A Brief History and Recent Achievements in Bidirectional Search

Nathan R. Sturtevant, University of Denver
Ariel Felner, Ben Gurion University

Other major contributors:
Jingwei Chen, University of Denver
Eshed Shaham, Ben Gurion University
Robert Holte, University of Alberta
Sandra Zilles, University of Regina

UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

Key Related Work

- 1959 - Dijkstra’s Algorithm

Key Related Work

- 1959 - Dijkstra’s Algorithm
- 1966 - Bidirectional Search (Nicholson & Doran)

Key Related Work

- 1959 - Dijkstra's Algorithm
- 1966 - Bidirectional Search (Nicholson & Doran)
- 1968 - Heuristic Search (Hart, Nilsson & Raphael)

Key Related Work

- 1959 - Dijkstra's Algorithm
- 1966 - Bidirectional Search (Nicholson & Doran)
- 1968 - Heuristic Search (Hart, Nilsson & Raphael)
- 1969 - Bidirectional Heuristic Search (Pohl)

Key Related Work

- 1959 - Dijkstra's Algorithm
- 1966 - Bidirectional Search (Nicholson & Doran)
- **1968 - Heuristic Search (Hart, Nilsson & Raphael)**
- 1969 - Bidirectional Heuristic Search (Pohl)
- **1985 - A* Theory (Dechter & Pearl)**

Overview

- Bidirectional Theory
  - Eckerle et al, ICAPS 2017
- Optimal algorithm (offline)
  - Shaham et al, SoCS 2017
- Near-optimal algorithm (online)
  - Chen et al, IJCAI 2017

Assumptions

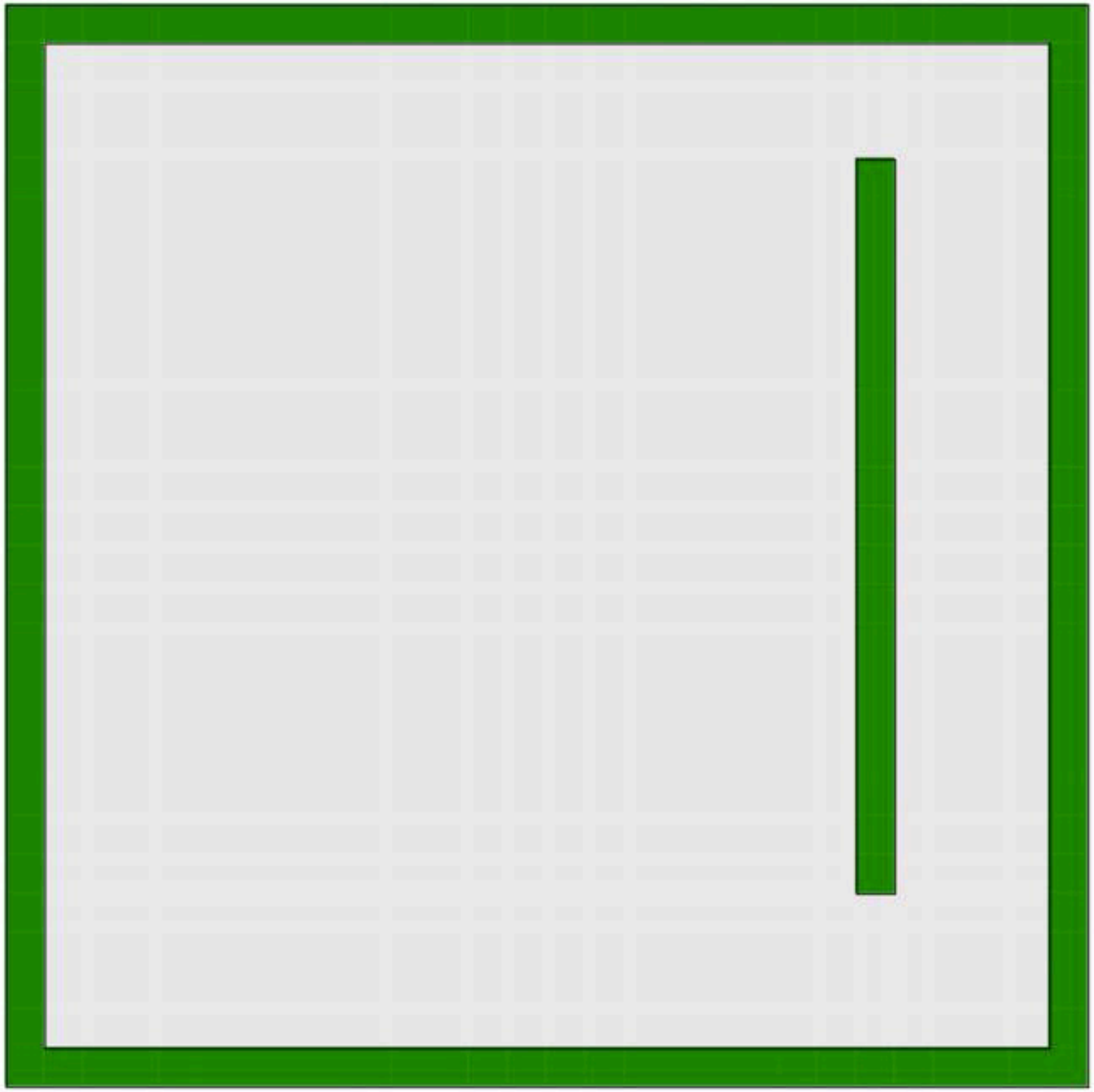
- Front-to-end bidirectional search
- Admissible algorithms
    - Performance with consistent heuristics
- Deterministic, black box algorithm

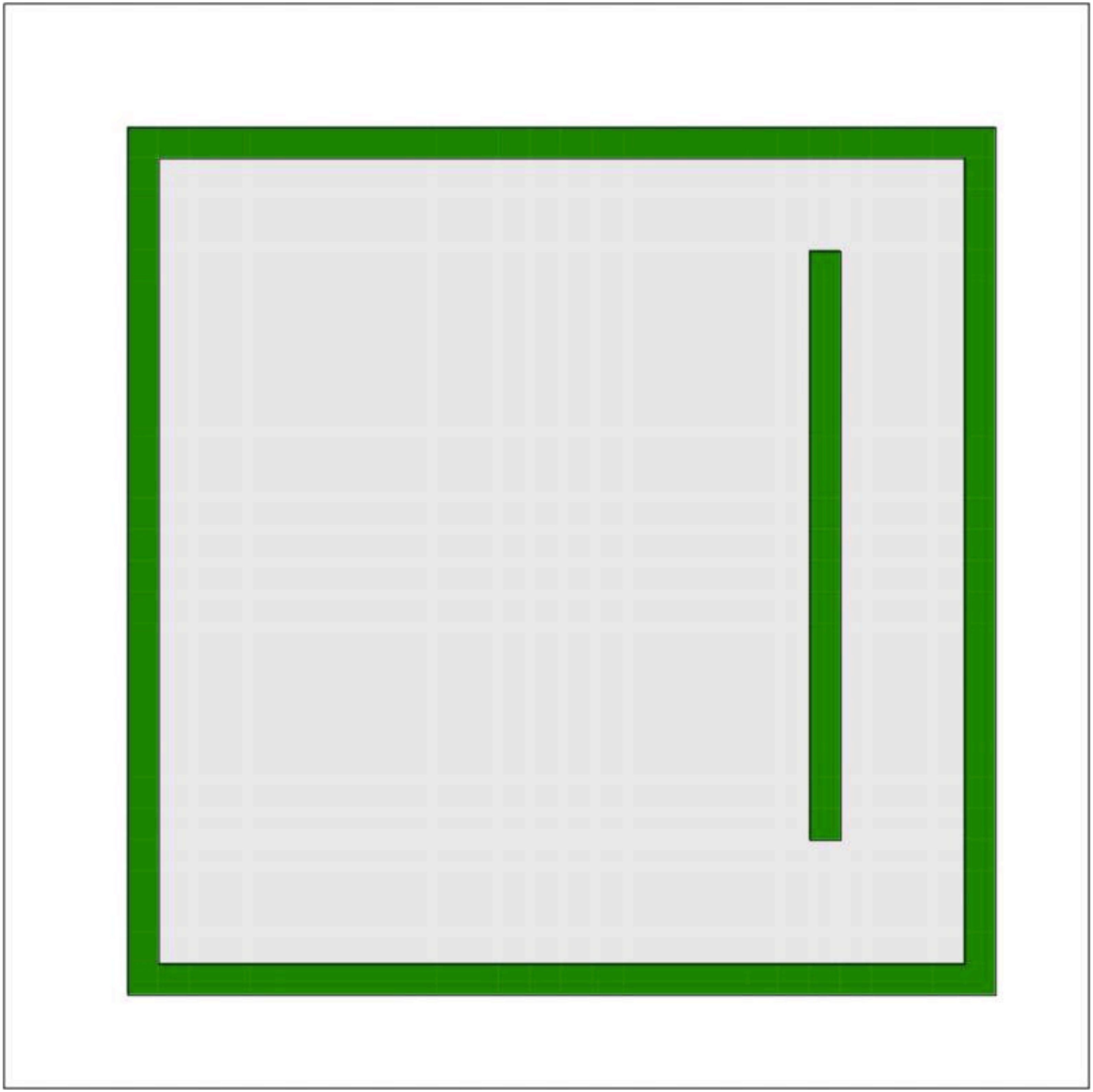
Unidirectional Theory

- ANY admissible unidirectional search algorithm:
  - Must expand ALL states with:
    - $f(s) = g(s) + h(s) < C^*$

- Otherwise we can construct instances on which it won't find the optimal solution

What states must be expanded by *all* bidirectional algorithms?





