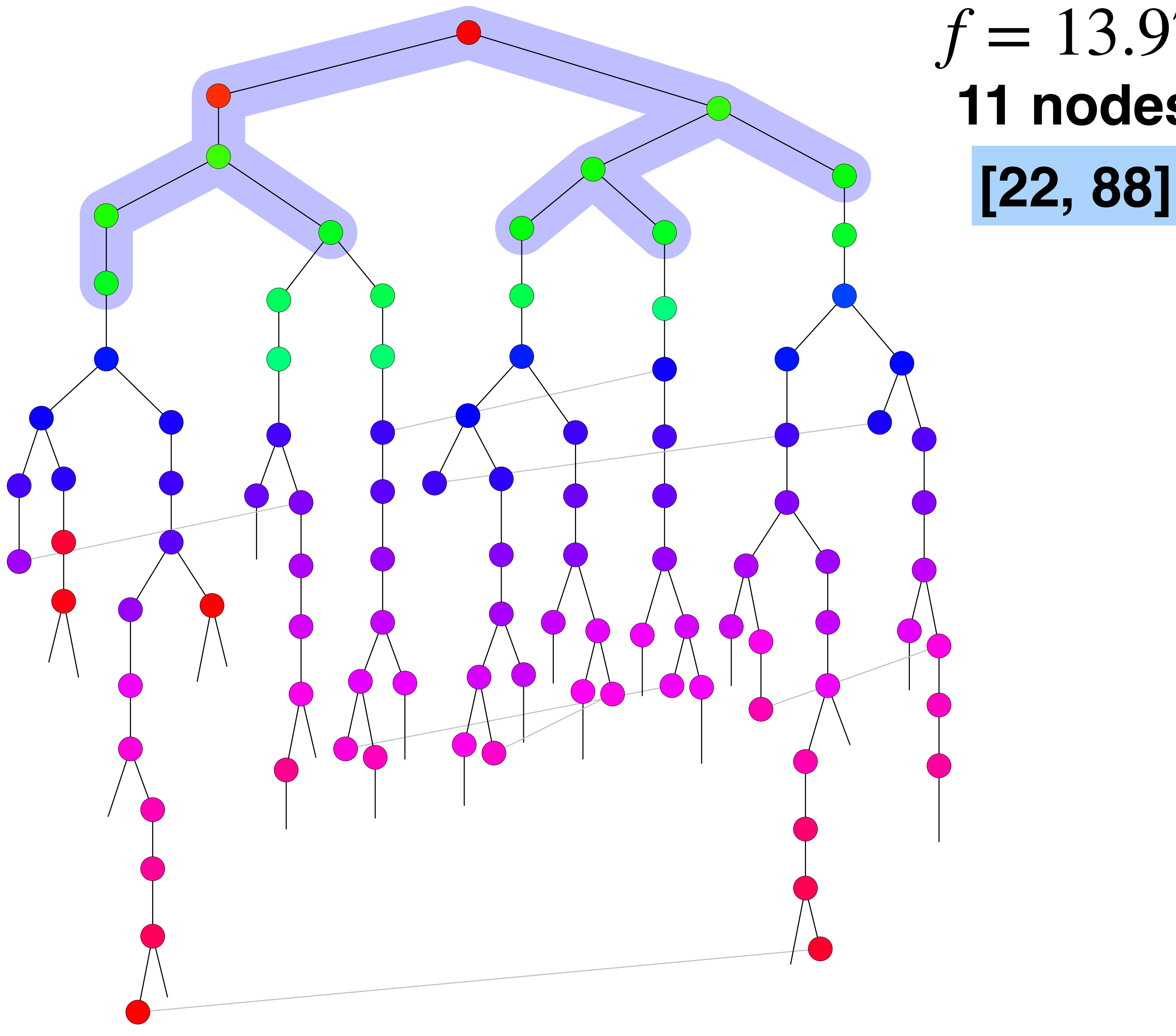


$f = 11.25$

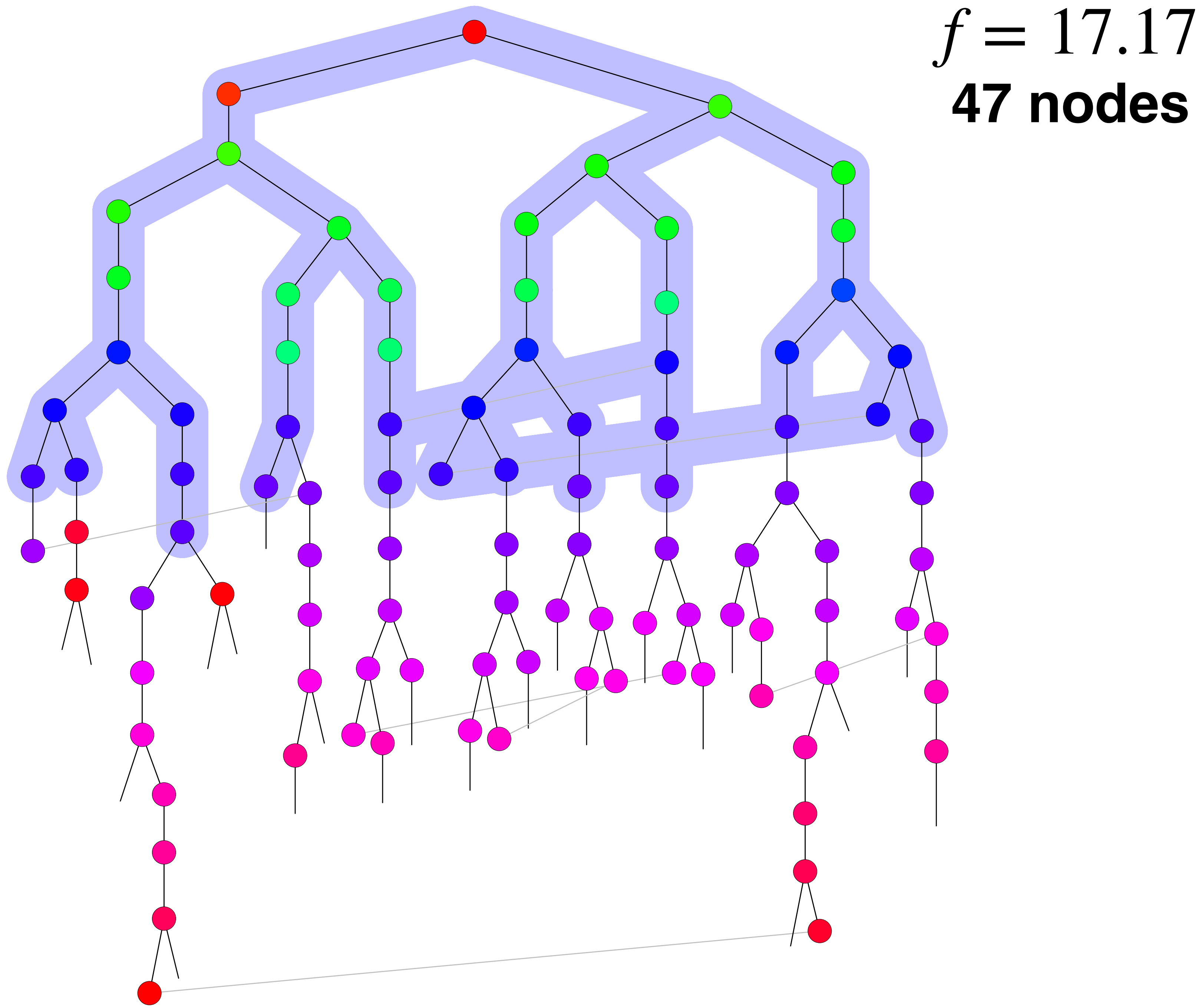
2 nodes

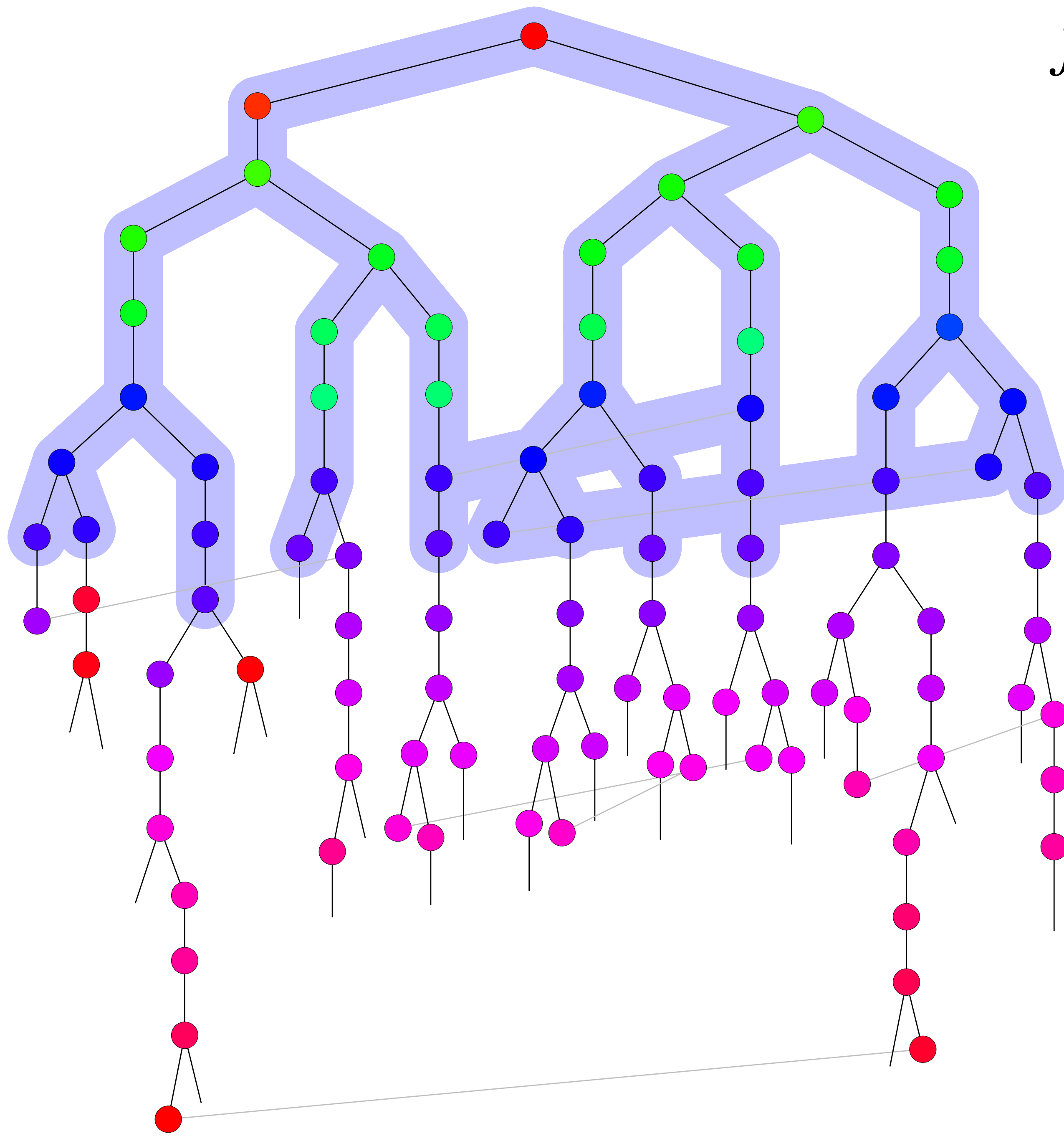
[4, 16]


$f = 13.97$
11 nodes



$f = 13.97$
11 nodes
[22, 88]