Conclusion

- Given a single state *s*
  - There exists a bidirectional algorithm that does not expand *s*

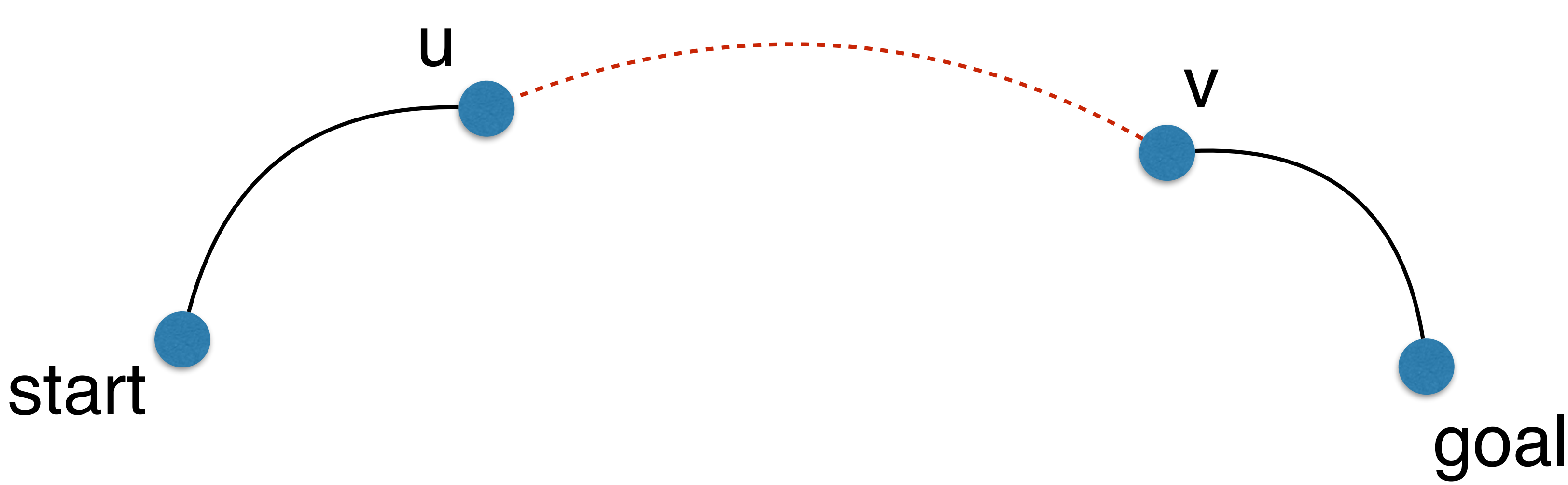
Conclusion

- Given a single state *s*
  - There exists a bidirectional algorithm that does not expand *s*
- Given some pairs of states (*u*, *v*)
  - We can avoid expanding *u*
  - We can avoid expanding *v*
  - We can't avoid expanding BOTH *u* and *v*

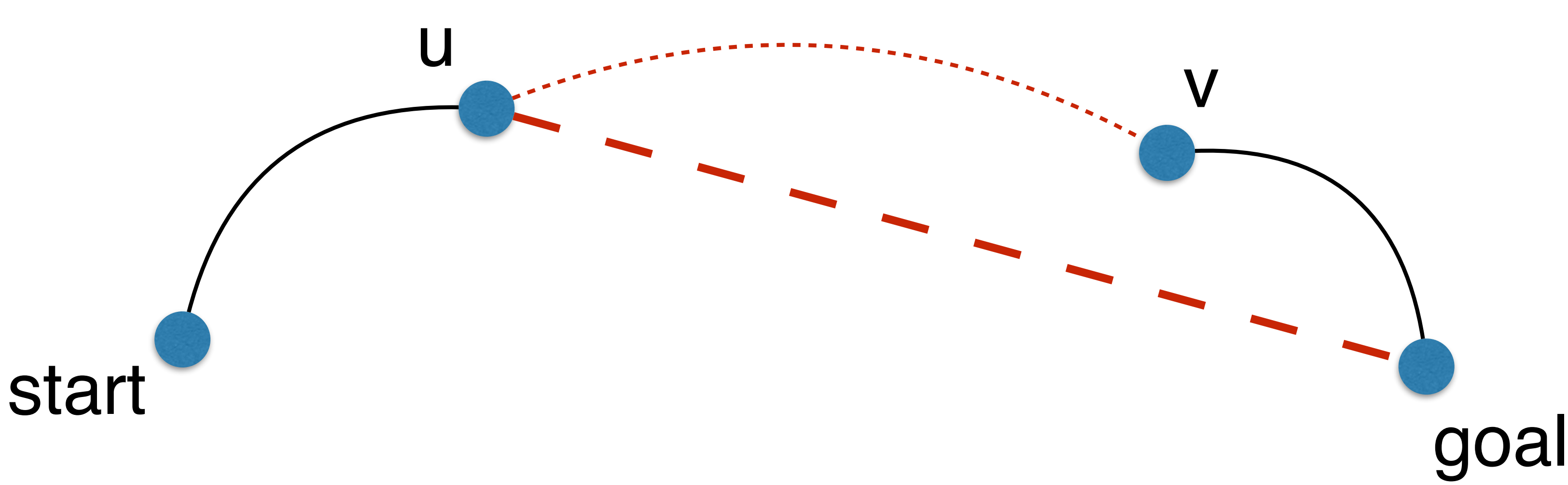
High-Level Picture

High-Level Picture

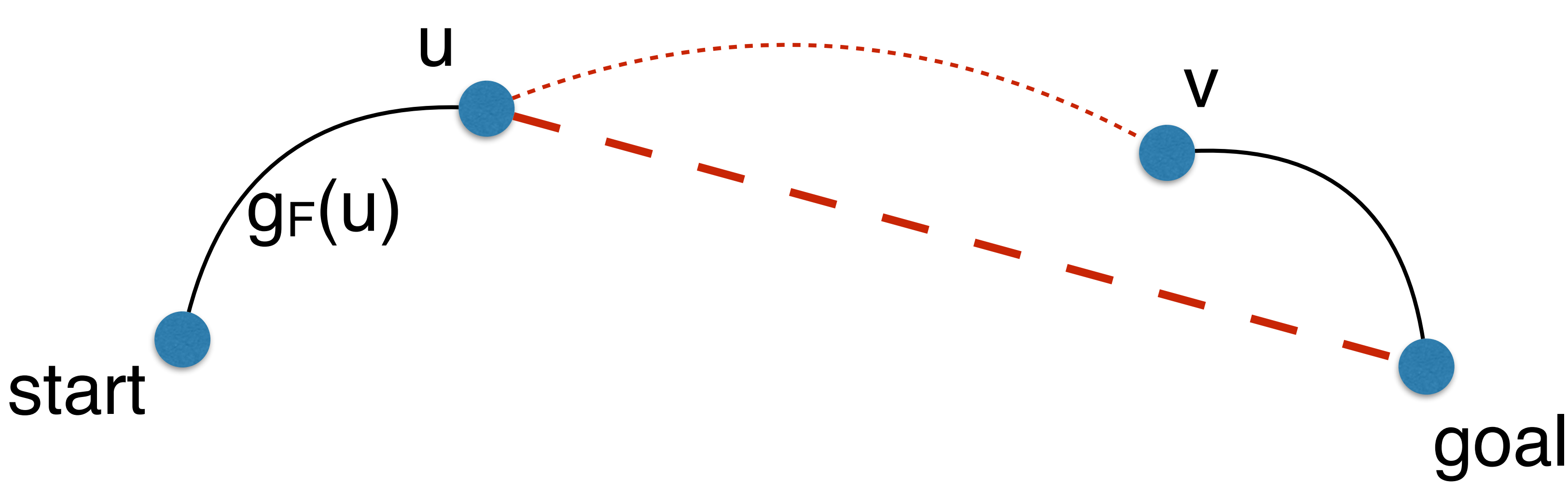
High-Level Picture



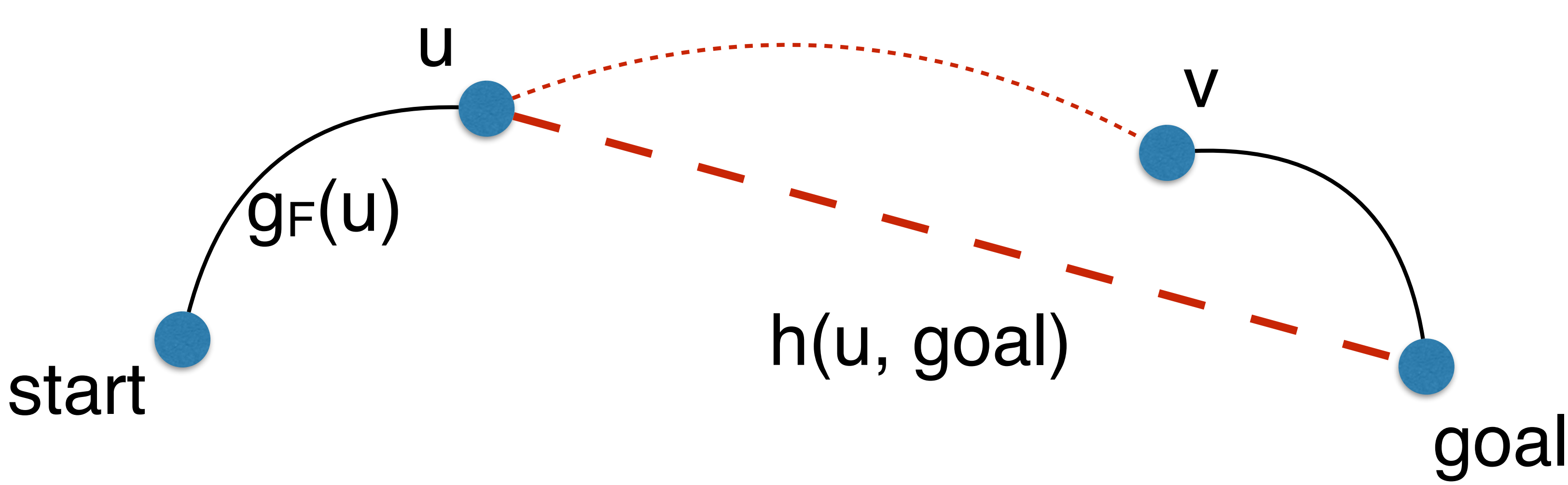
High-Level Picture



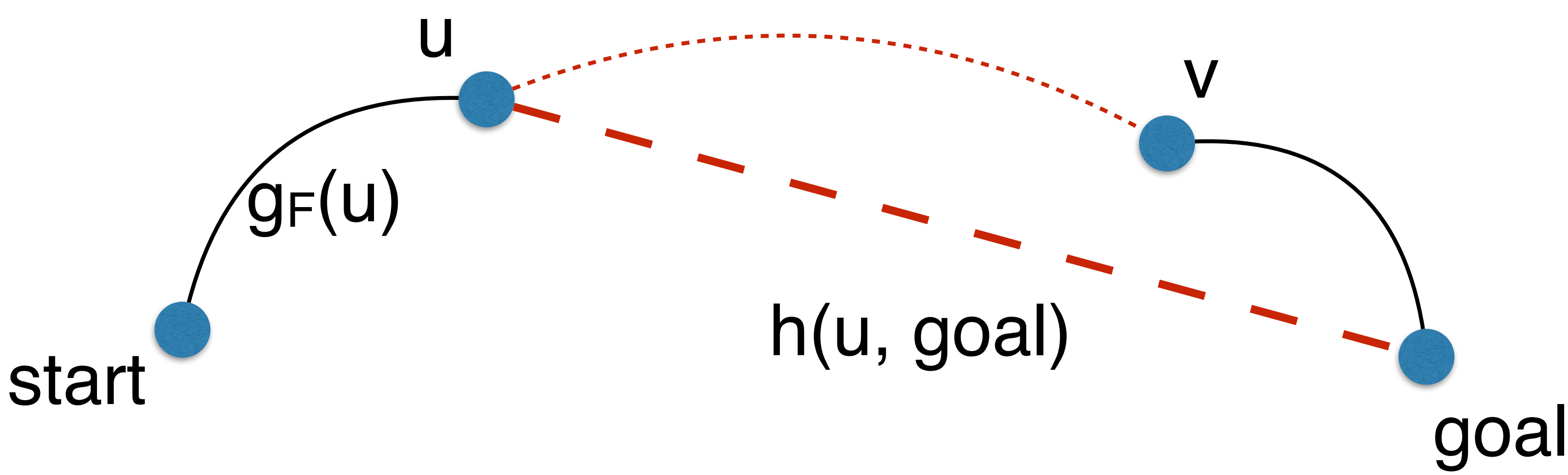
High-Level Picture



High-Level Picture

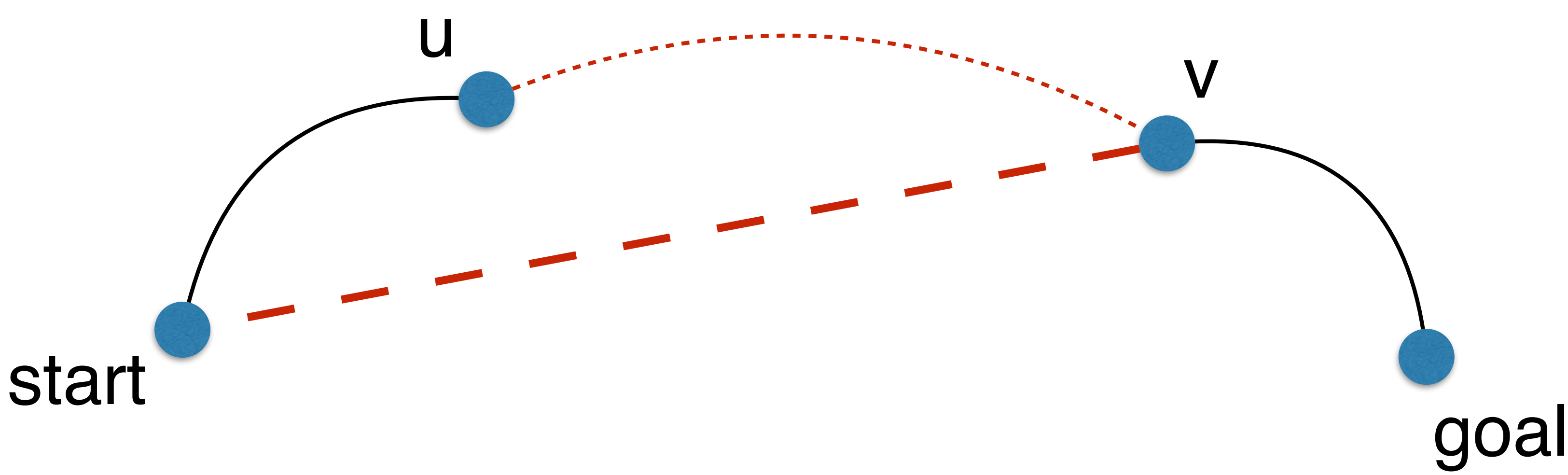


High-Level Picture

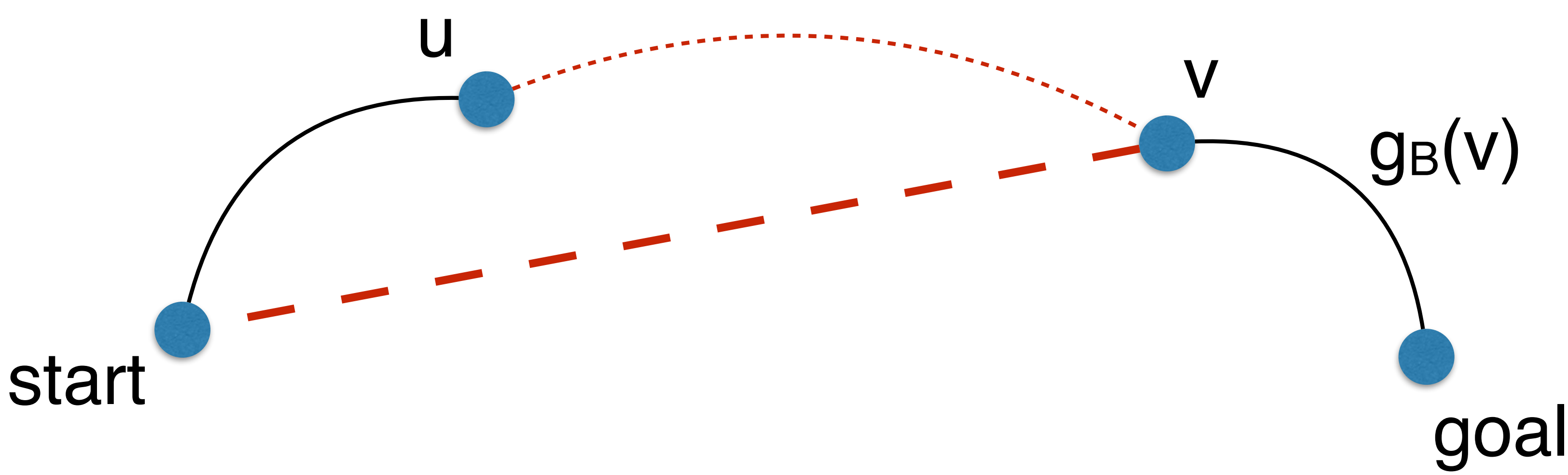