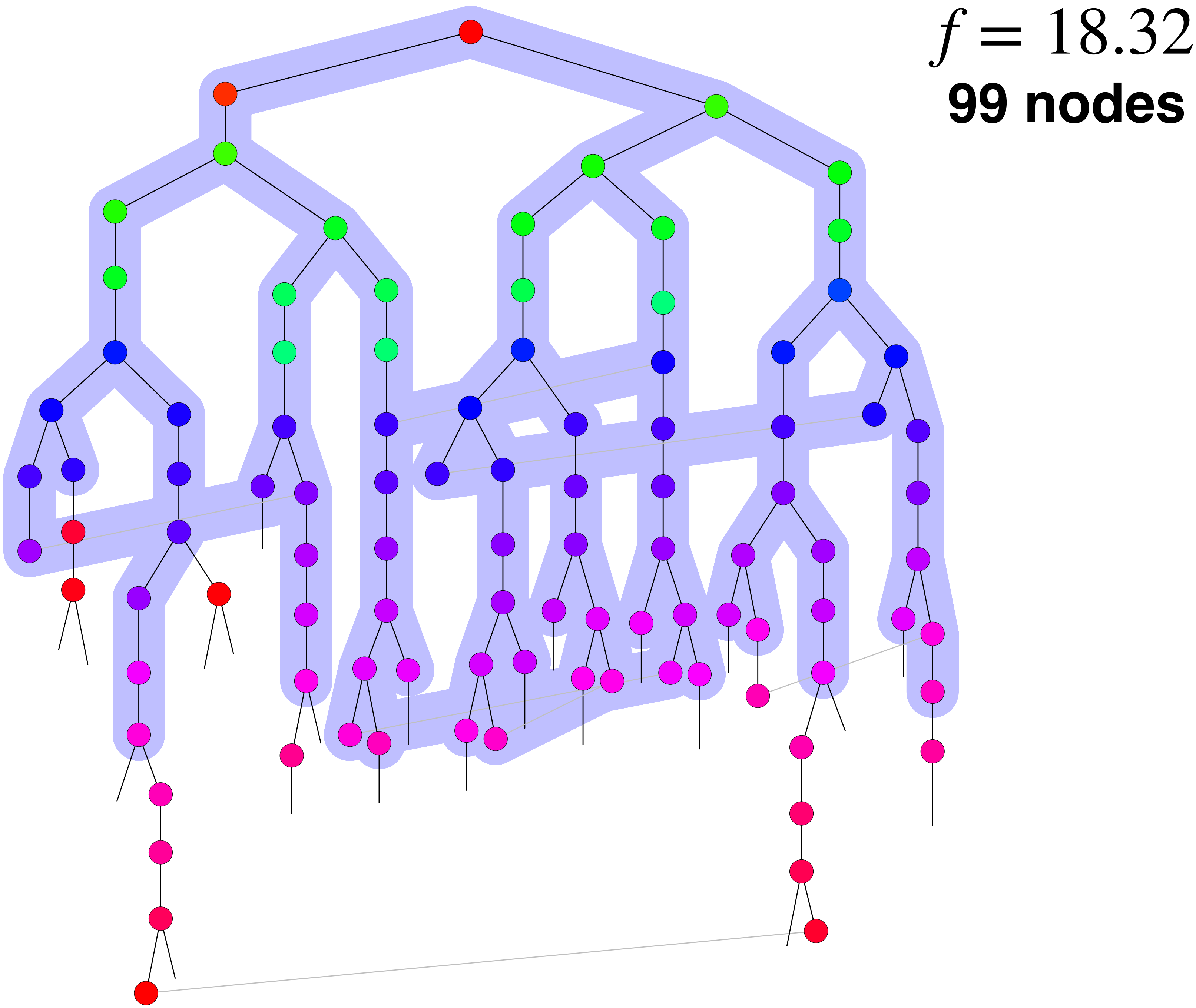

$f = 17.17$
47 nodes

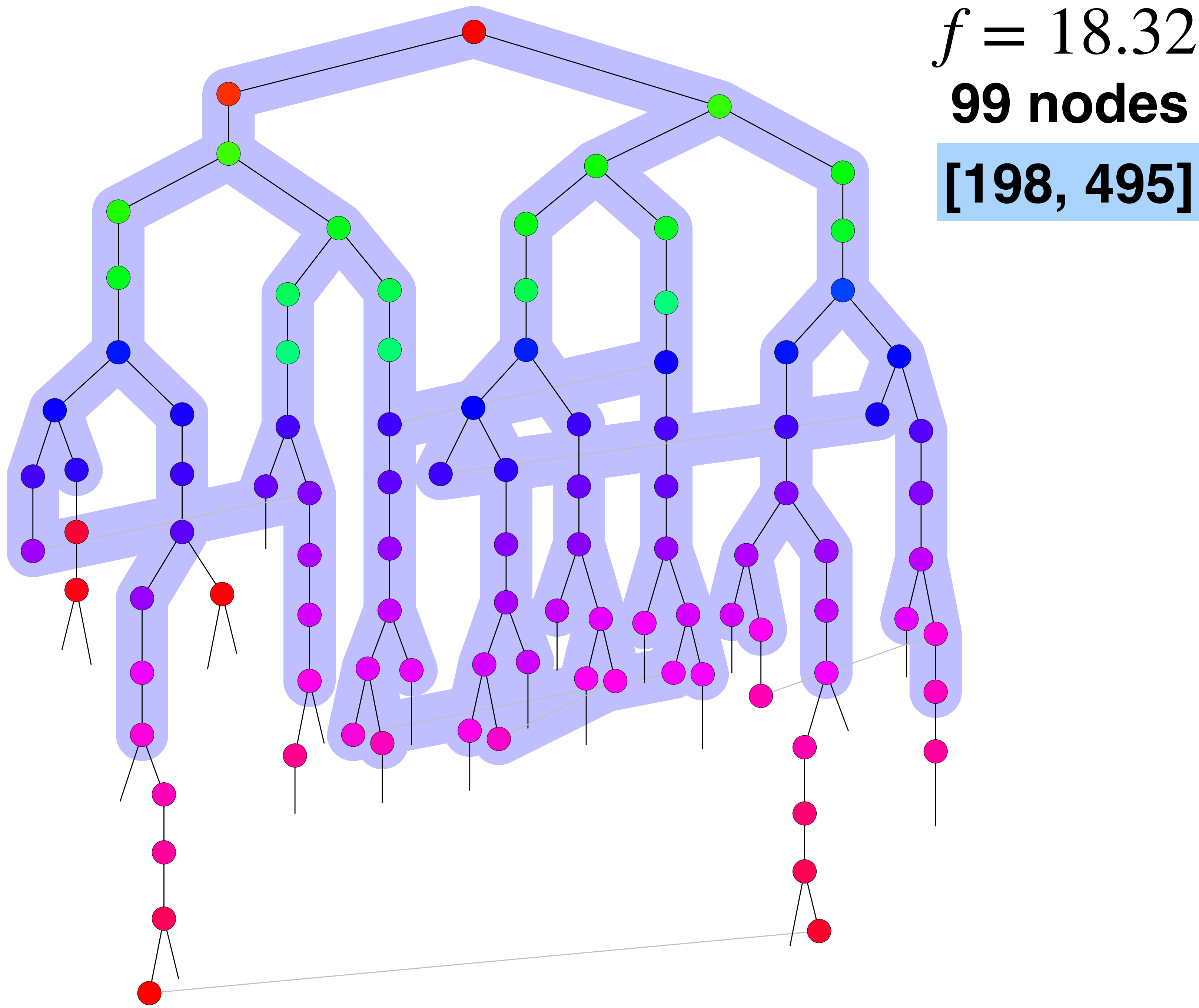


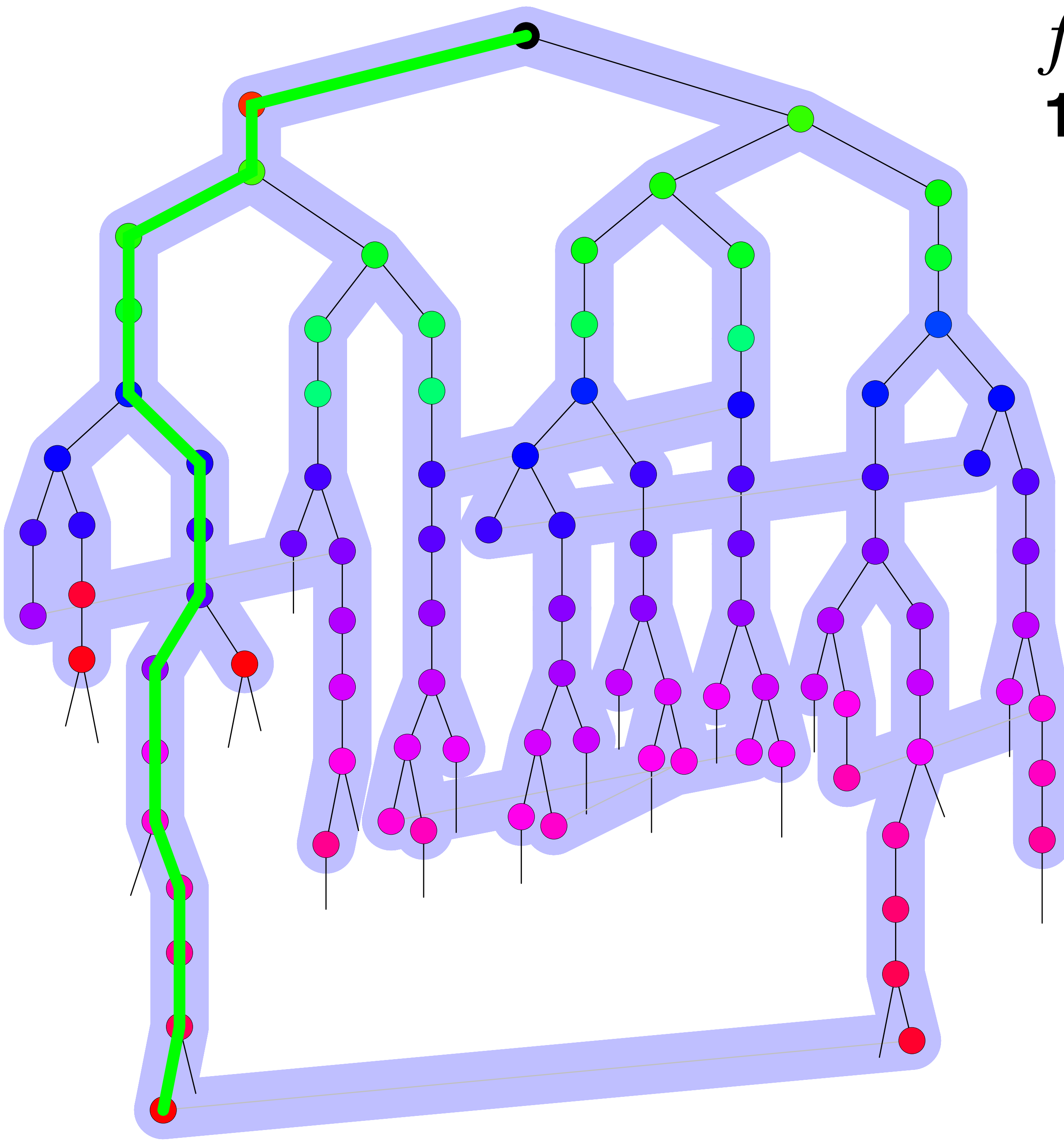
$f = 17.17$

47 nodes

[94, 376]


$f = 18.32$
99 nodes


$f = 18.32$
99 nodes
[198, 495]



$f = 19.35$

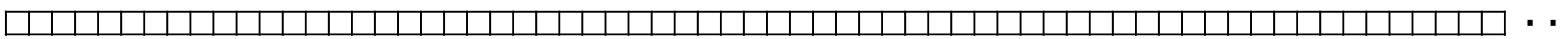
117 nodes

Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array

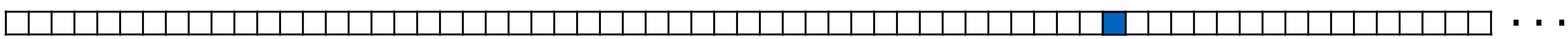
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



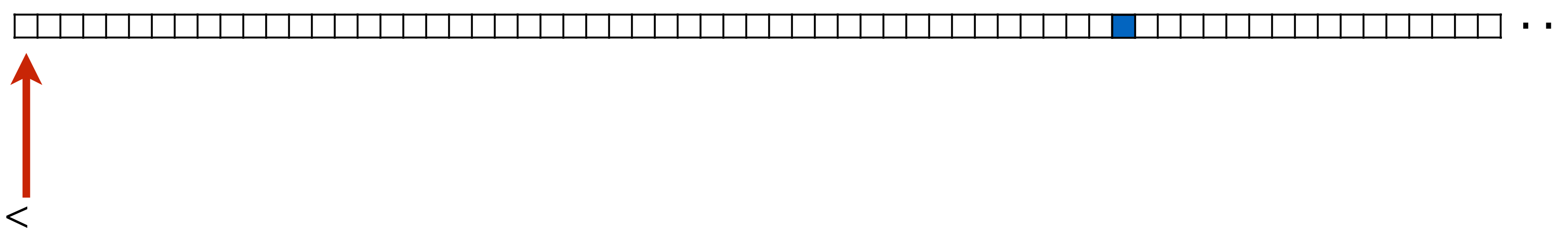
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



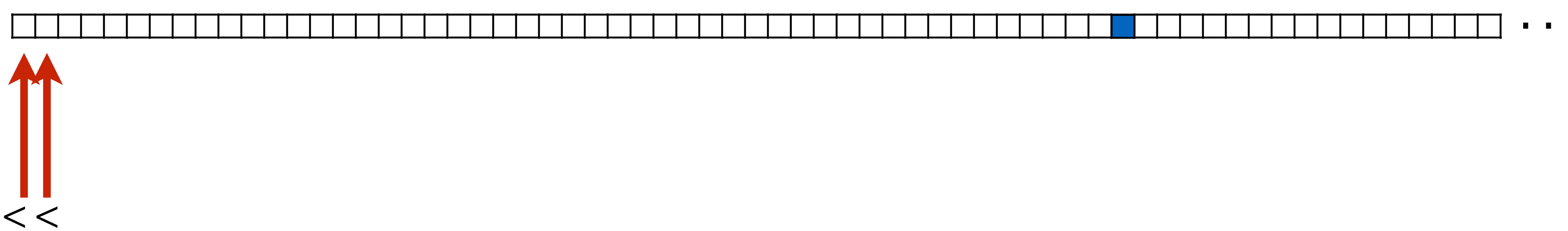
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



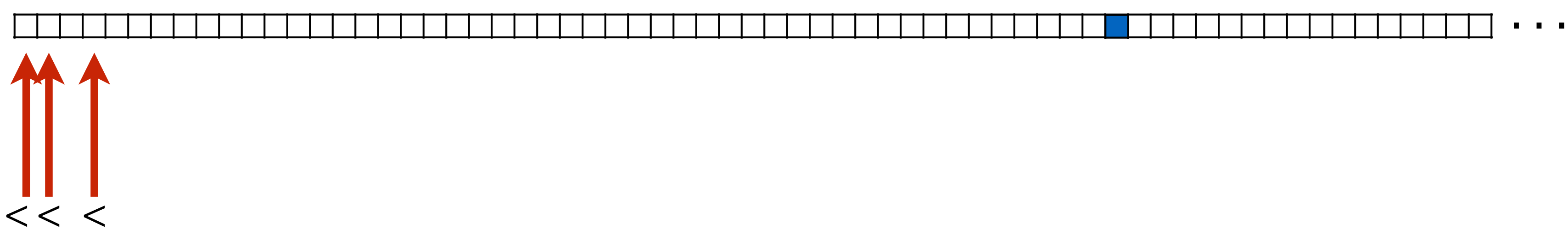
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



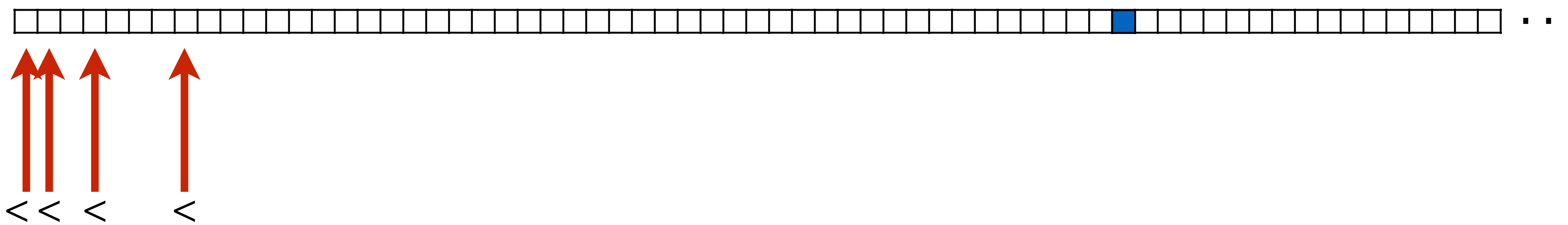
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



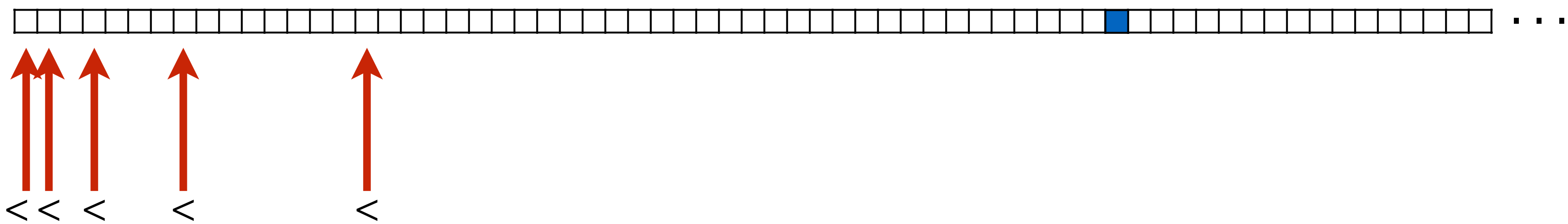
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



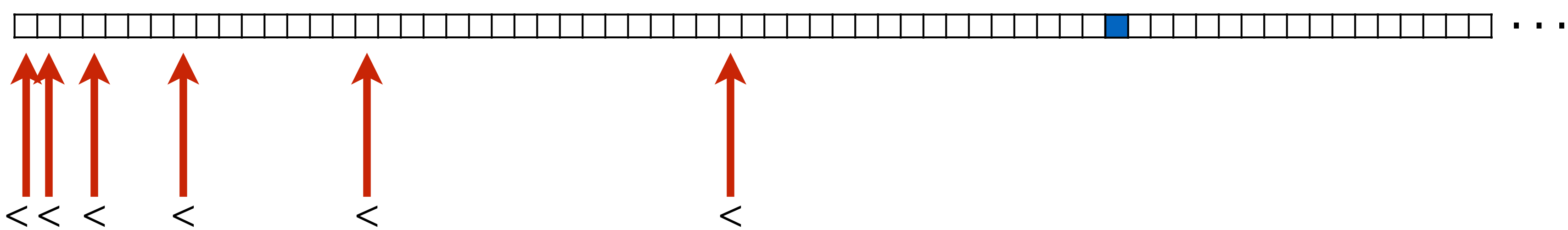
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



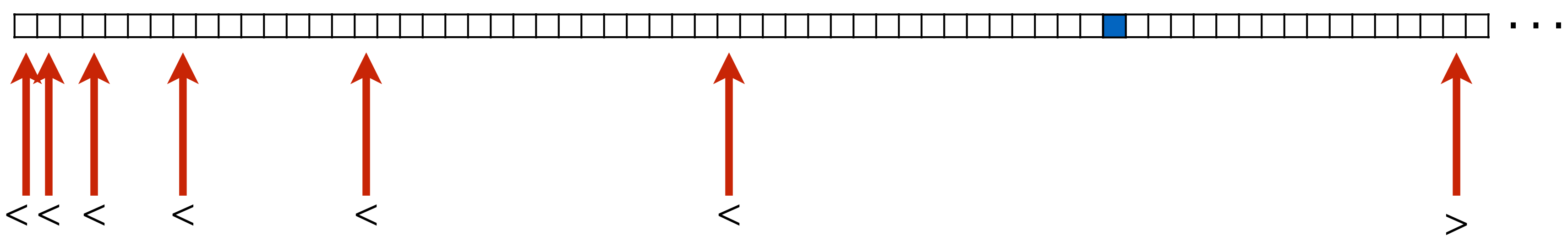
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



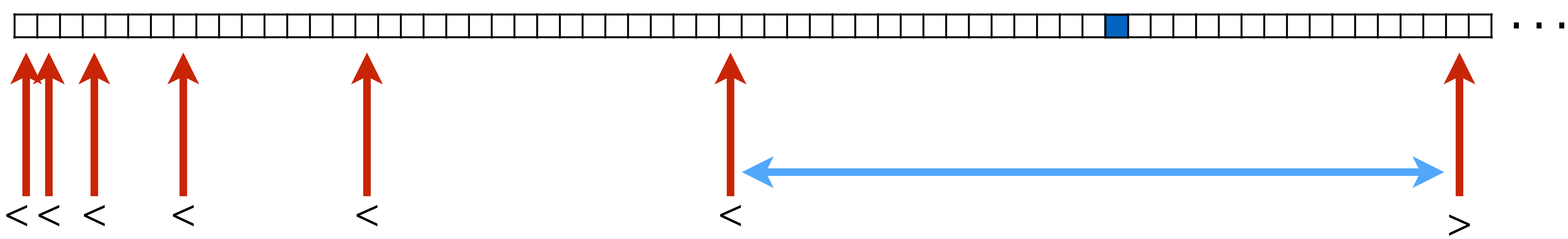
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



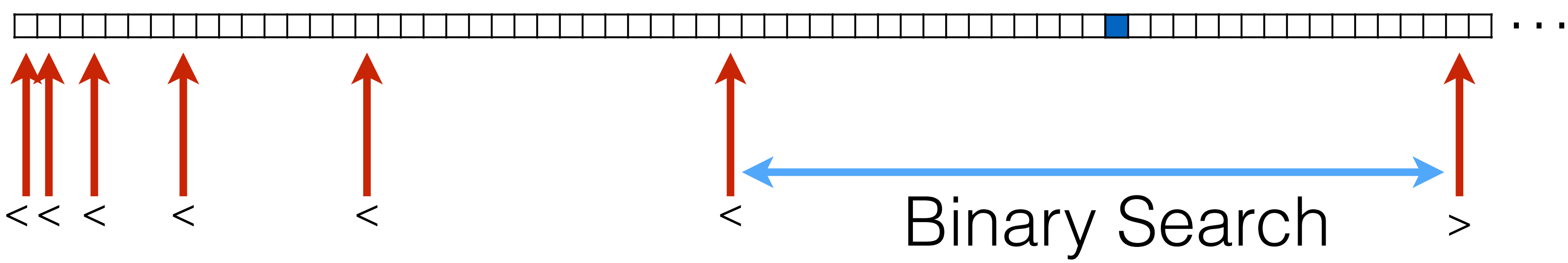
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



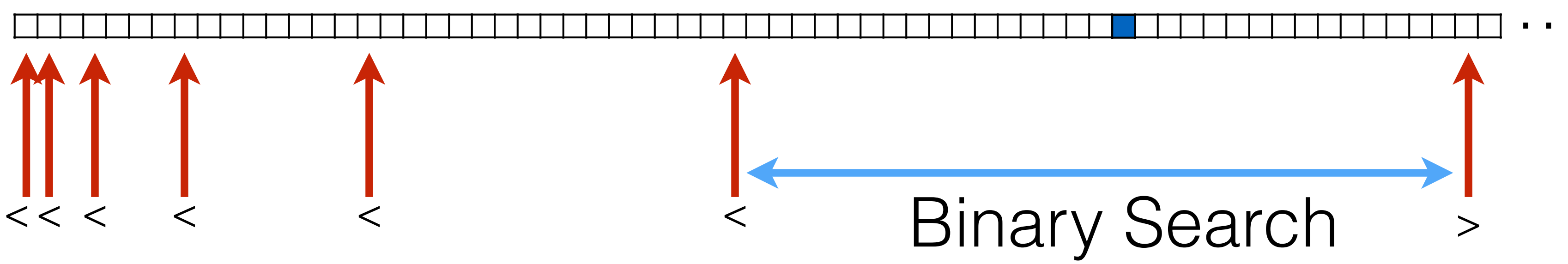
Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



Exponential Search

- Bentley and Yao, 1976
- Algorithm for searching sorted/unbounded array



- Running time: $\log(i)$

Nodes and *f*-costs

- Exponential Search:
  - Find *value* in *unbounded sorted array*
- Tree Search:
  - Find (*node expansions*) in (*f-costs*)

- Nodes expansions non-decreasing with *f*-cost

How does BTS work?