$$f_F(u) = g_F(u) + h(u, \text{goal})$$

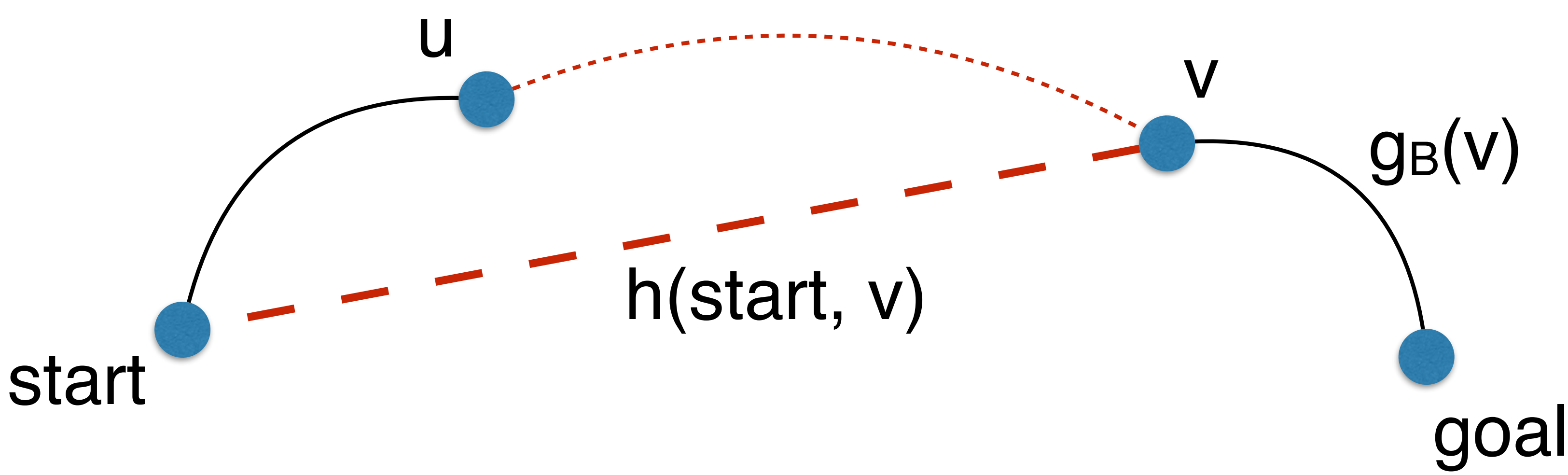
High-Level Picture



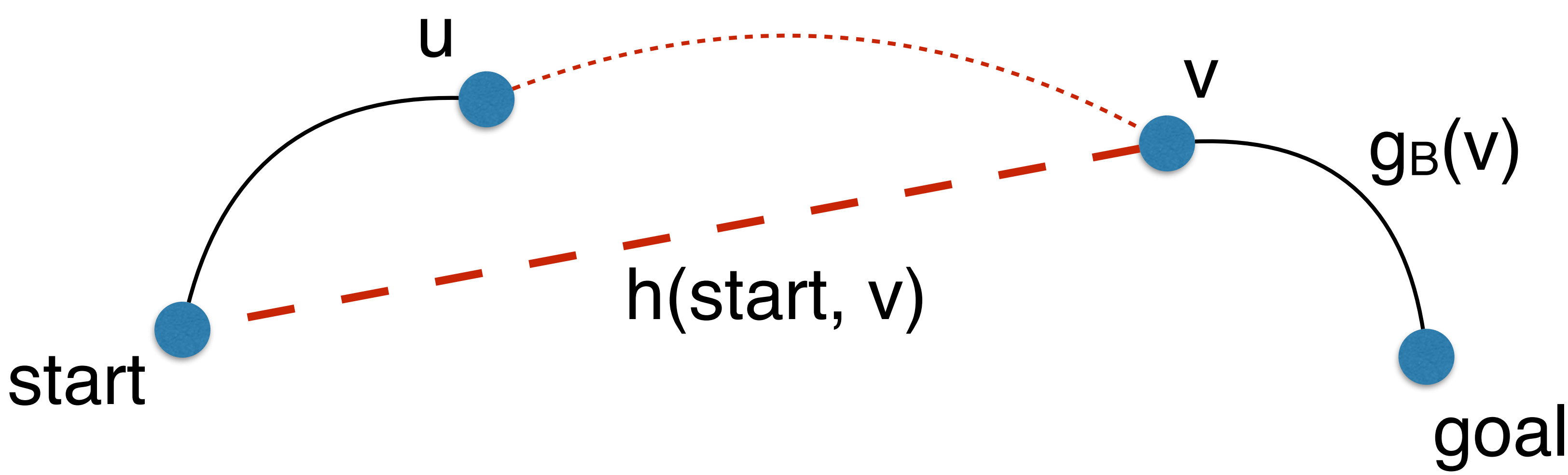
High-Level Picture



High-Level Picture

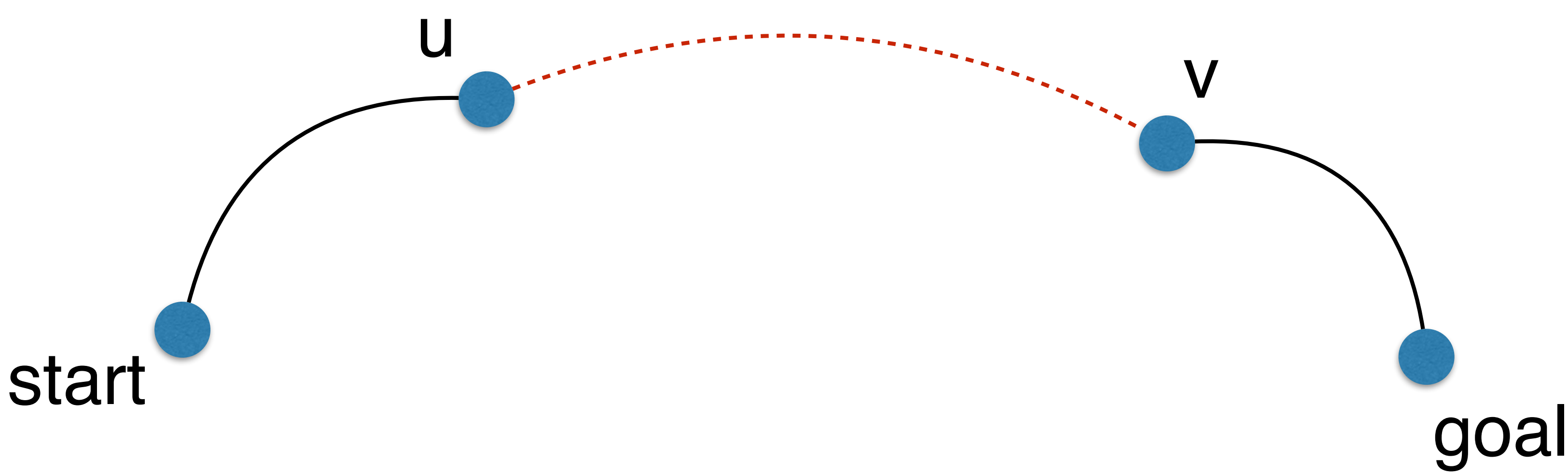


High-Level Picture

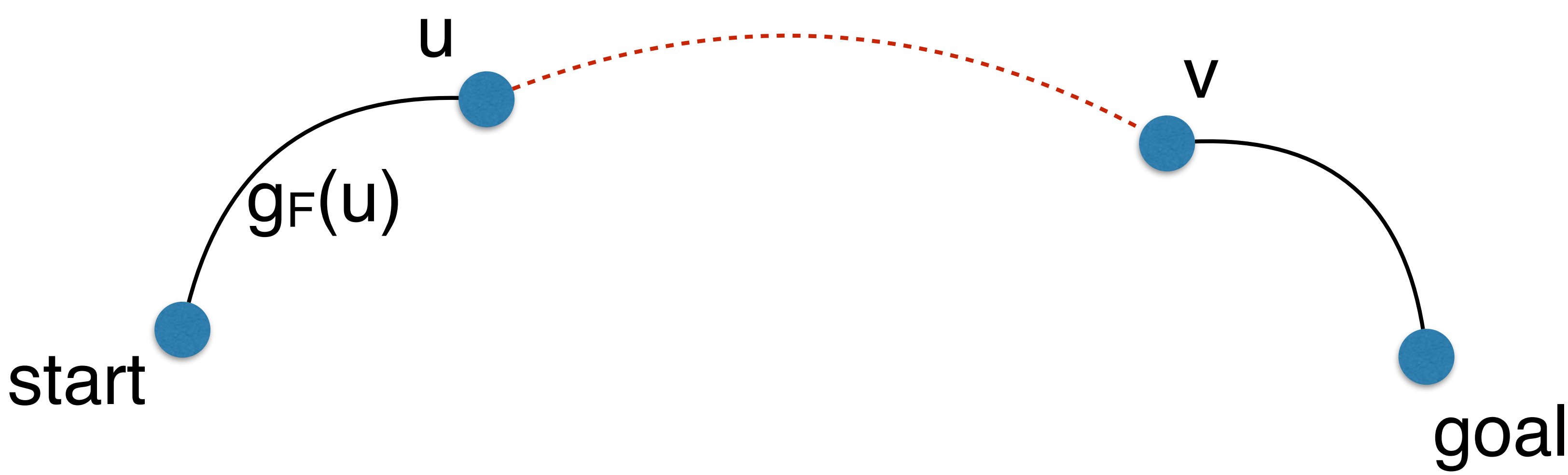


$$f_B(v) = g_B(v) + h(\text{start}, v)$$

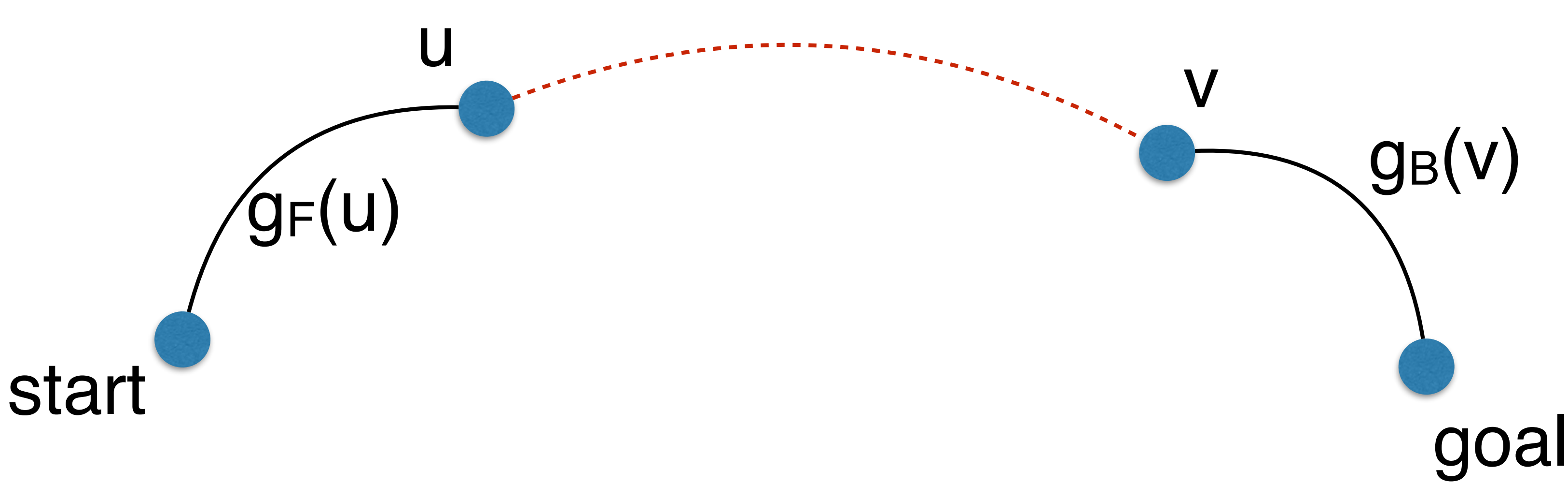
High-Level Picture



High-Level Picture



High-Level Picture



UNIVERSITY of DENVER
DANIEL FELIX RITCHIE SCHOOL OF ENGINEERING & COMPUTER SCIENCE

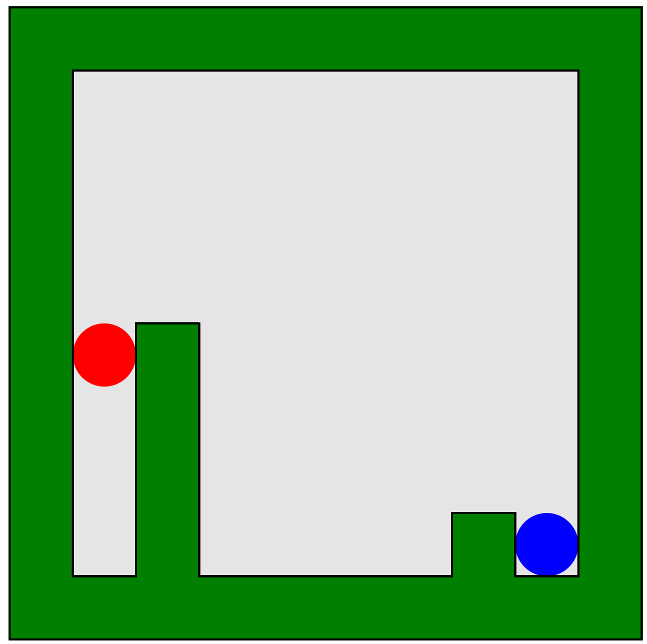
Theorem

f-cost:
estimate of total path length

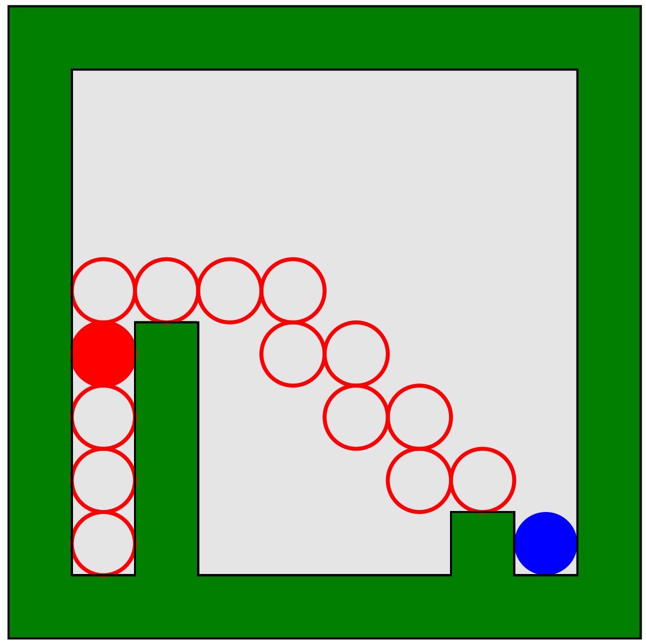
$$\begin{aligned}\mathrm{lb}(u,v) = max(&f_F(u),\\ &f_B(v),\\ &g_F(u) + g_B(v))\end{aligned}$$