Budgeted search

- Like exponential search on f-costs

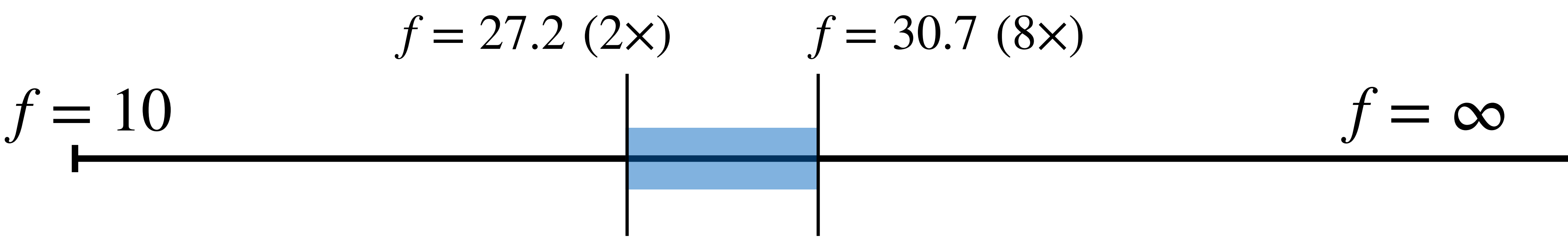
Budgeted search

- Like exponential search on f-costs

$f = 10$ $f = \infty$

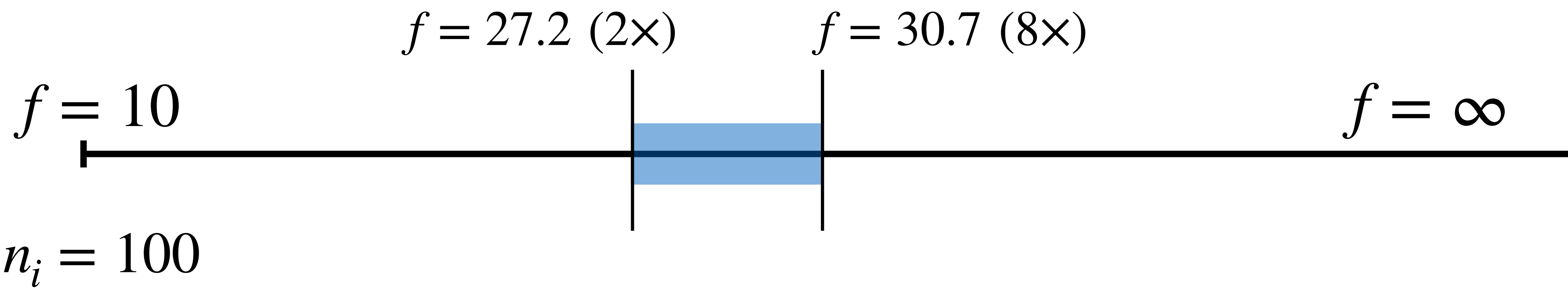
Budgeted search

- Like exponential search on f-costs



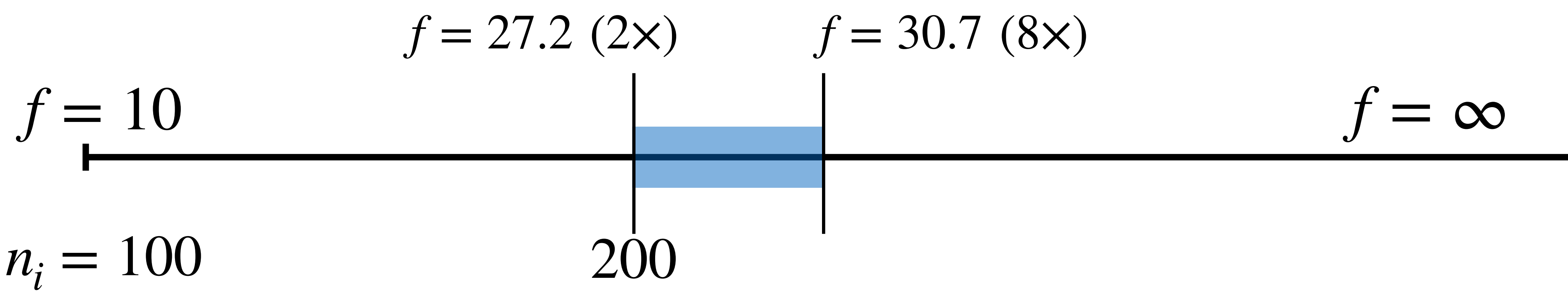
Budgeted search

- Like exponential search on f-costs



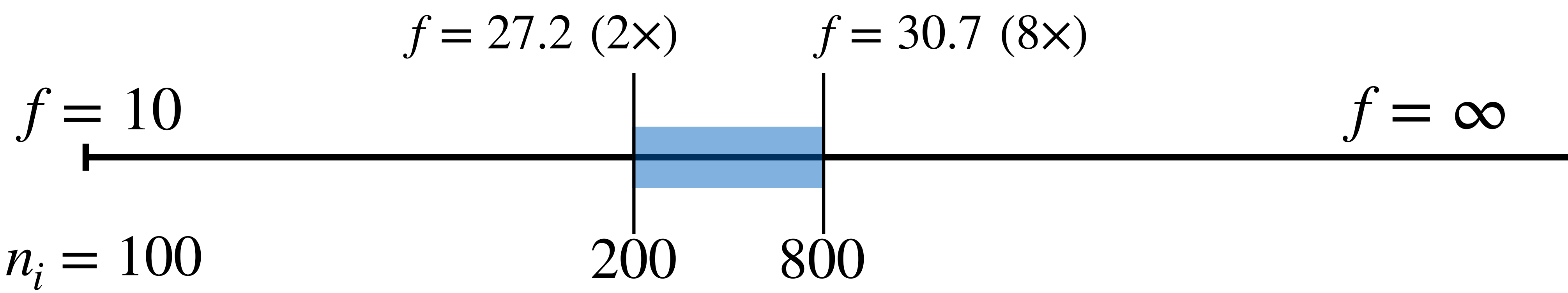
Budgeted search

- Like exponential search on f-costs



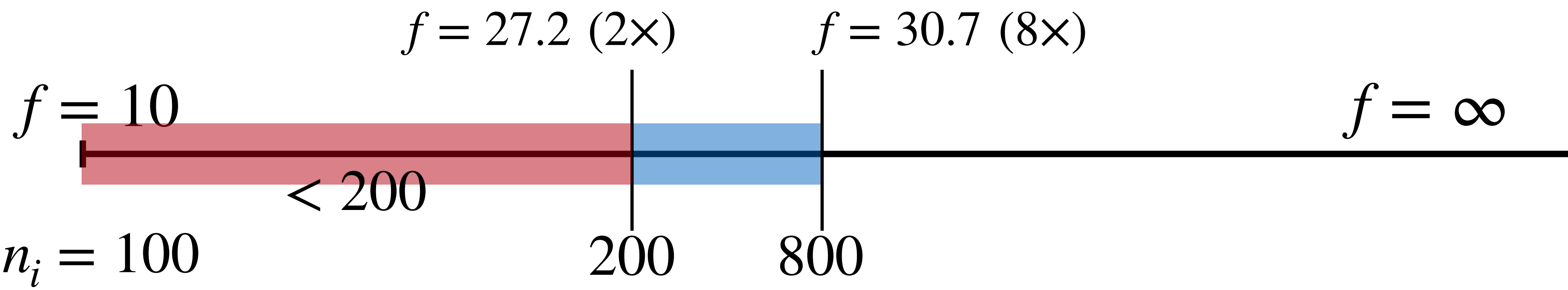
Budgeted search

- Like exponential search on f-costs



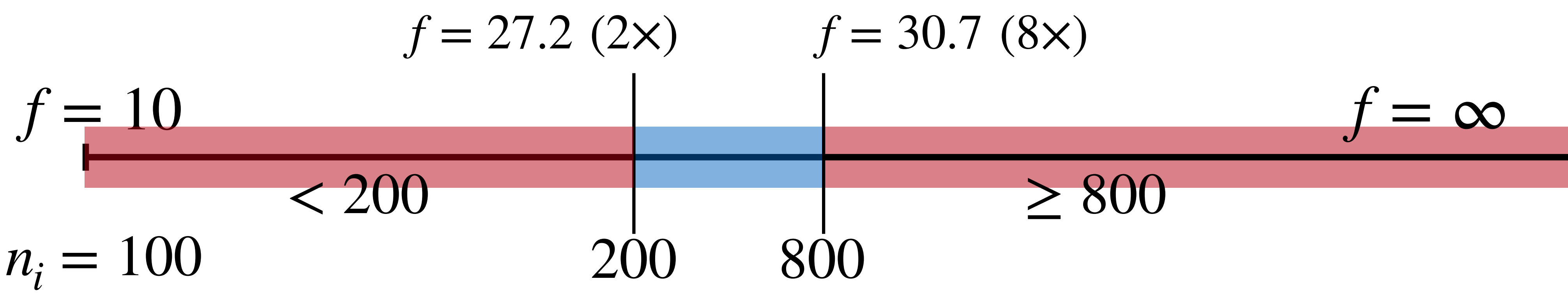
Budgeted search

- Like exponential search on f-costs



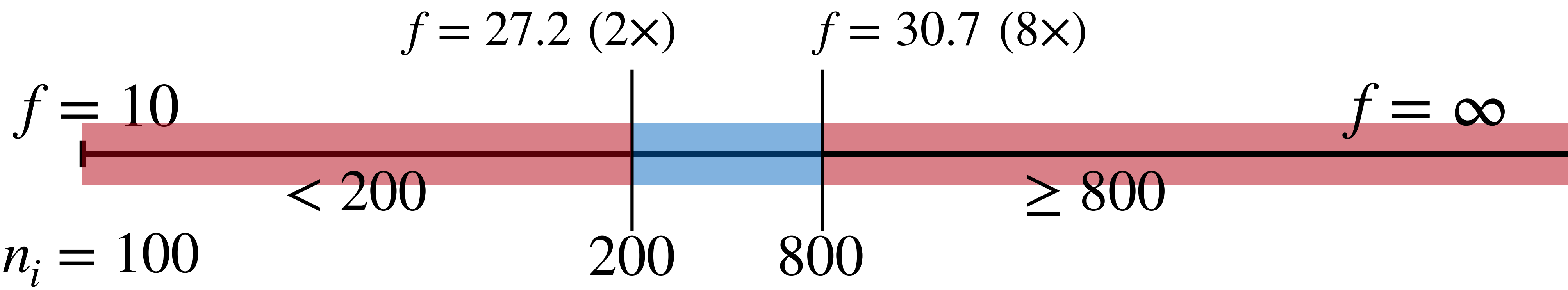
Budgeted search

- Like exponential search on f-costs



Budgeted search

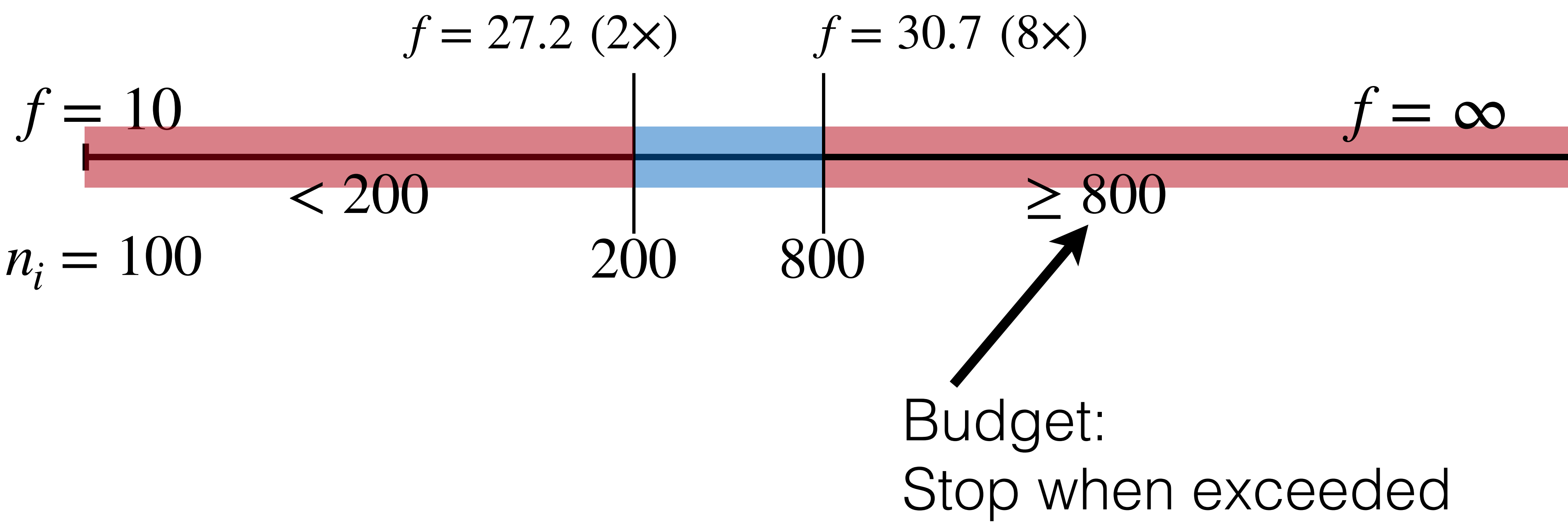
- Like exponential search on f-costs



Budget:
Stop when exceeded

Budgeted search

- Like exponential search on f-costs



Budgeted search

- Like exponential search on f-costs

Budgeted search

- Like exponential search on f-costs

$f = 10$

Budgeted search

- Like exponential search on f-costs

$f = 10$

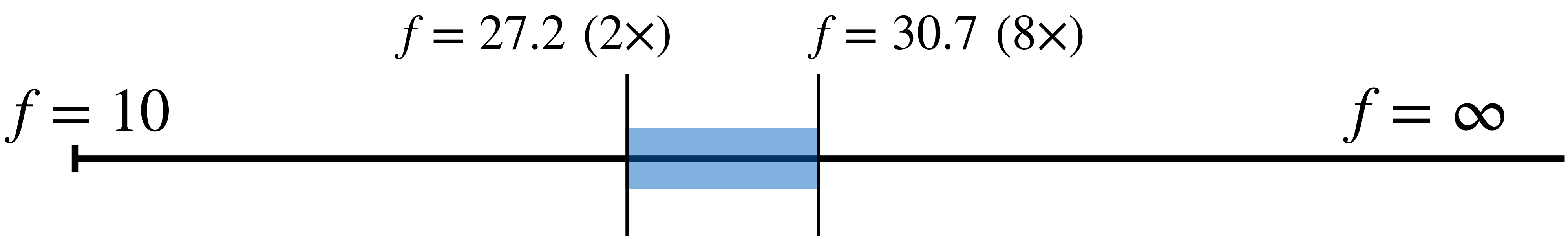
Budgeted search

- Like exponential search on f-costs

$f = 10$ $\qquad\qquad$ $f = \infty$

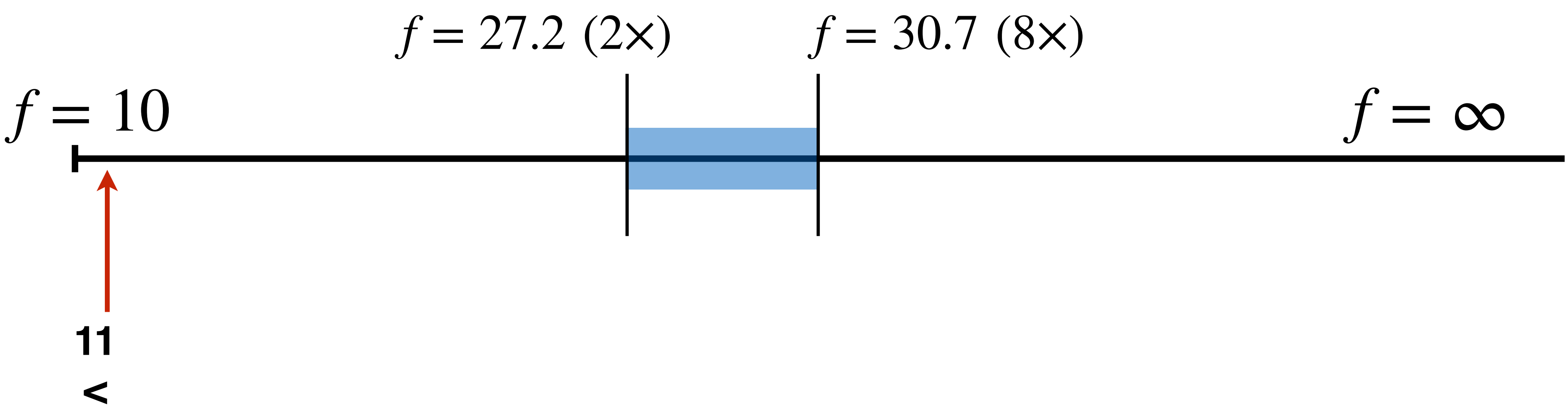
Budgeted search

- Like exponential search on f-costs



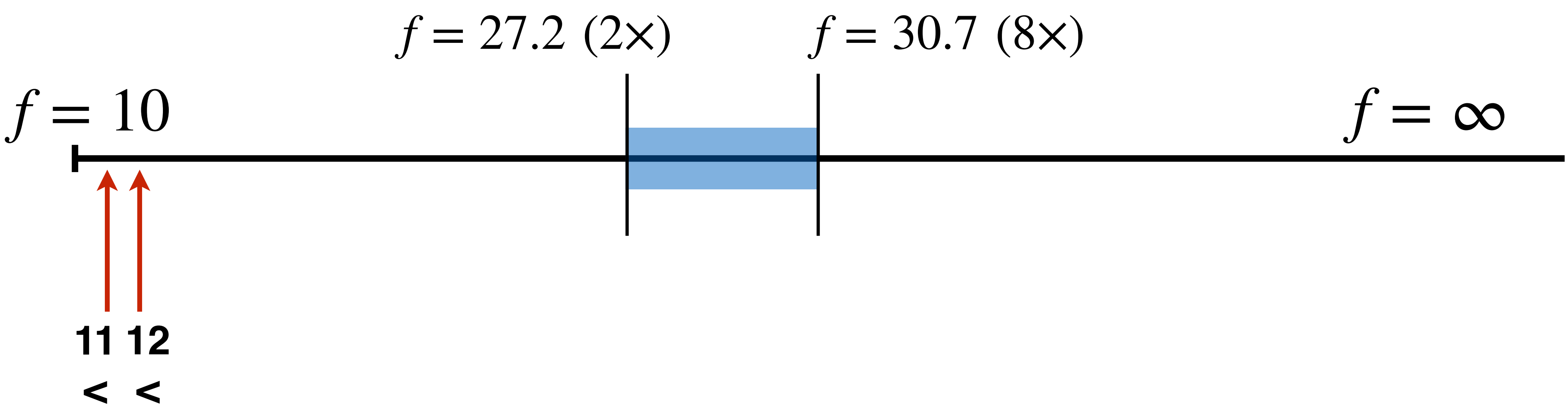
Budgeted search

- Like exponential search on f-costs



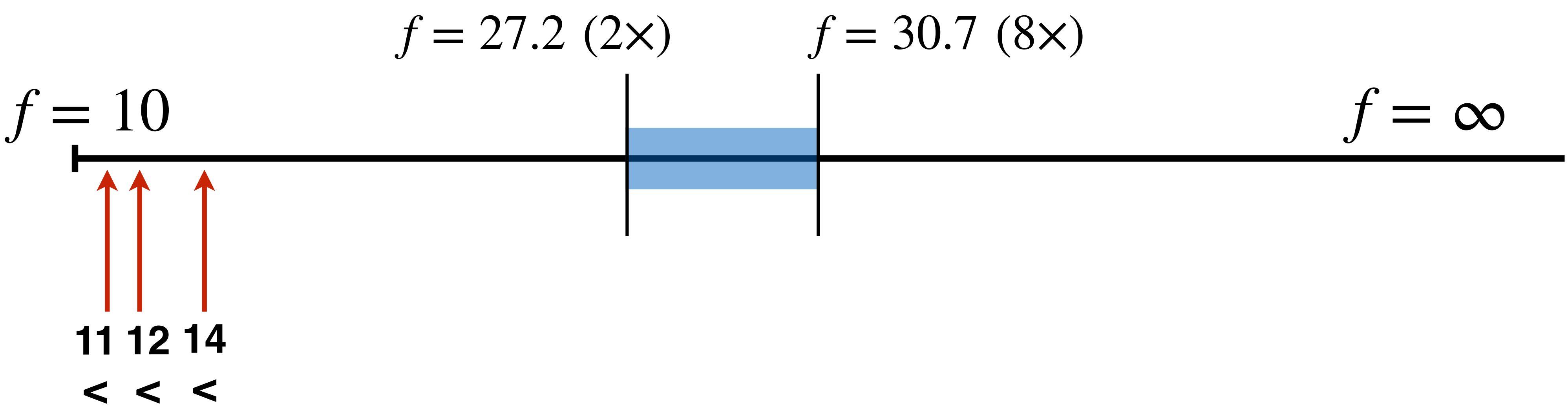
Budgeted search

- Like exponential search on f-costs



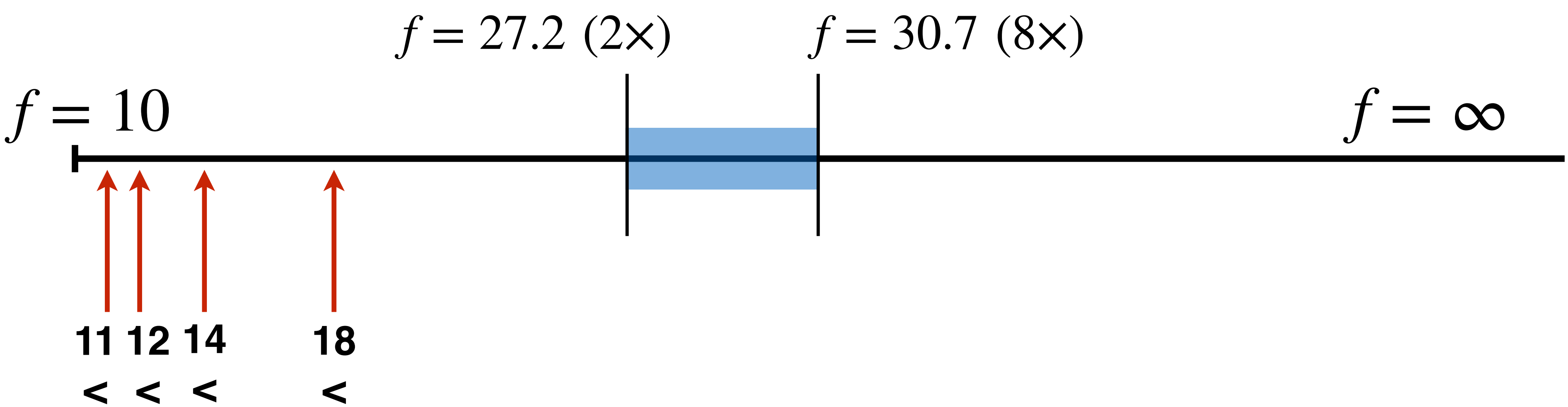
Budgeted search

- Like exponential search on f-costs



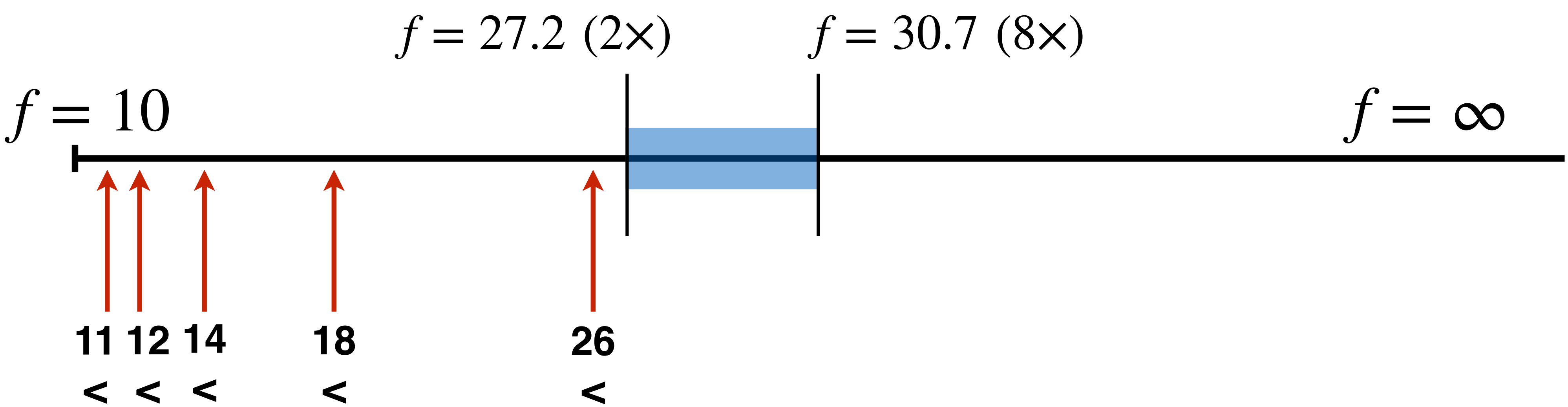
Budgeted search

- Like exponential search on f-costs



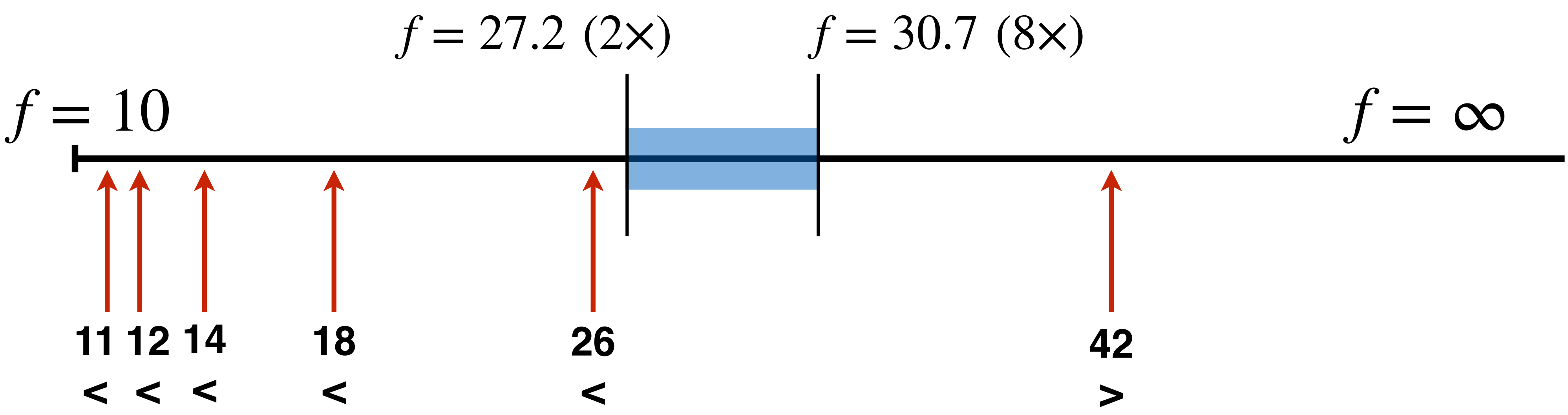
Budgeted search

- Like exponential search on f-costs



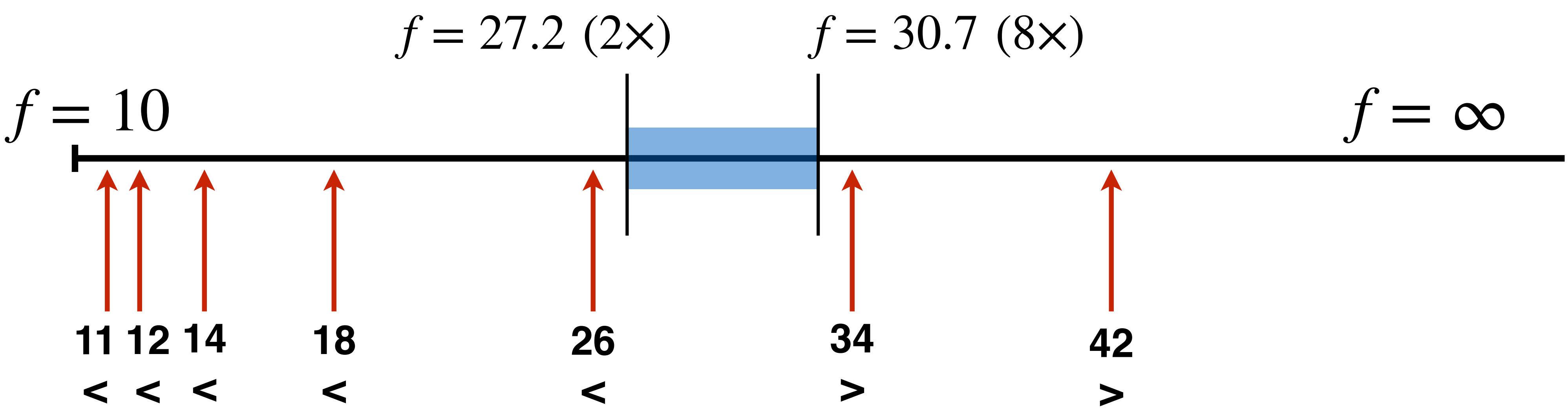
Budgeted search

- Like exponential search on f-costs



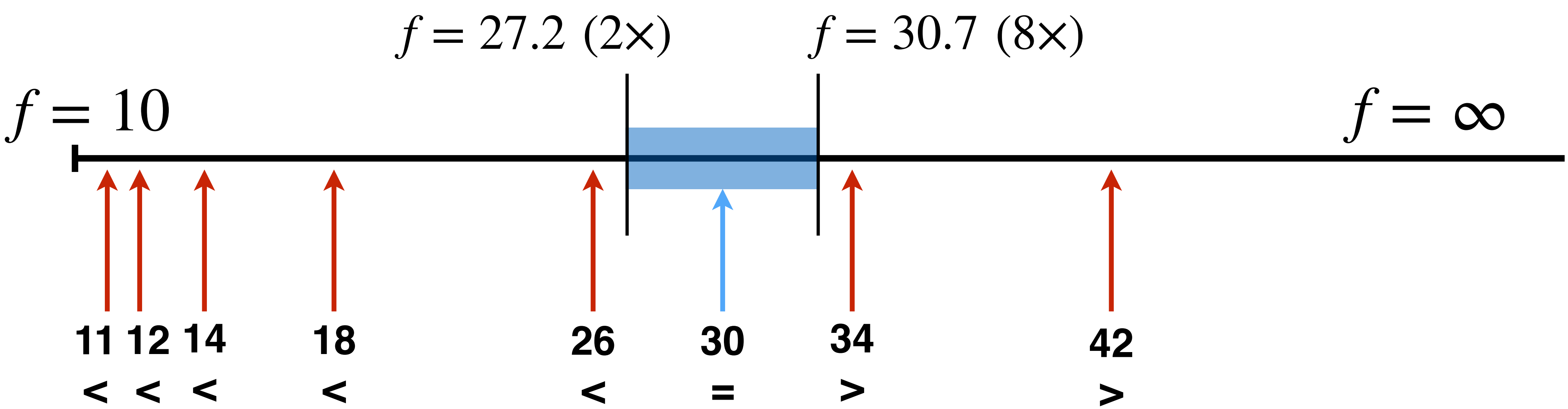
Budgeted search

- Like exponential search on f-costs



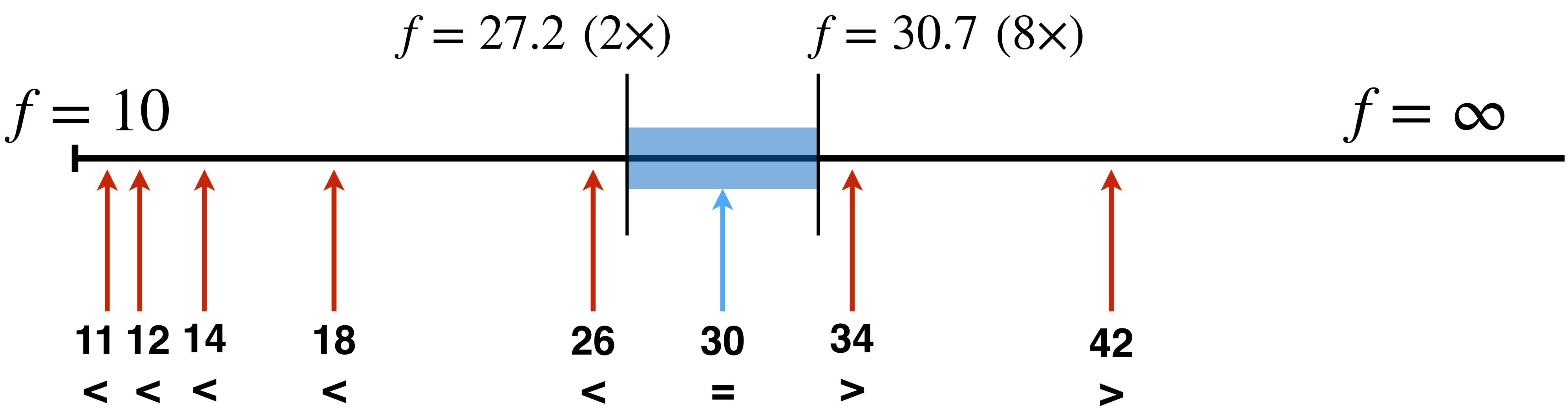
Budgeted search

- Like exponential search on f-costs



Budgeted search

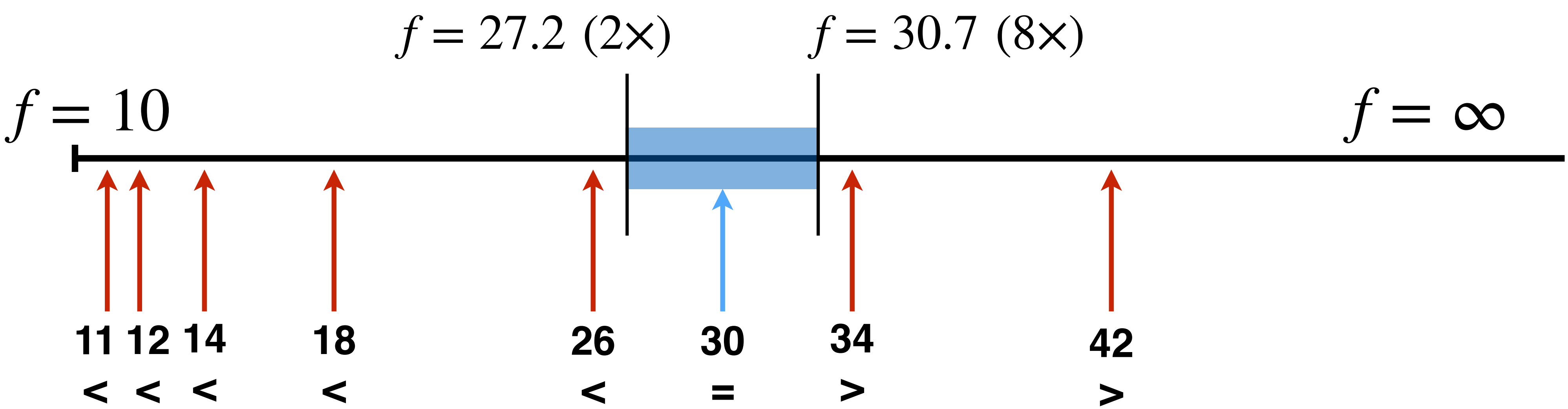
- Like exponential search on f-costs



$n_i \log(f_i)$

Budgeted search

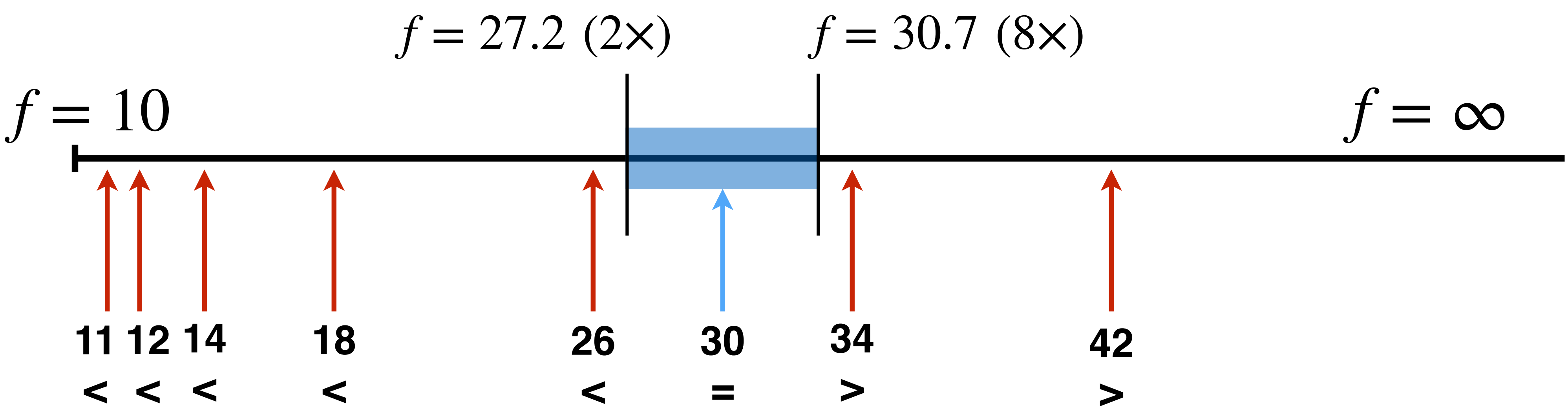
- Like exponential search on f-costs



$$\sum_{0}^{i} n_i \log(f_i)$$

Budgeted search

- Like exponential search on f-costs



$$\sum_{0}^{i} n_i \log(f_i) < \left(\sum_{0}^{i} n_i\right) \log(C^*) \approx N \log(C^*)$$

BTS Phases

Find next *f*-cost bound:

BTS Phases

Find next *f*-cost bound:

1. Search with conservative *f*, ∞ budget (IDA*)

BTS Phases

Find next f-cost bound:

1. Search with conservative f, ∞ budget (IDA*)
2. Grow f exponentially, constant budget

BTS Phases

Find next *f*-cost bound:

1. Search with conservative *f*, ∞ budget (IDA*)
2. Grow *f* exponentially, constant budget
3. Do binary search on *f*, constant budget

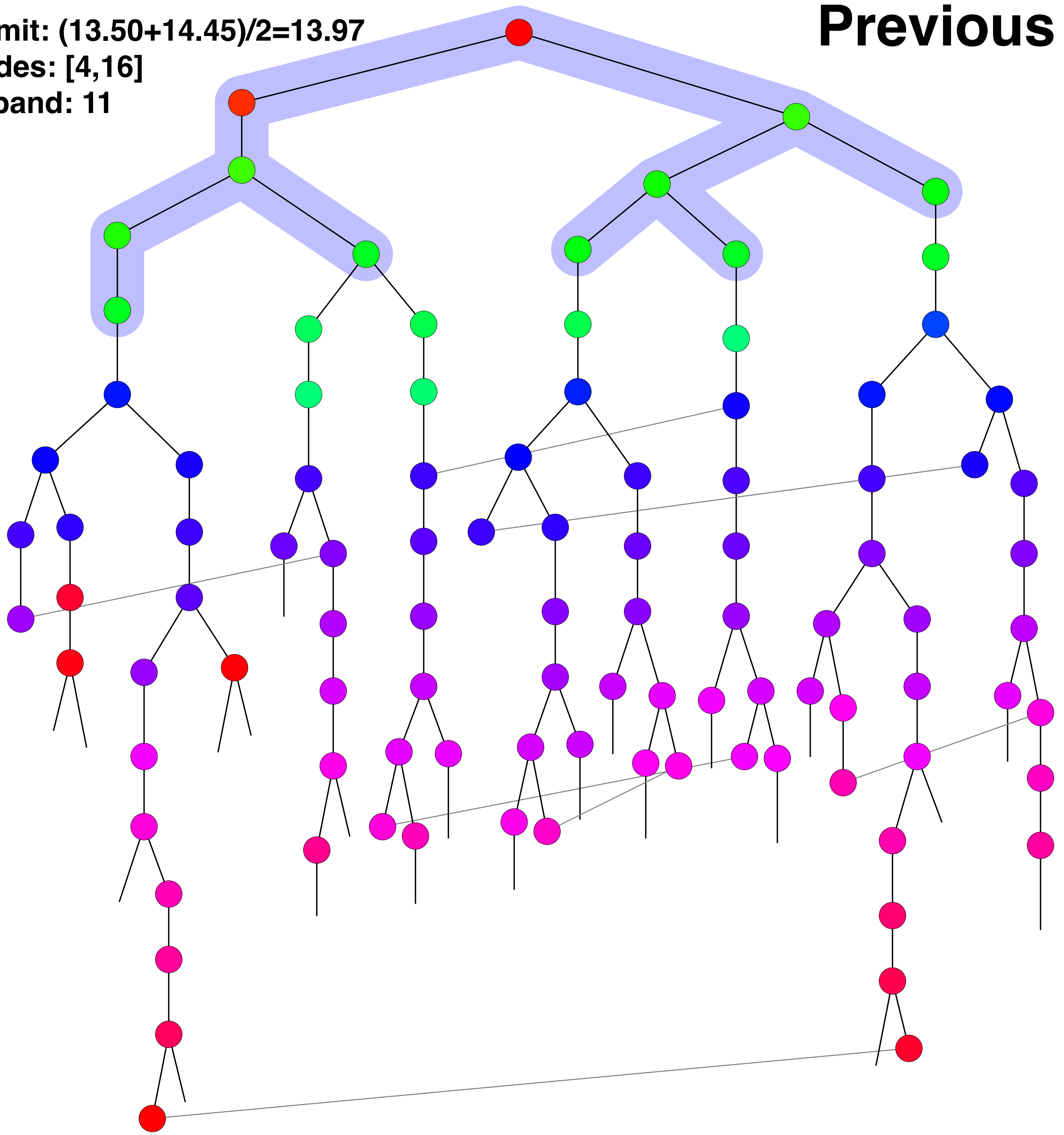
BTS Phases

Find next *f*-cost bound:

1. Search with conservative *f*, ∞ budget (IDA*)
2. Grow *f* exponentially, constant budget
3. Do binary search on *f*, constant budget

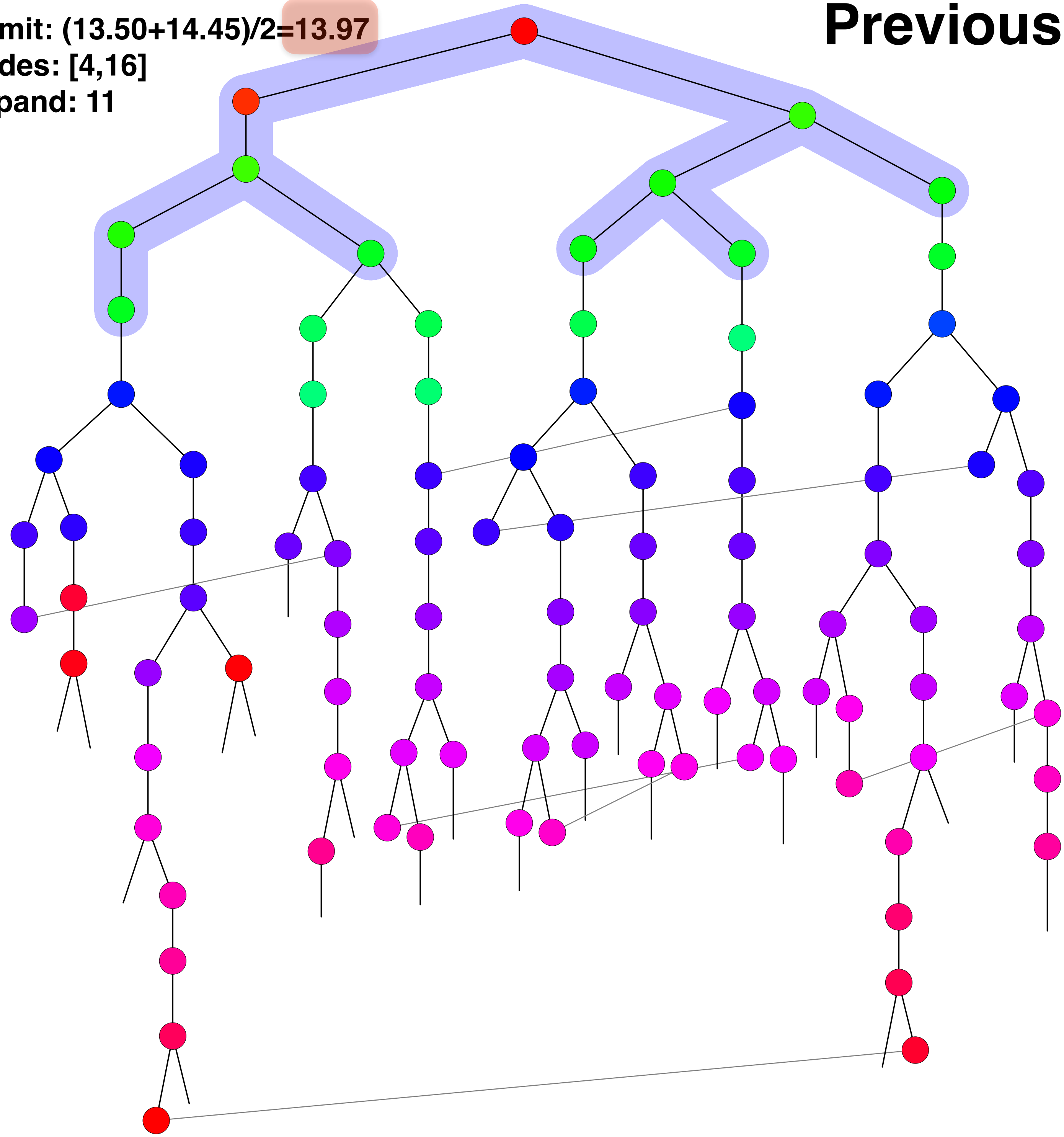
Previous Iteration

f-limit: (13.50+14.45)/2=13.97
nodes: [4,16]
expand: 11



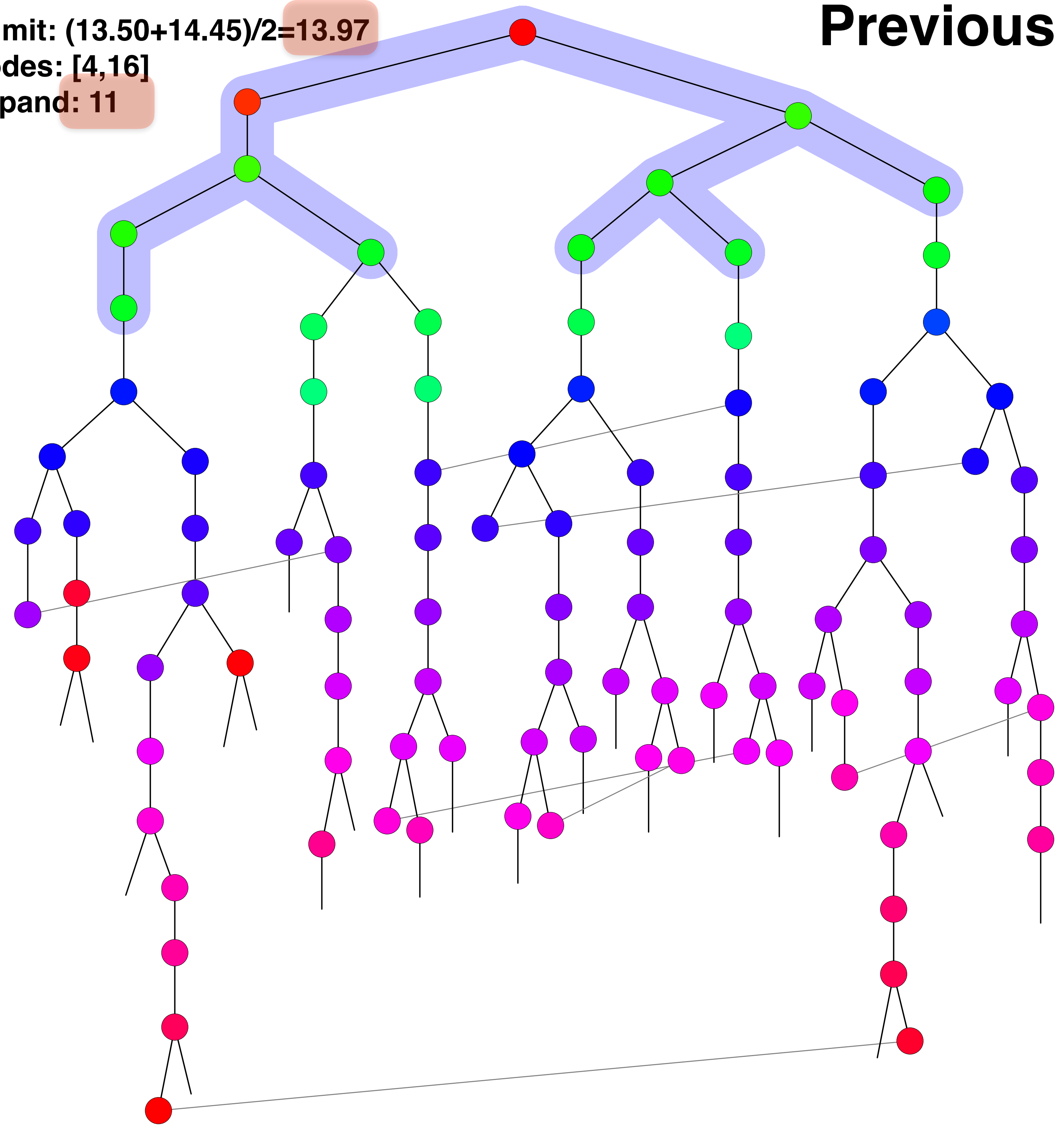
Previous Iteration

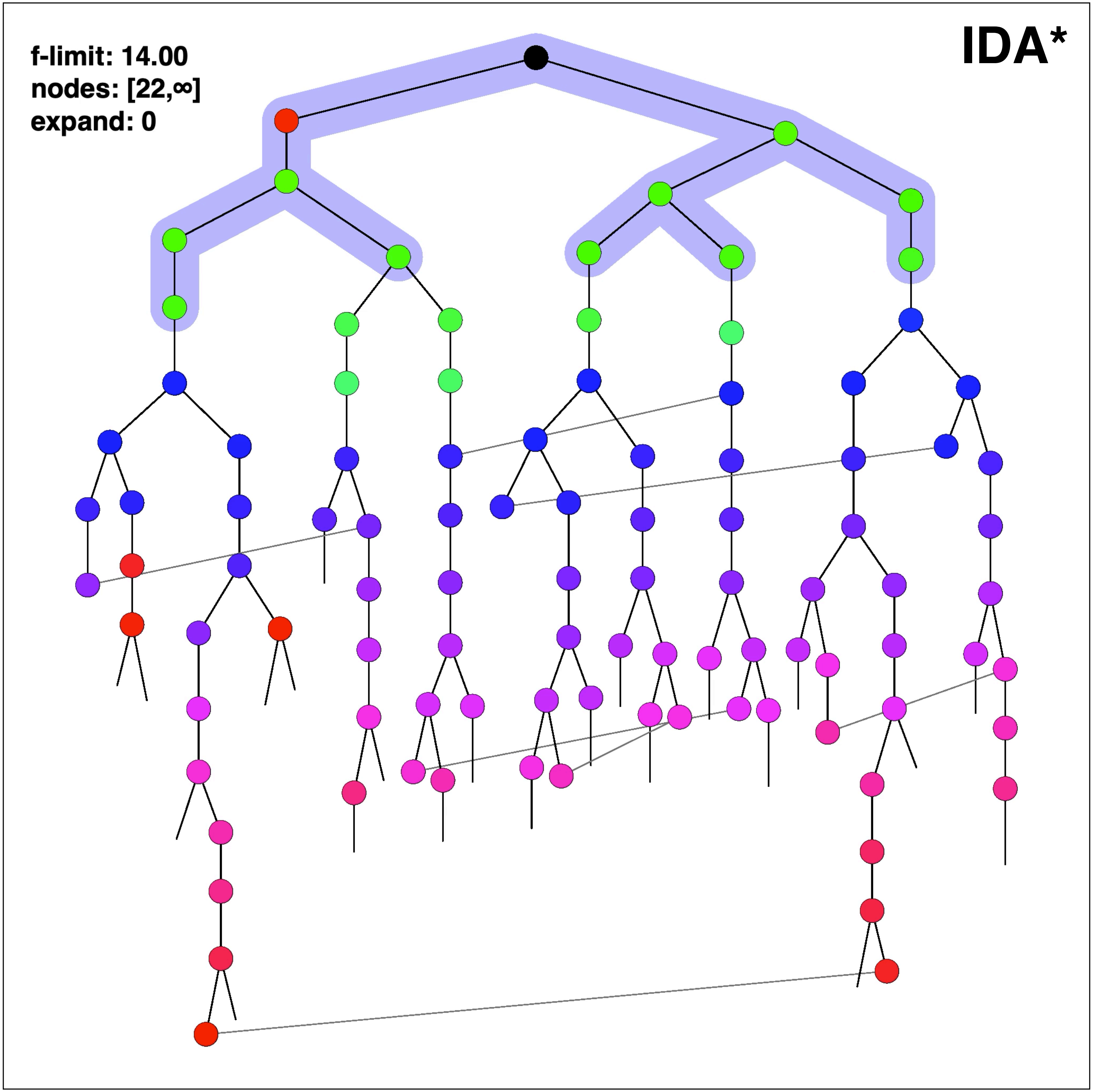
f-limit: (13.50+14.45)/2=13.97
nodes: [4,16]
expand: 11