- *If lb(u, v) < C* then we must expand either u or v*
- *Leads implicitly to termination conditions*

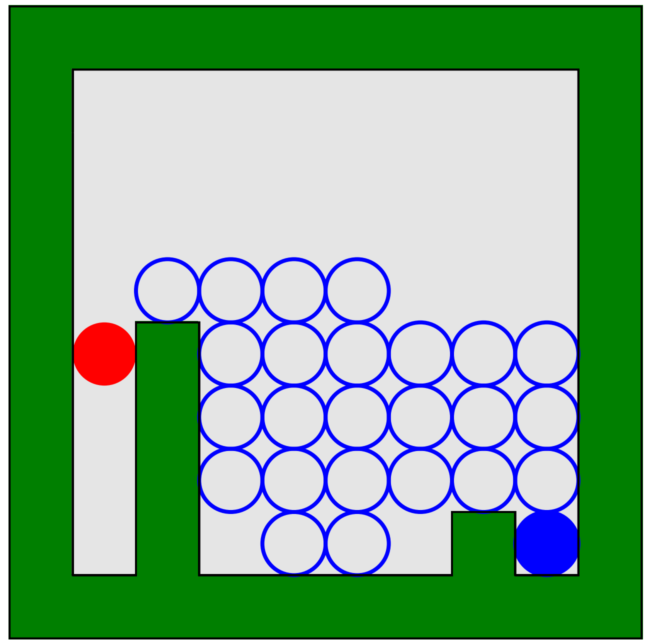
Sufficient Conditions for Node Expansion in Bidirectional Heuristic Search, Jurgen Eckerle, Jingwei Chen, Nathan Sturtevant, Sandra Zilles and Robert Holte, International Conference on Automated Planning and Scheduling (ICAPS), **2017**