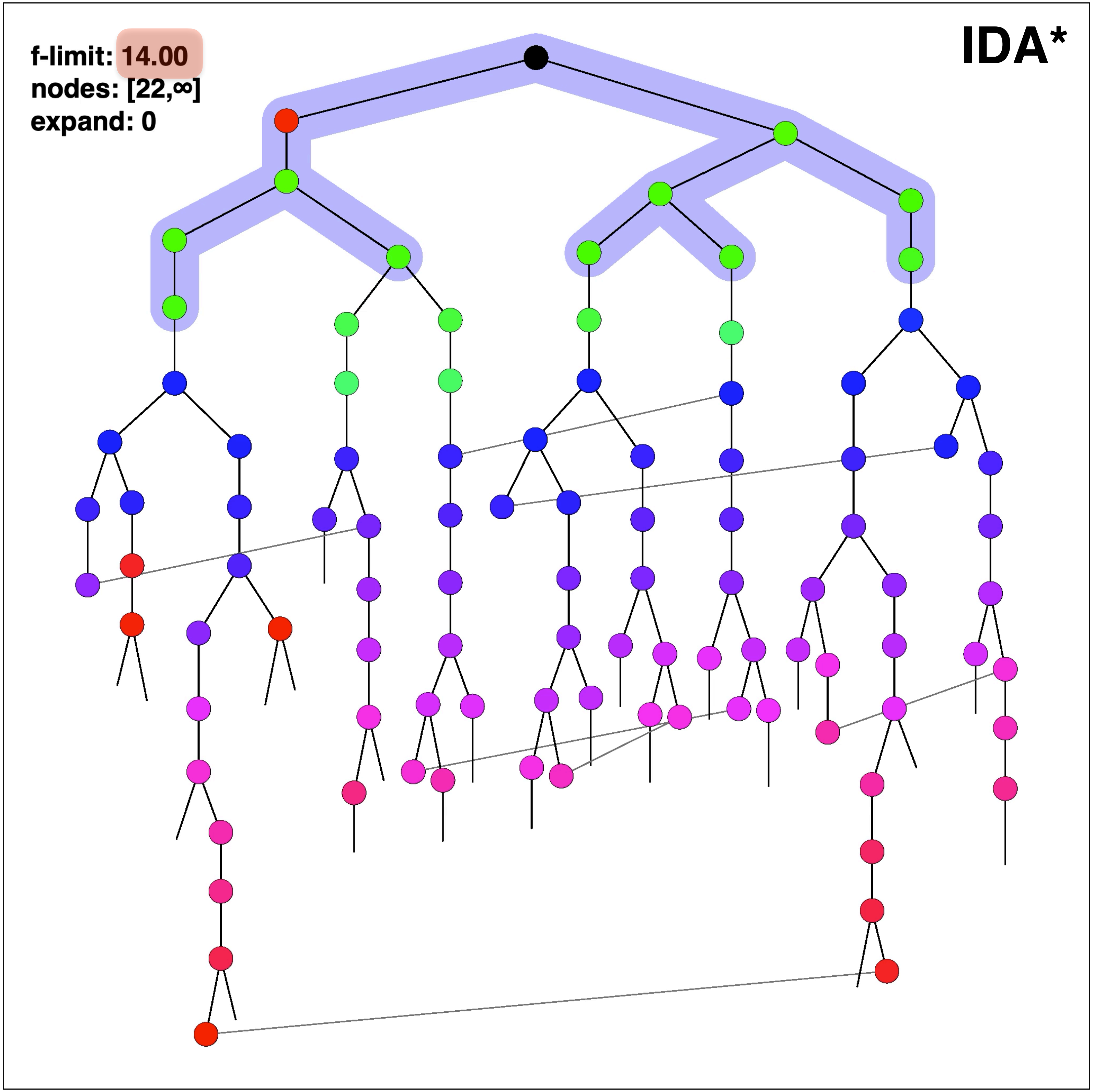


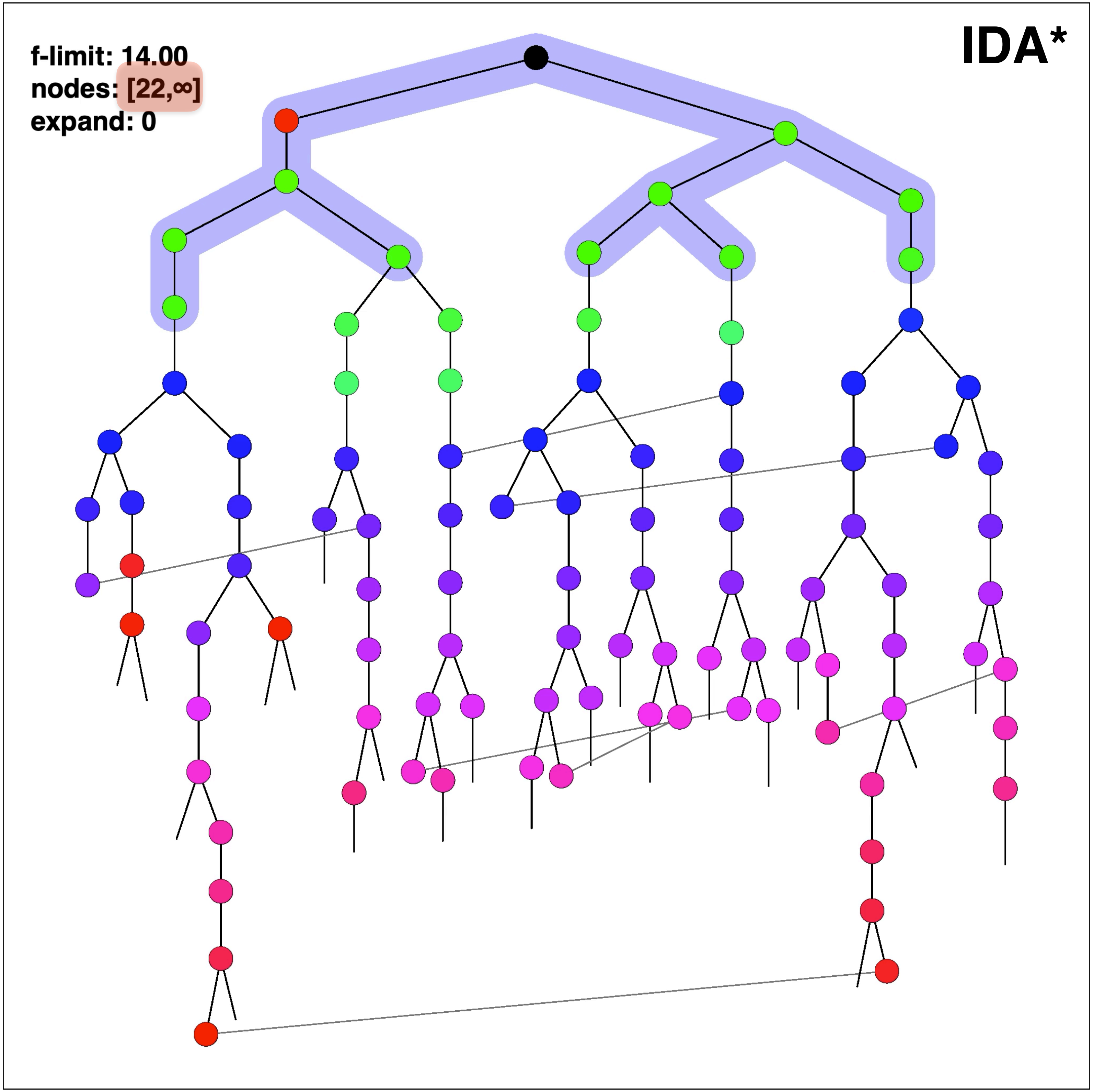
Previous Iteration

f-limit: (13.50+14.45)/2=13.97
nodes: [4,16]
expand: 11

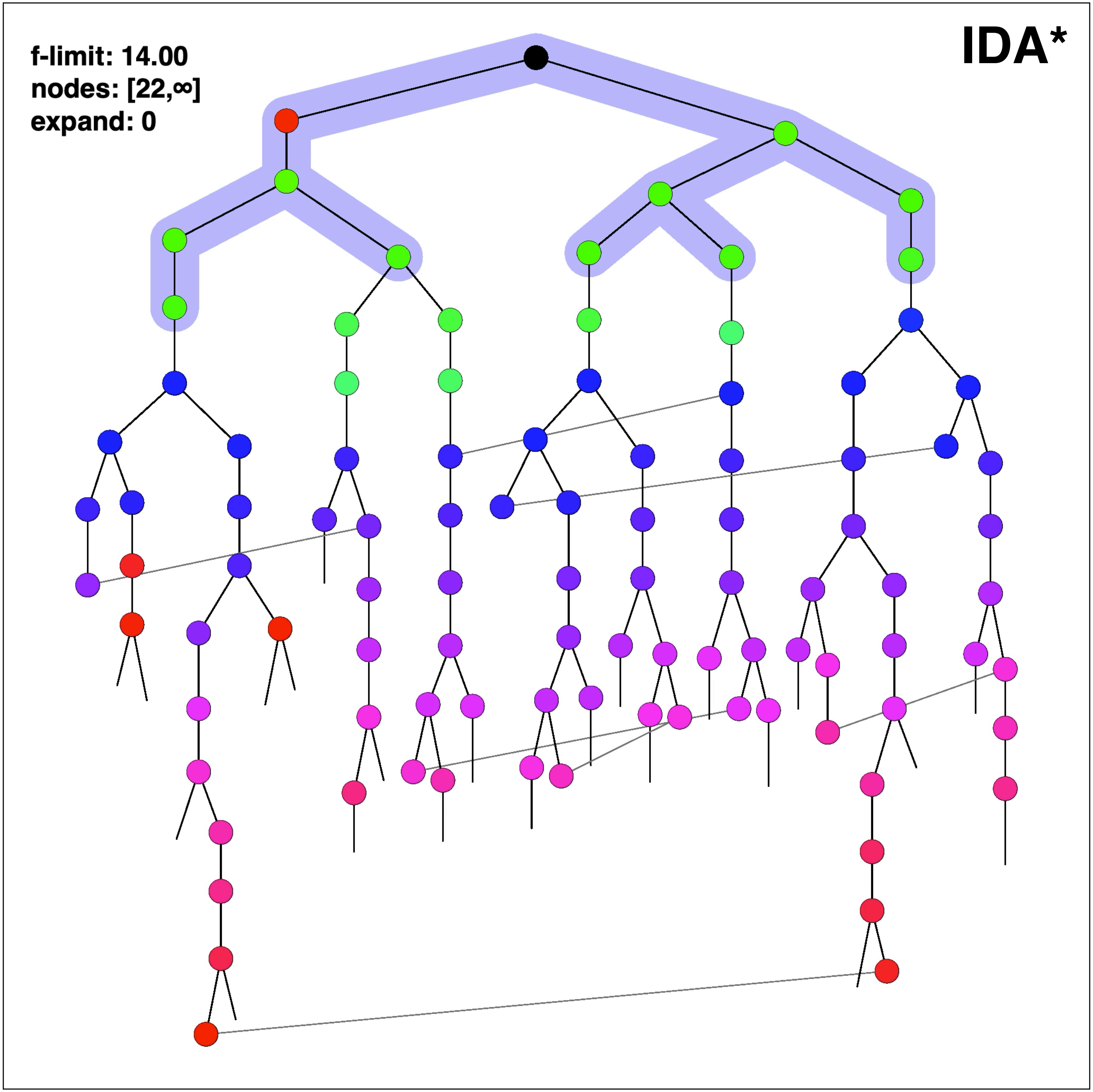



f-limit: 14.00
nodes: [22,∞]
expand: 0
IDA*


f-limit: 14.00
nodes: [22,∞]
expand: 0
IDA*

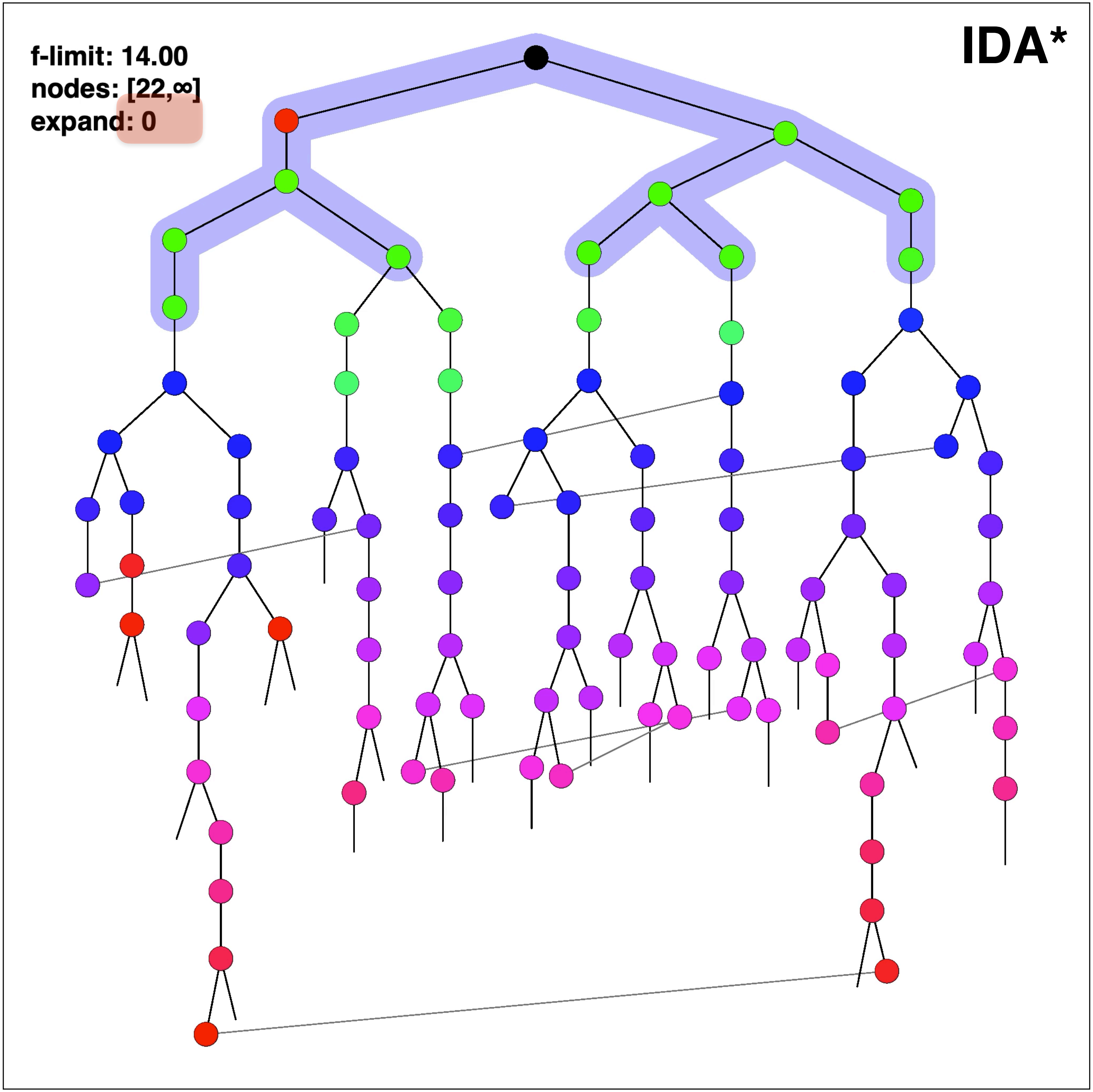


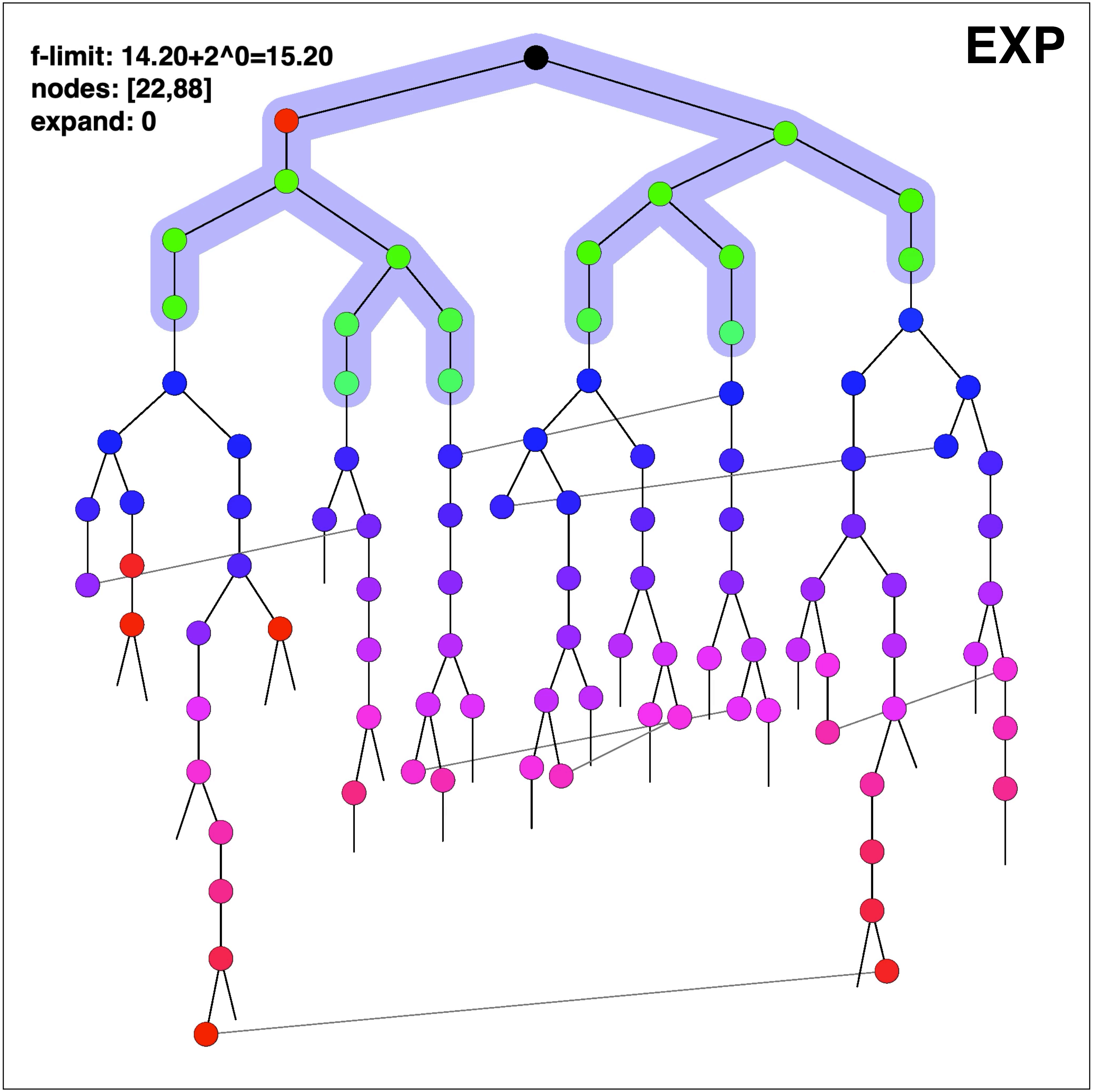
f-limit: 14.00
nodes: [22,∞]
expand: 0
IDA*

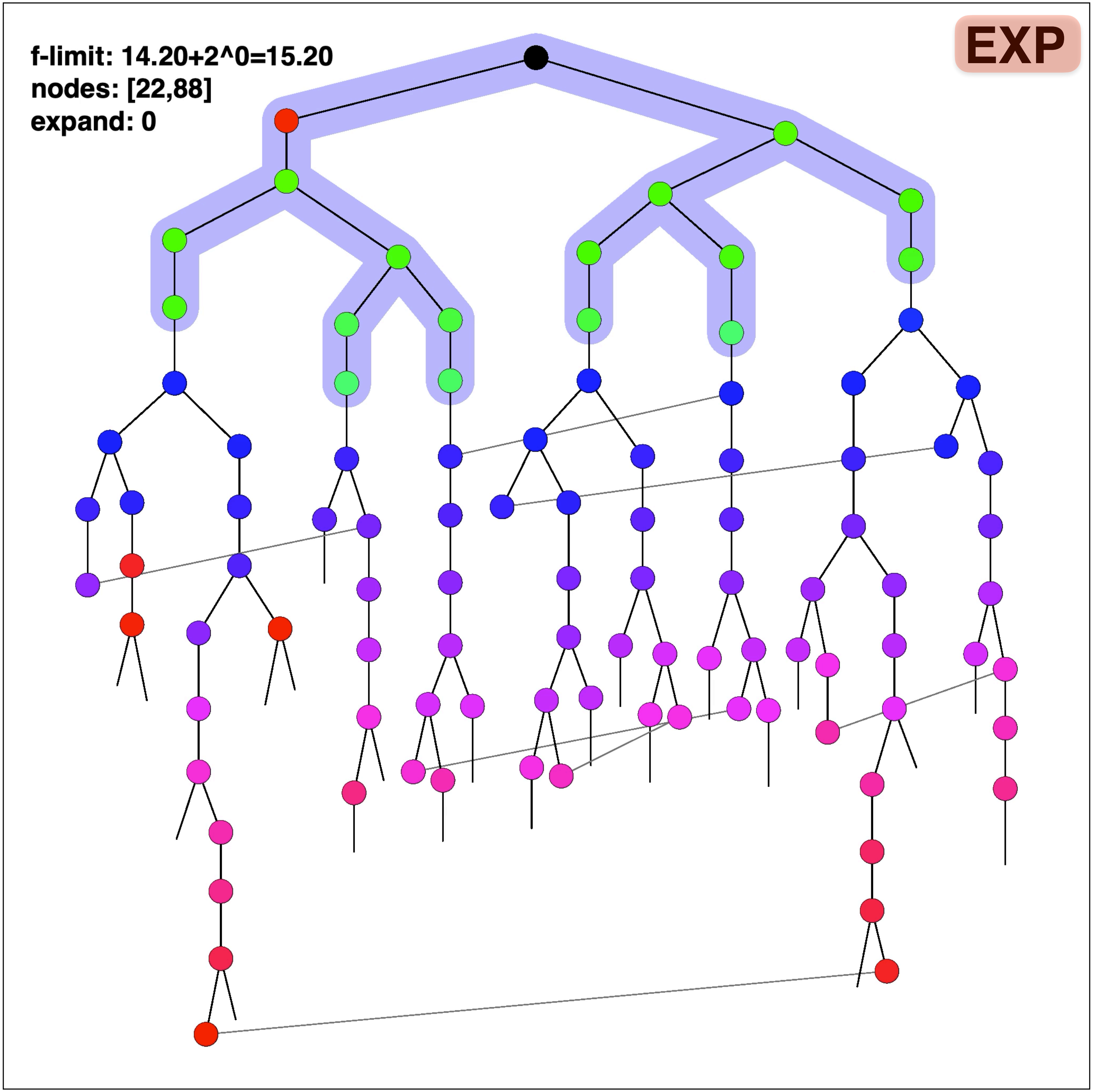


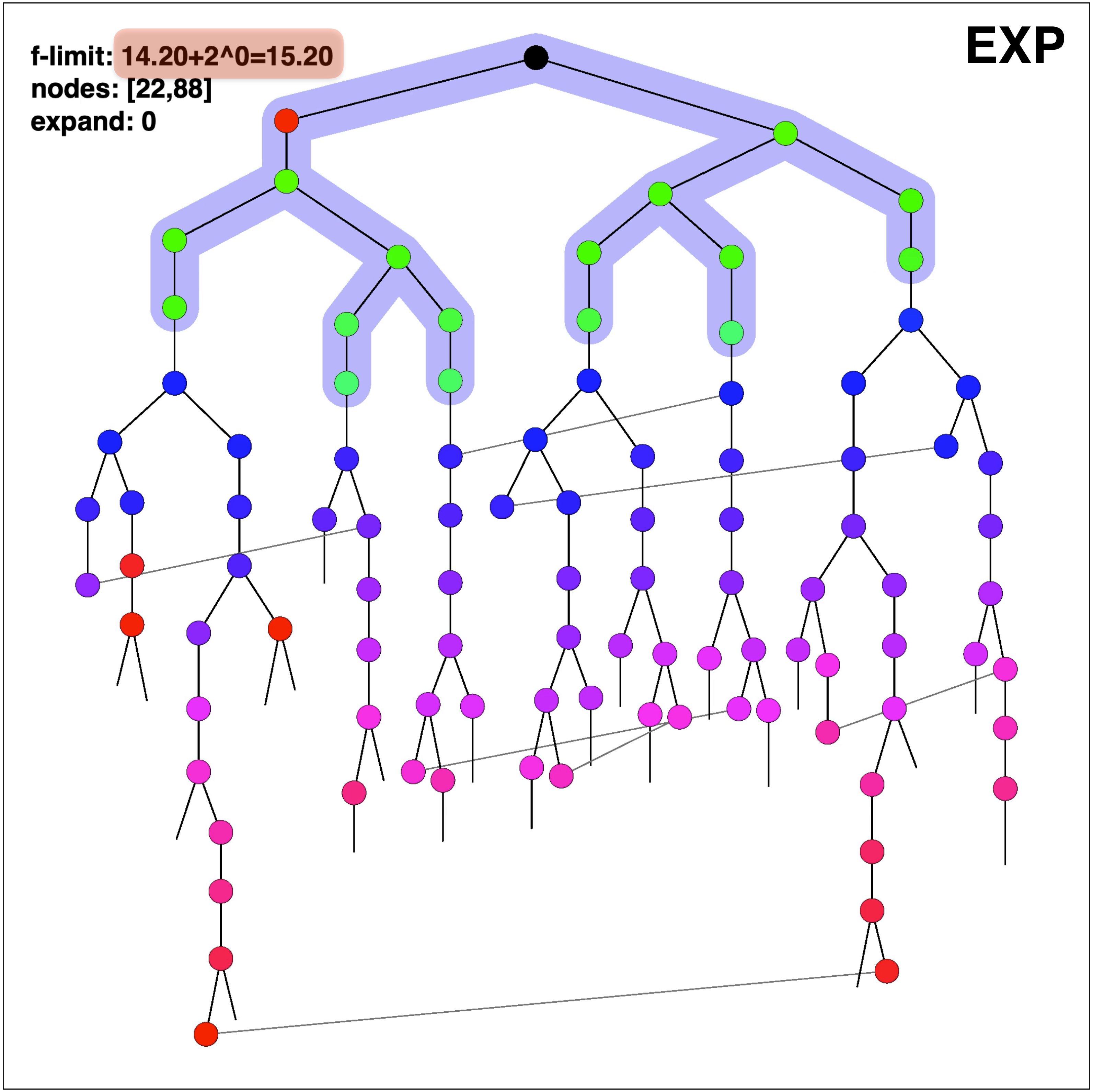
f-limit: 14.00
nodes: [22,∞]
expand: 0
IDA*

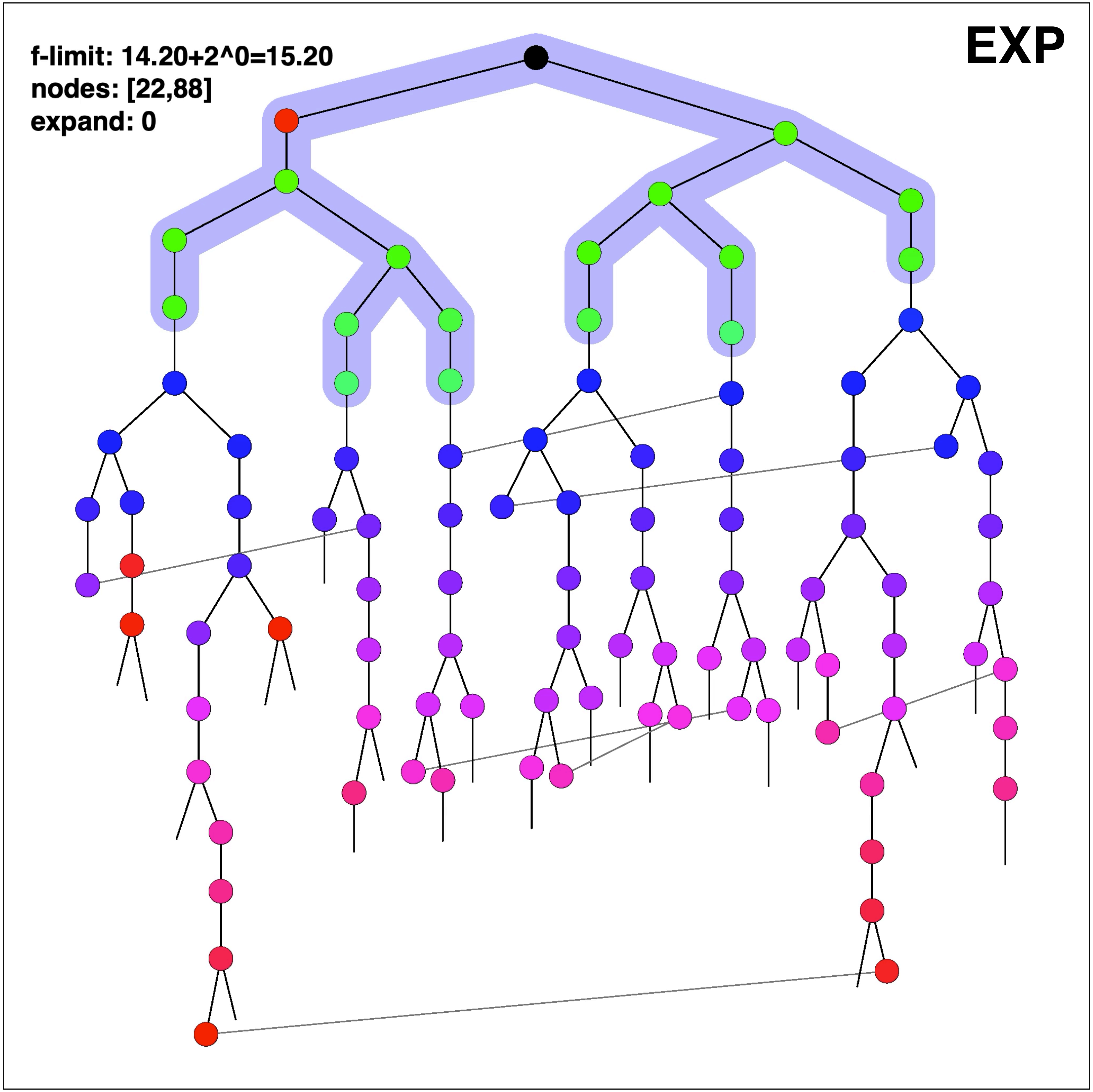
IDA*



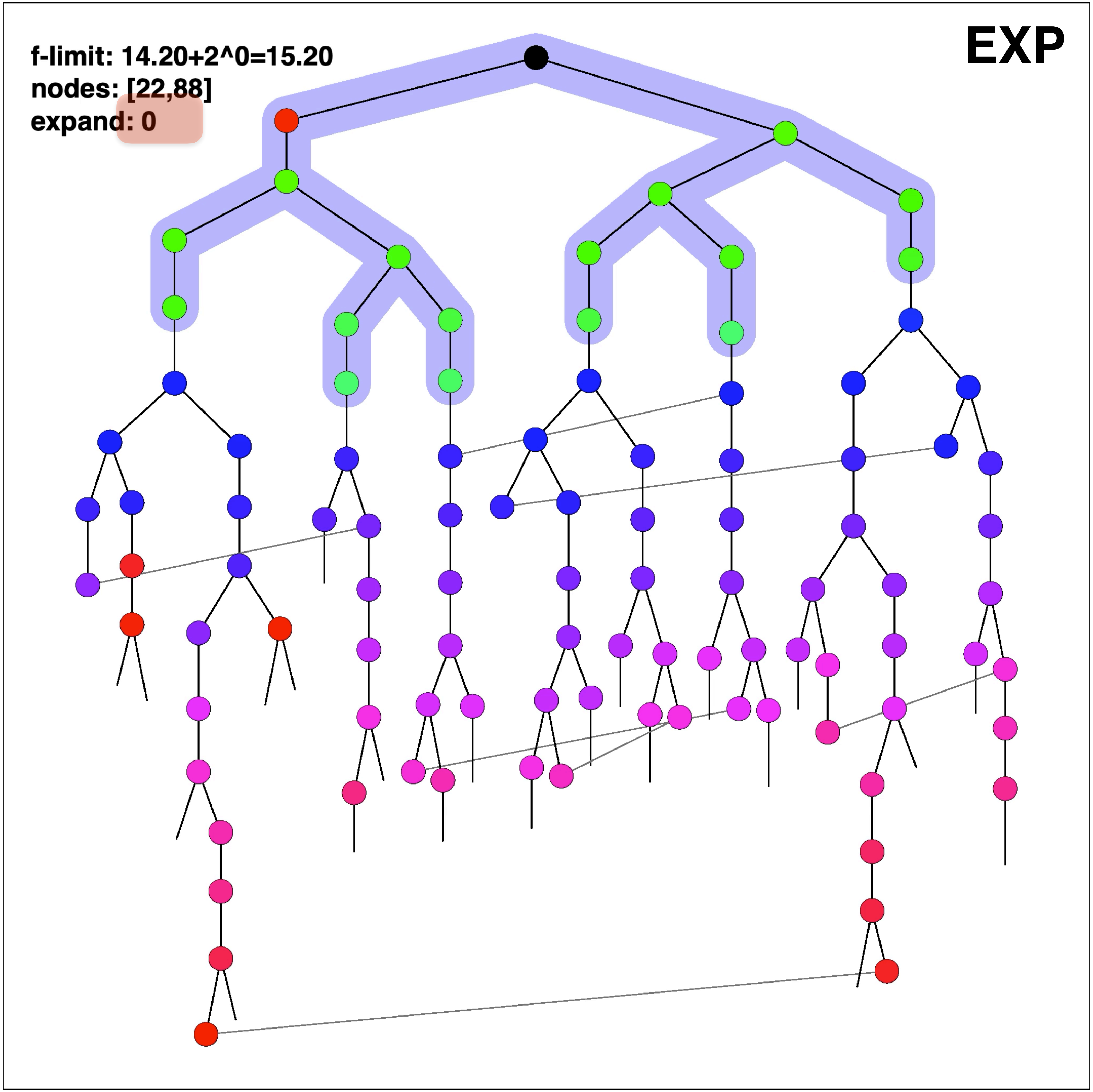

f-limit: 14.20+2^0=15.20
nodes: [22,88]
expand: 0
EXP

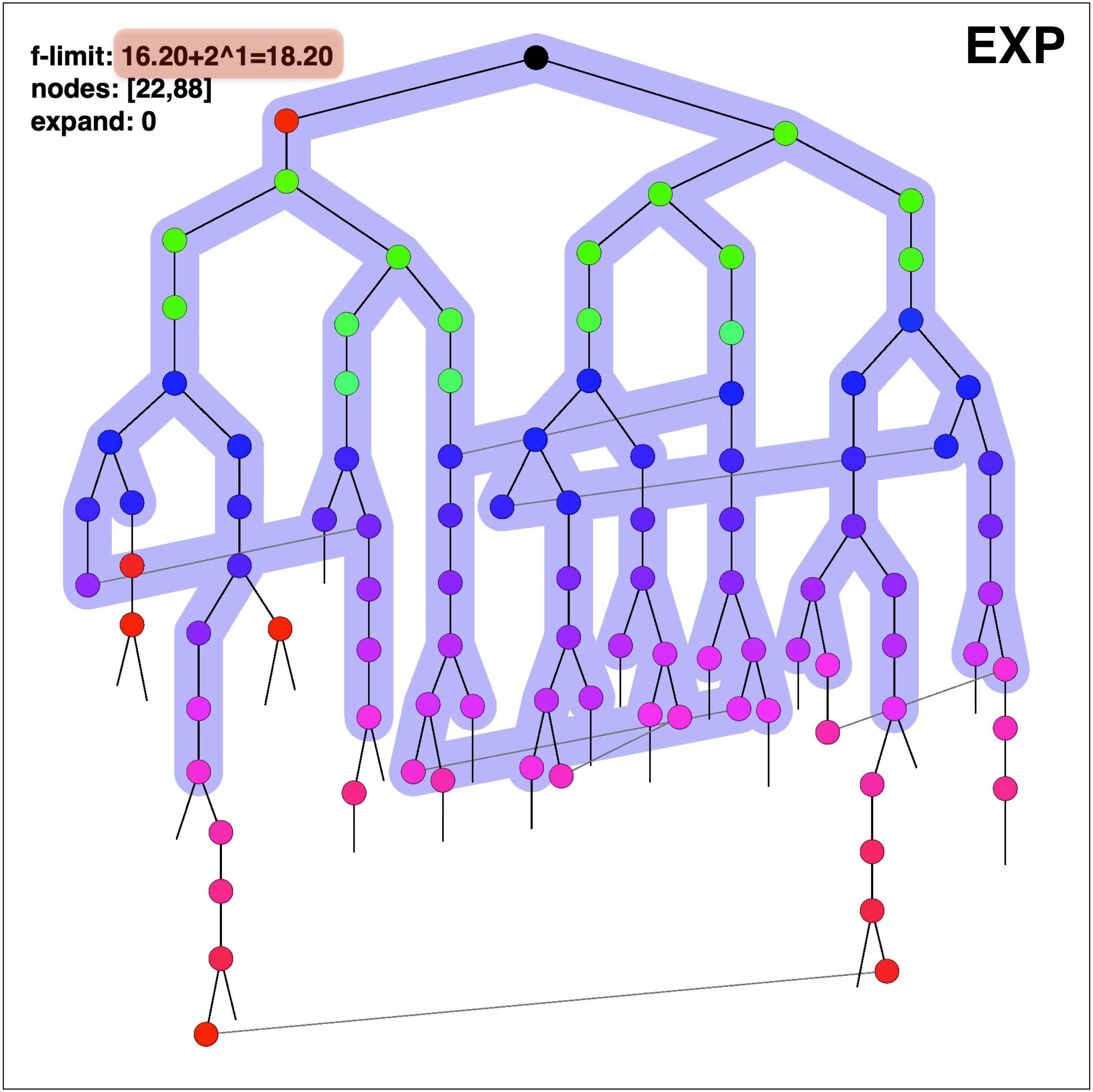

f-limit: 14.20+2^0=15.20
nodes: [22,88]
expand: 0
EXP

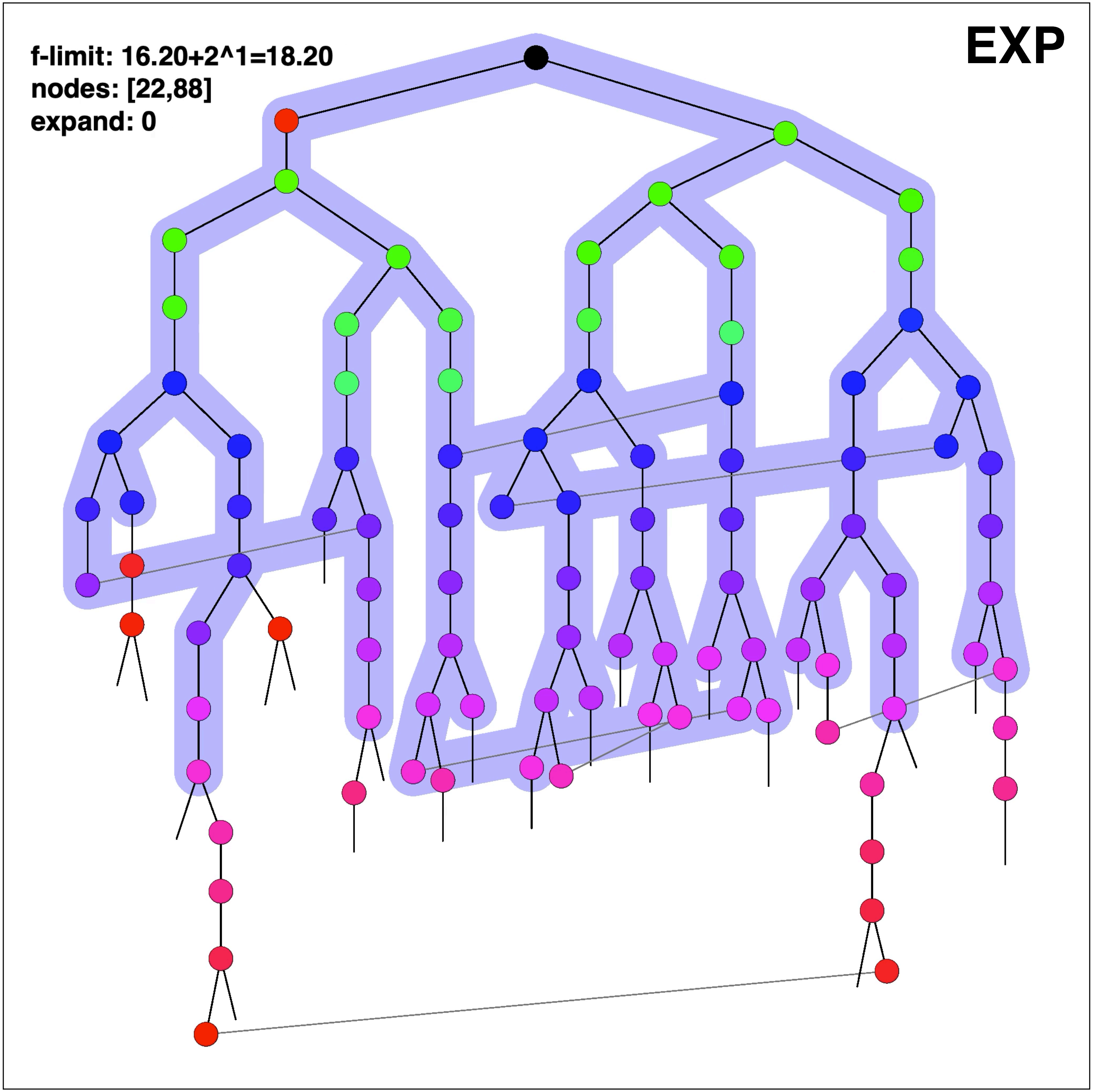

f-limit: 14.20+2^0=15.20
nodes: [22,88]
expand: 0
EXP

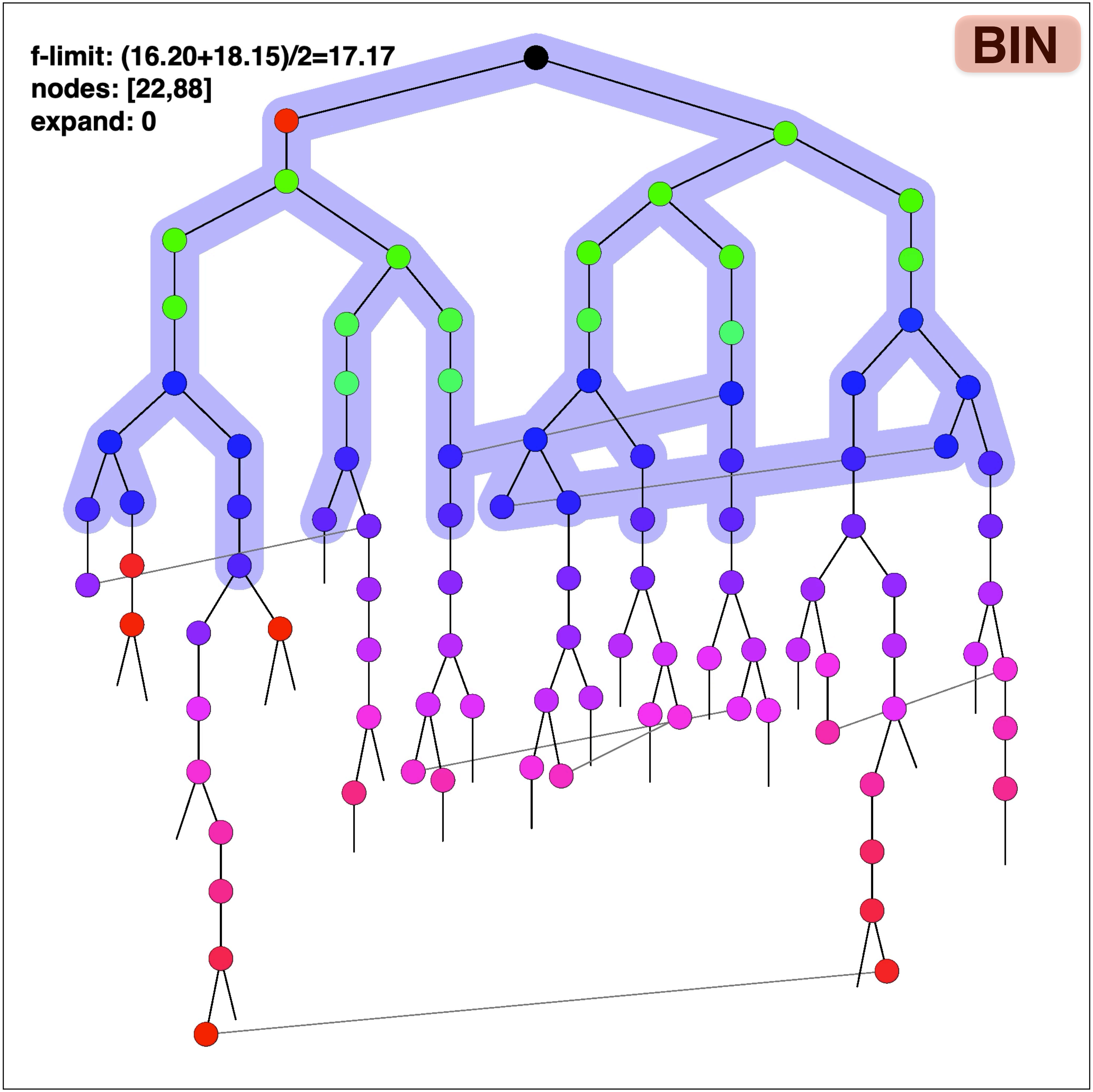


f-limit: 14.20+2^0=15.20
nodes: [22,88]
expand: 0
EXP


EXP
f-limit: 14.20+2^0=15.20
nodes: [22,88]
expand: 0


f-limit: 16.20+2^1=18.20
nodes: [22,88]
expand: 0
EXP

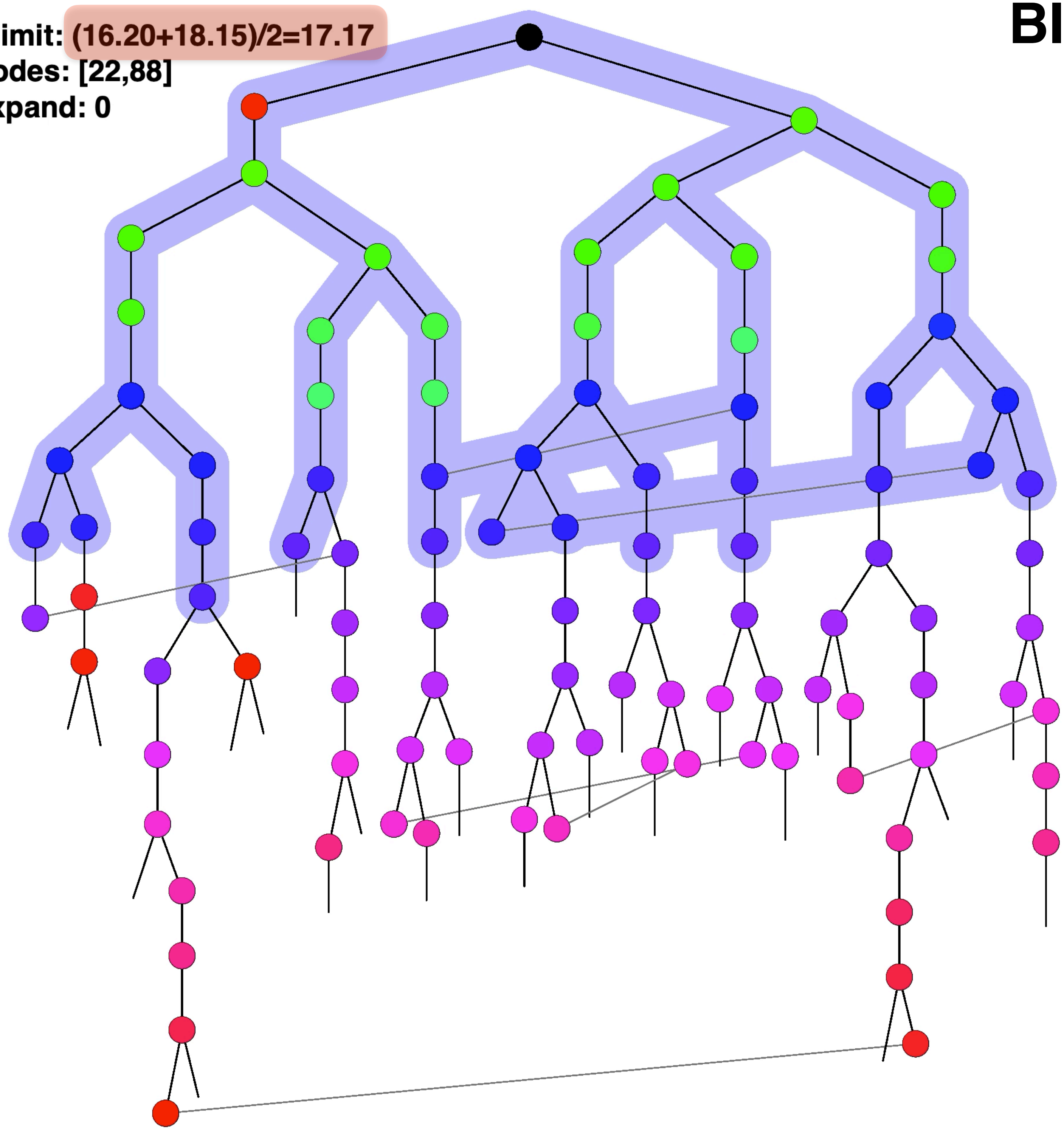

f-limit: 16.20+2^1=18.20
nodes: [22,88]
expand: 0
EXP

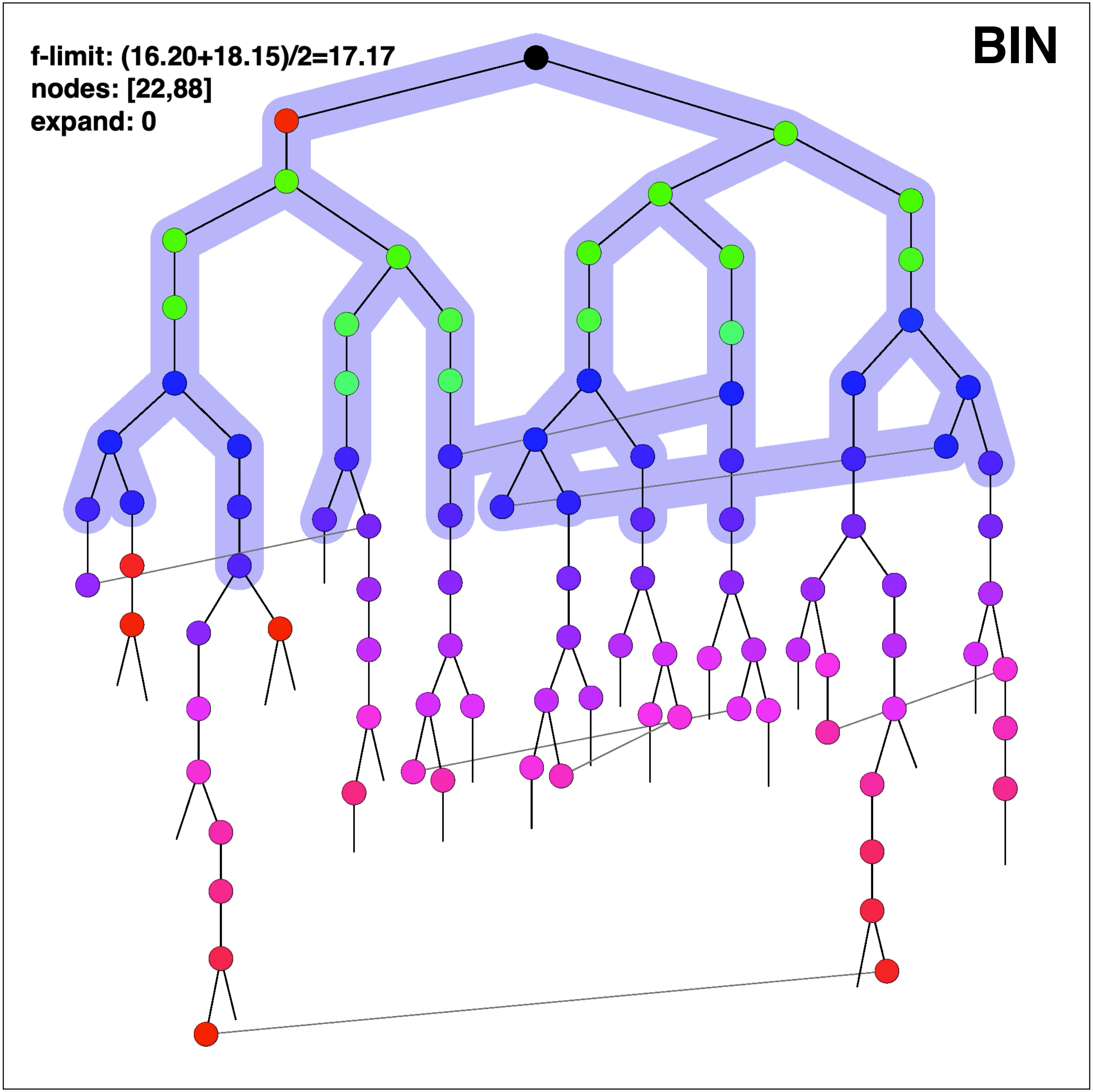


f-limit: (16.20+18.15)/2=17.17
nodes: [22,88]
expand: 0
BIN

f-limit: (16.20+18.15)/2=17.17
nodes: [22,88]
expand: 0