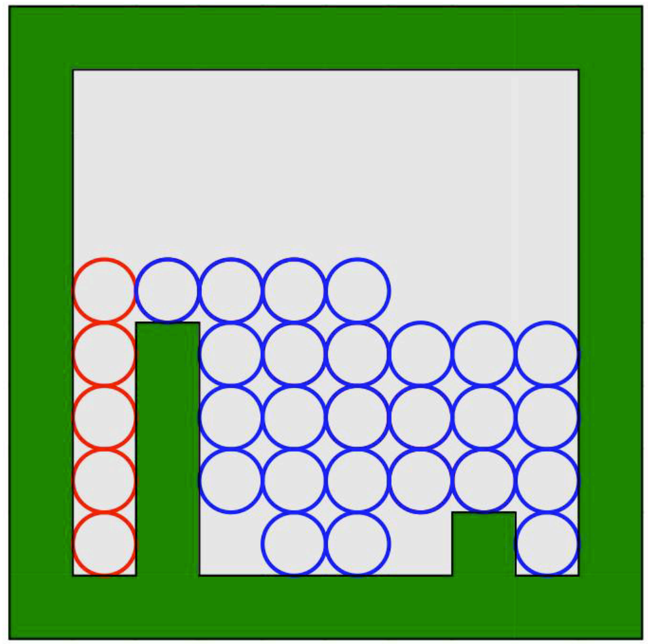


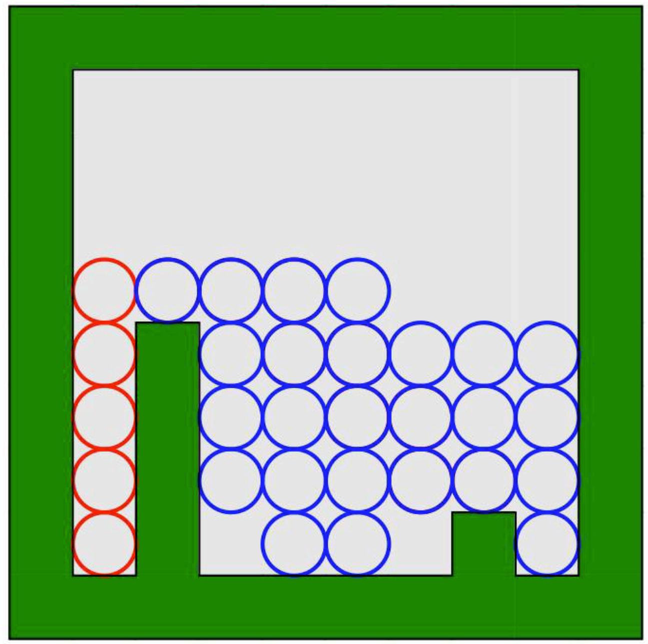
$$f_F(u) < C^*$$



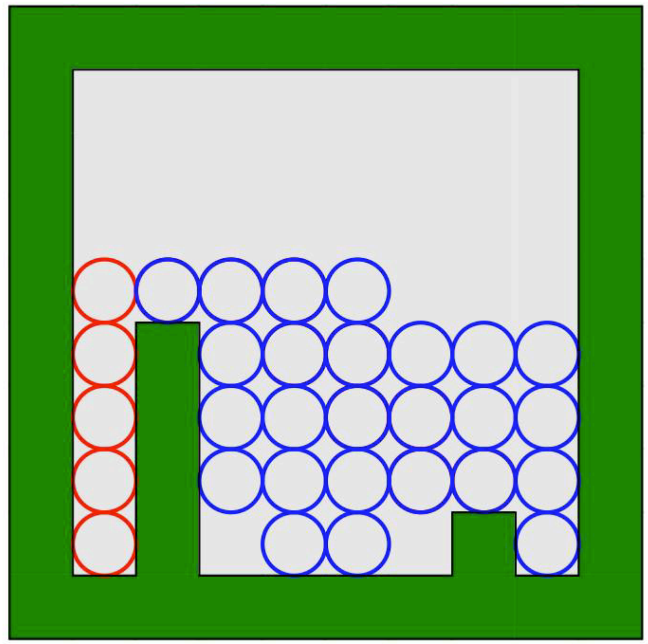
$$f_B(v) < C^*$$



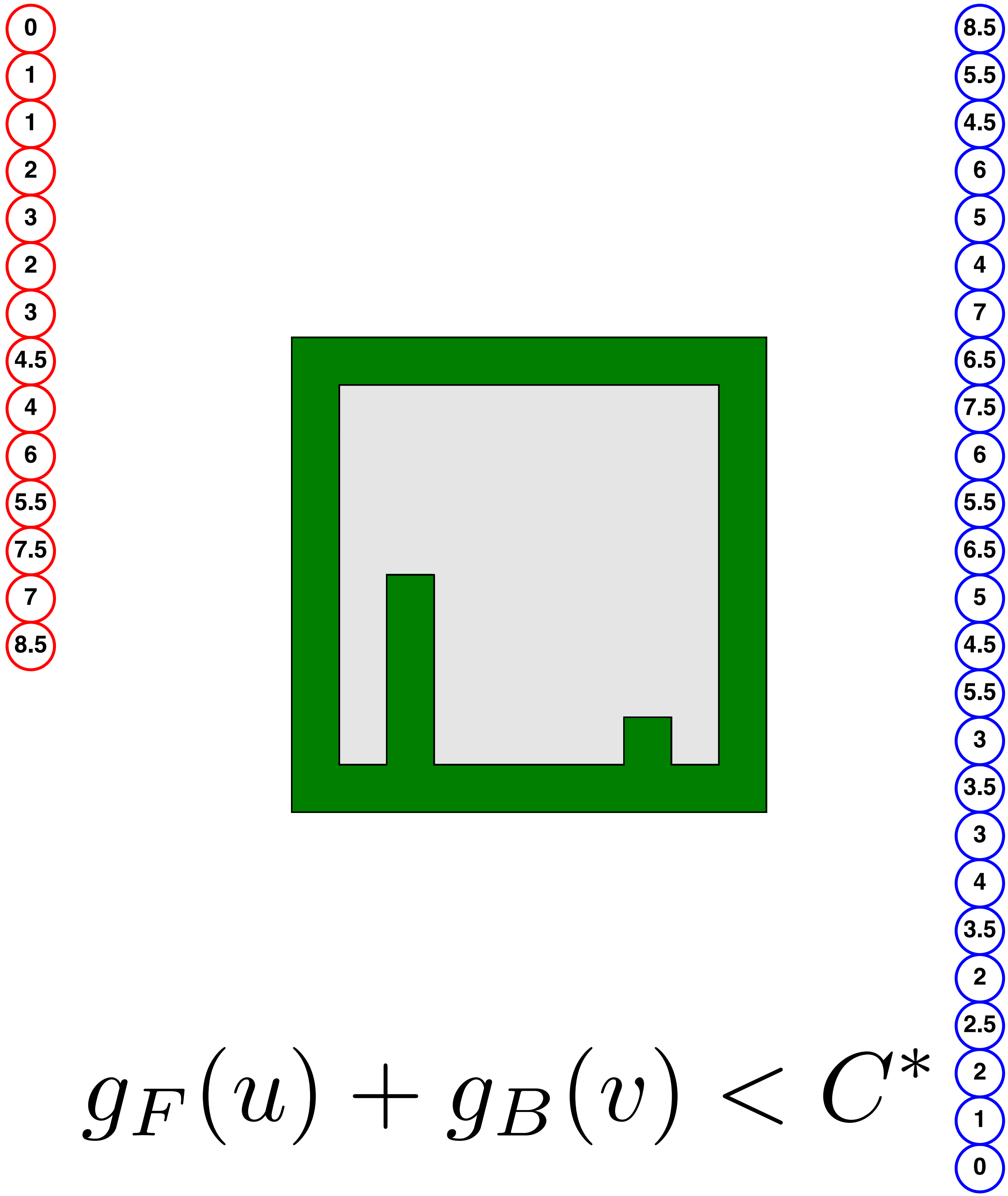




$$g_F(u) + g_B(v) < C^*$$

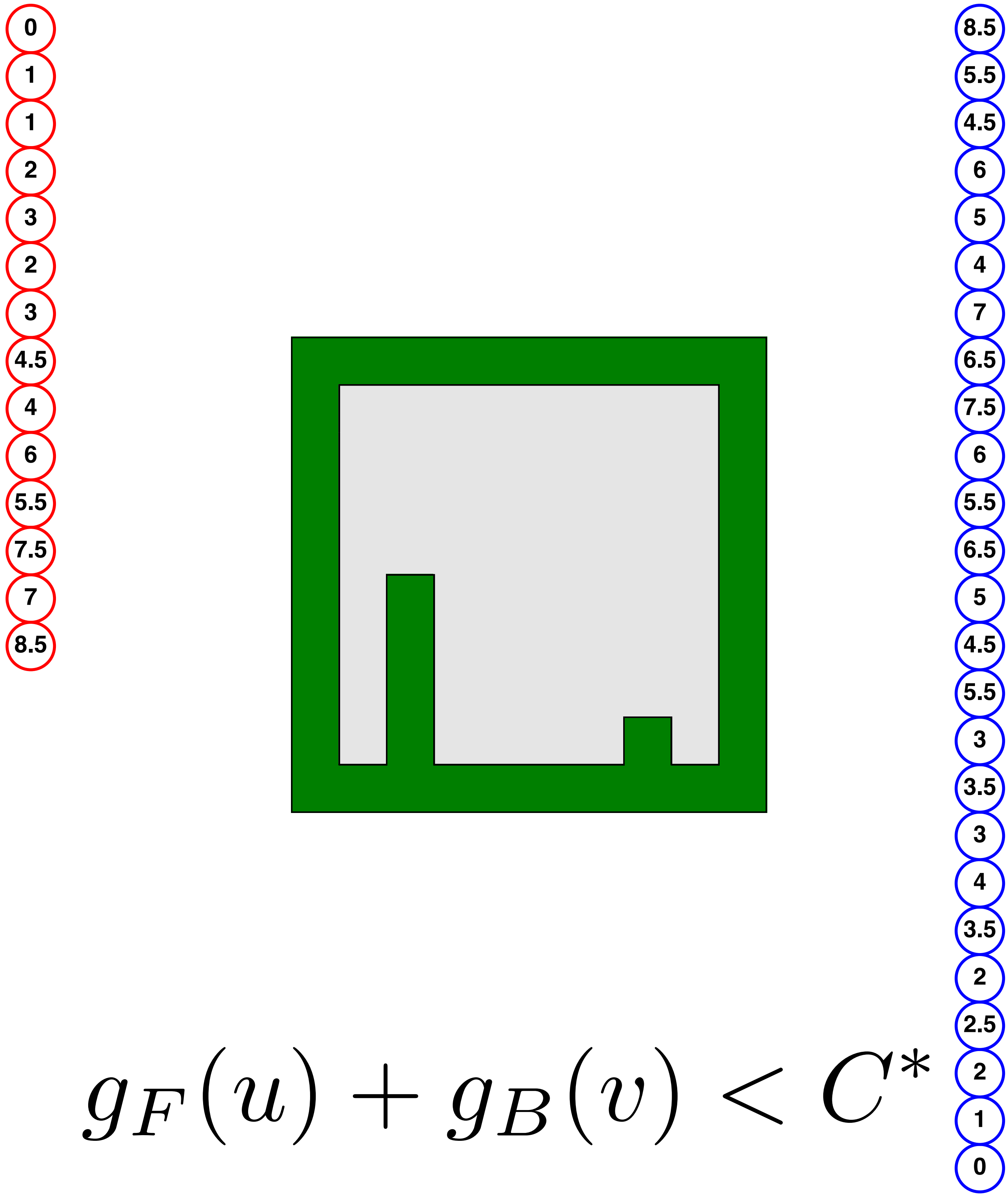


$$g_F(u) + g_B(v) < C^*$$


0
1
1
2
3
2
3
4.5
4
6
5.5
7.5
7
8.5
8.5
5.5
4.5
6
5
4
7
6.5
7.5
6
5.5
6.5
5
4.5
5.5
3
3.5
3
4
3.5
2
2.5
2
1
0