BIN





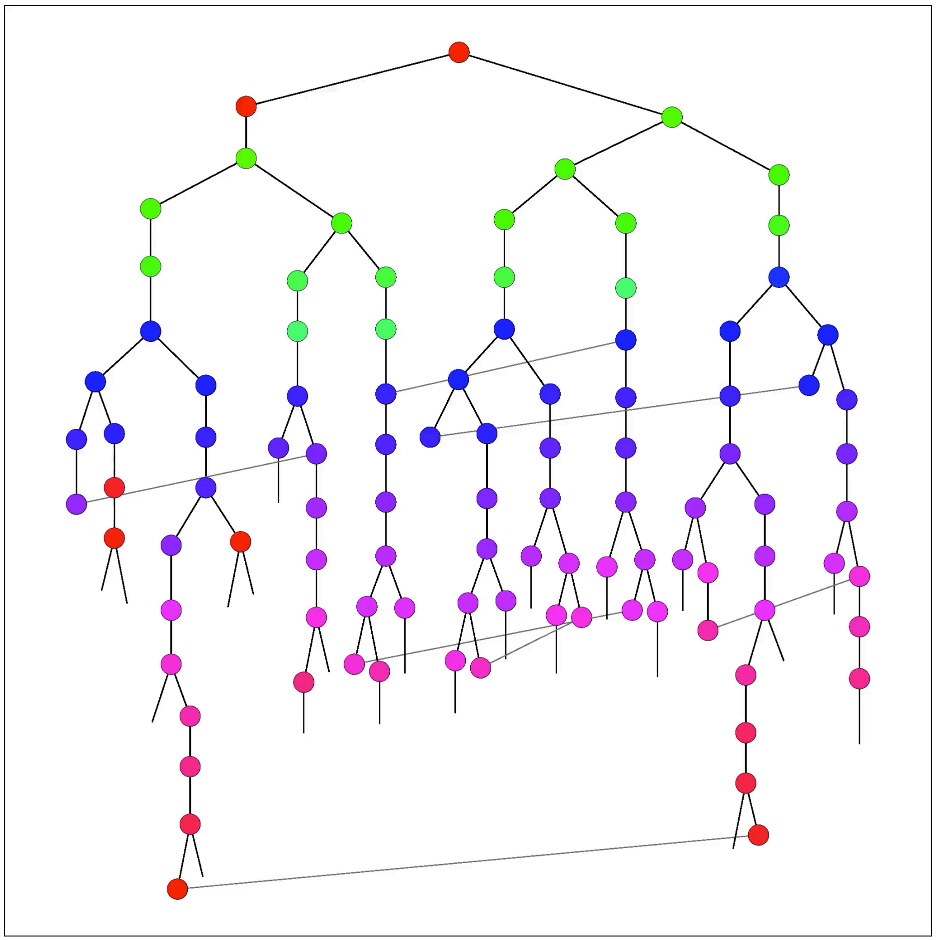
f-limit: (16.20+18.15)/2=17.17
nodes: [22,88]
expand: 0
BIN

How does BTS work?

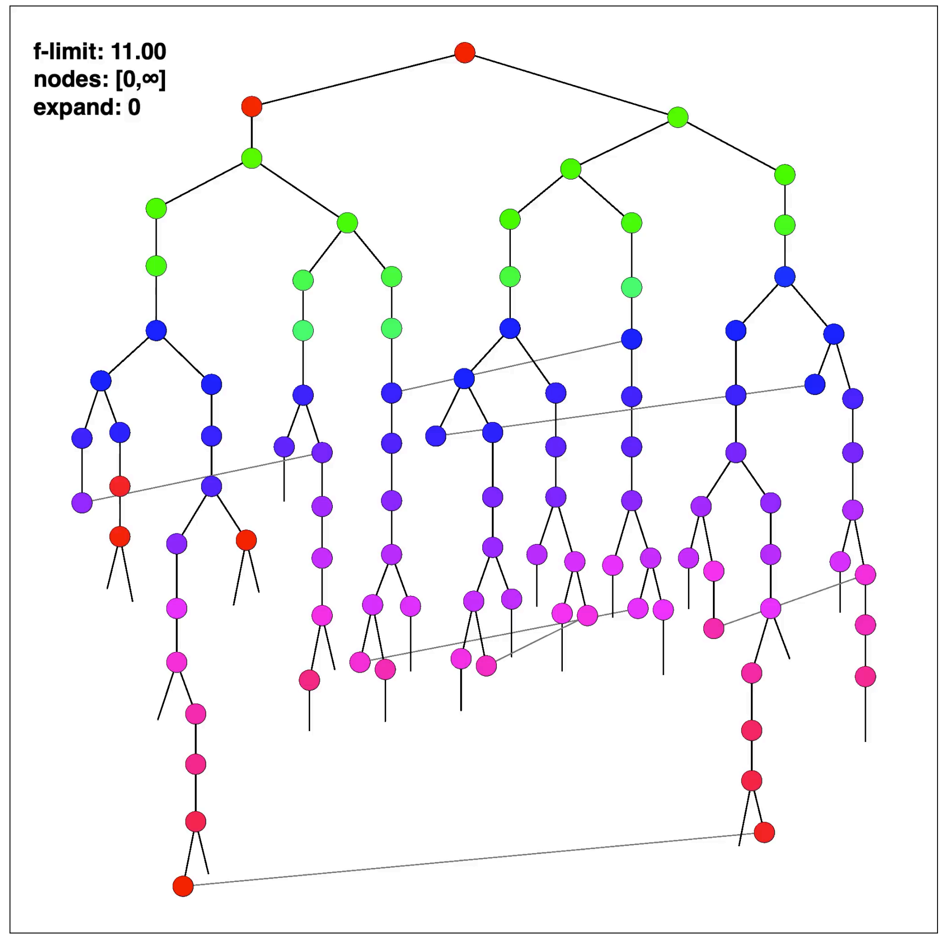
IDA*

With budget $\left\{ \begin{array}{l} \text{Exponential Search} \\ \text{Binary Search} \end{array} \right.$

IDA*

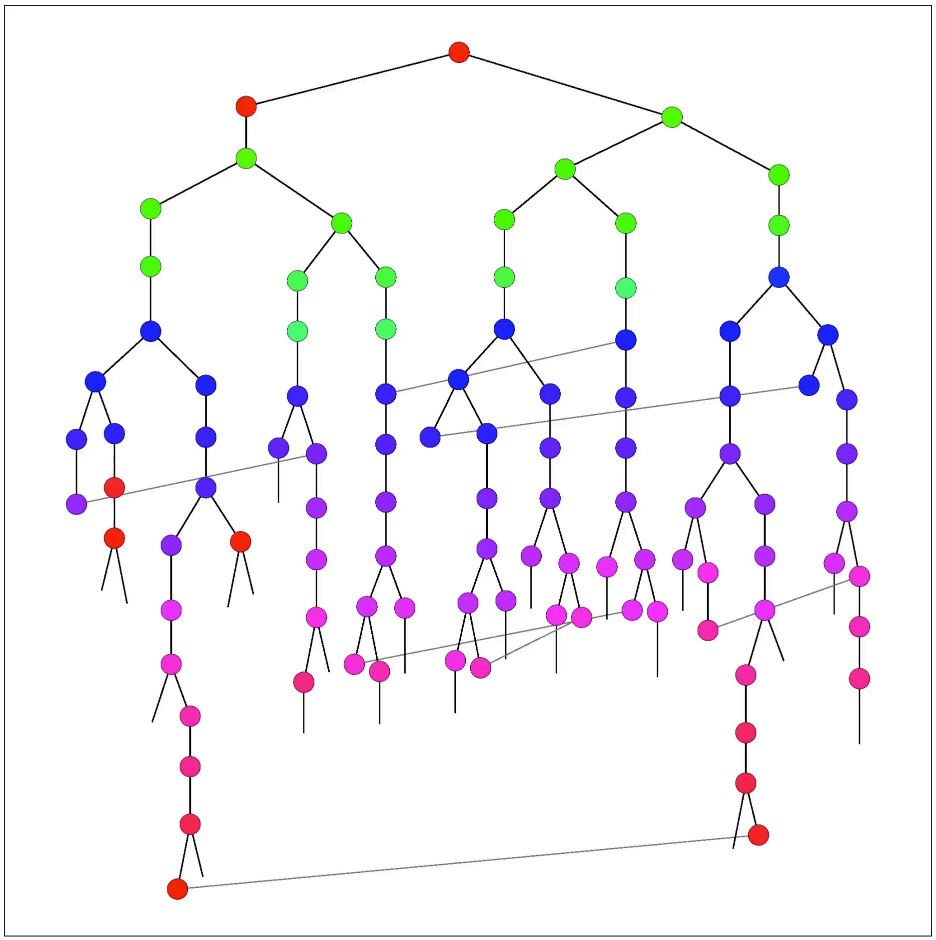


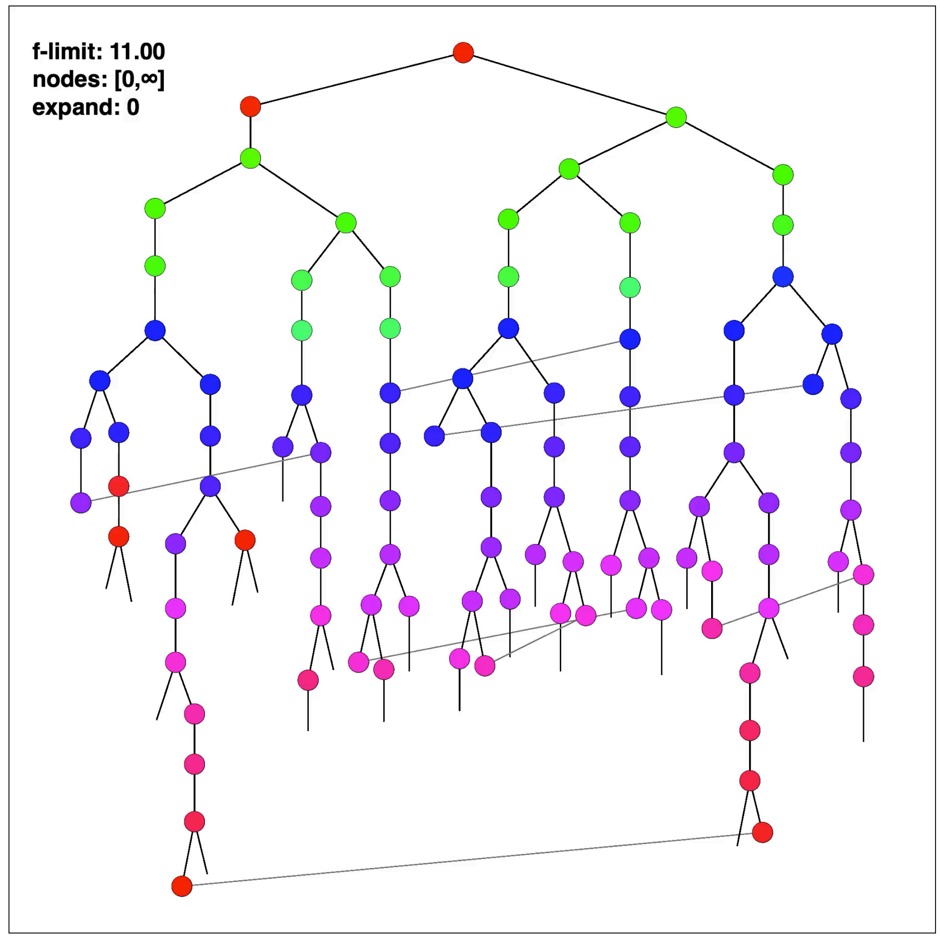
BTS*



IDA*

BTS*





Conclusions

- BTS reduces worst case of IDA*
  - Same performance as IDA* if tree grows exponentially
- IBEX solves similar problems in different contexts

Conclusions

- BTS reduces worst case of IDA*
  - Same performance as IDA* if tree grows exponentially
- IBEX solves similar problems in different contexts

- Demos & videos online:
  - https://www.movingai.com/SAS/
  - https://www.movingai.com/SAS/BTS/