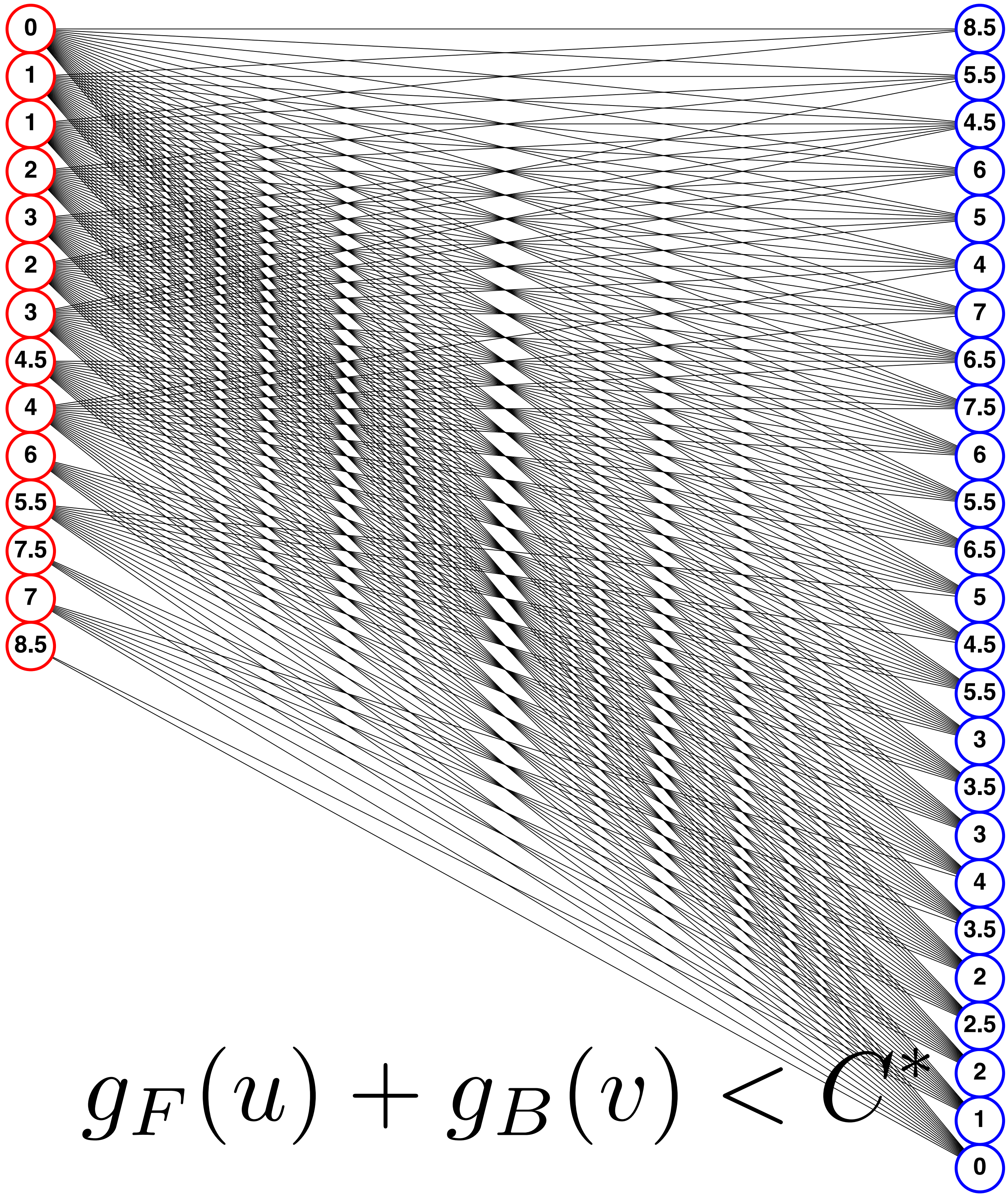
$$g_F(u) + g_B(v) < C^*$$

C* = 10.5



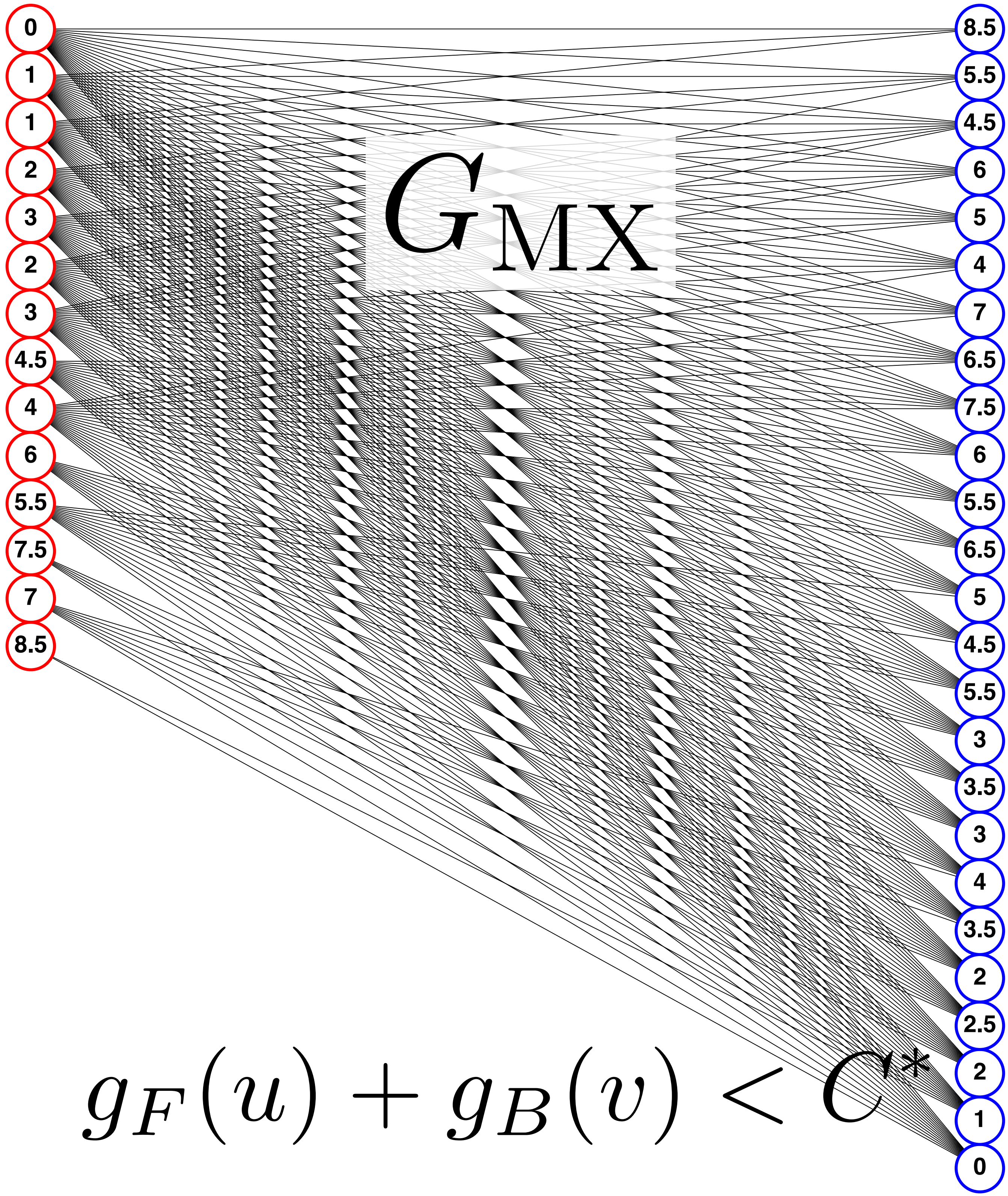
$$g_F(u) + g_B(v) < C^*$$

C* = 10.5

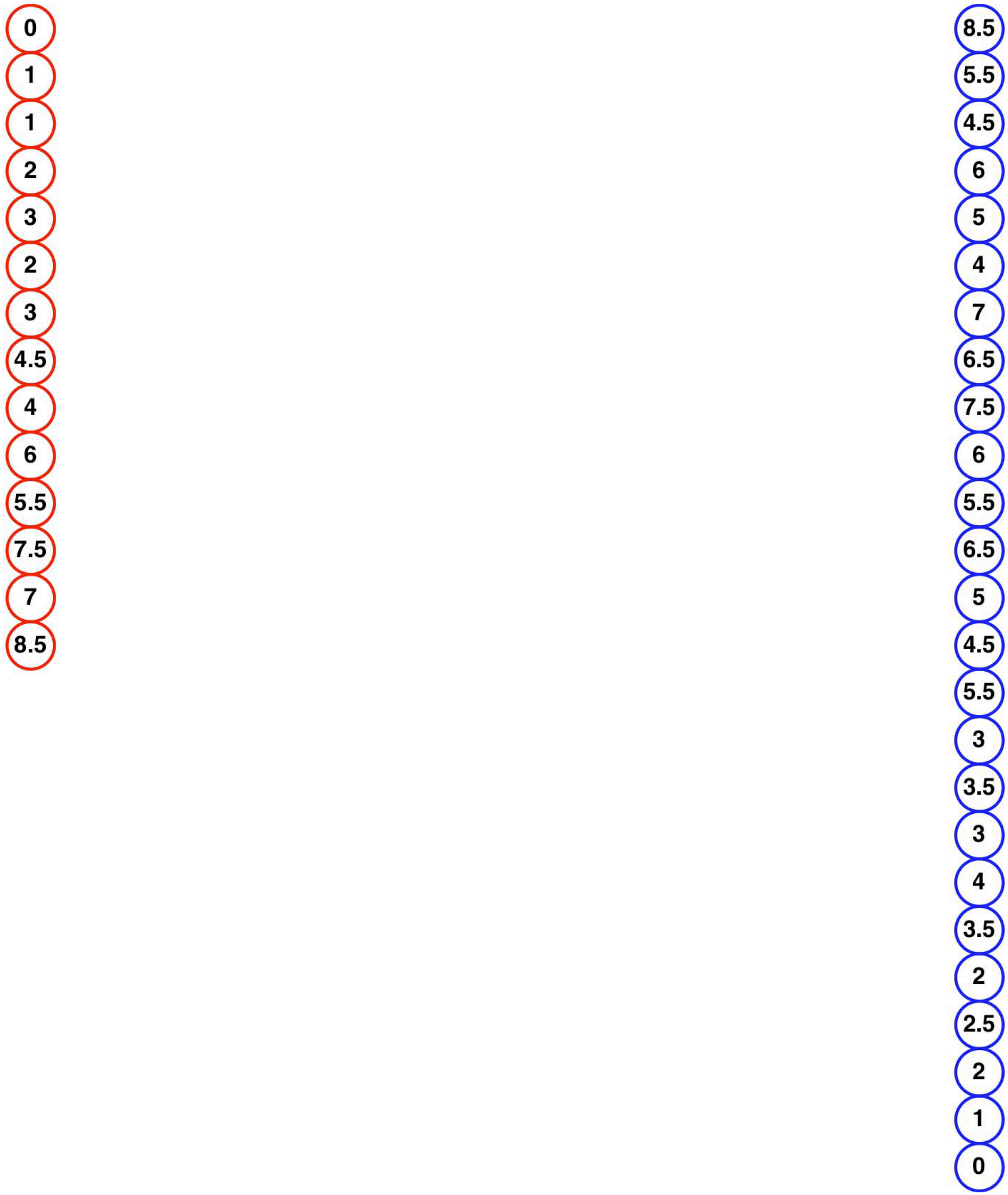


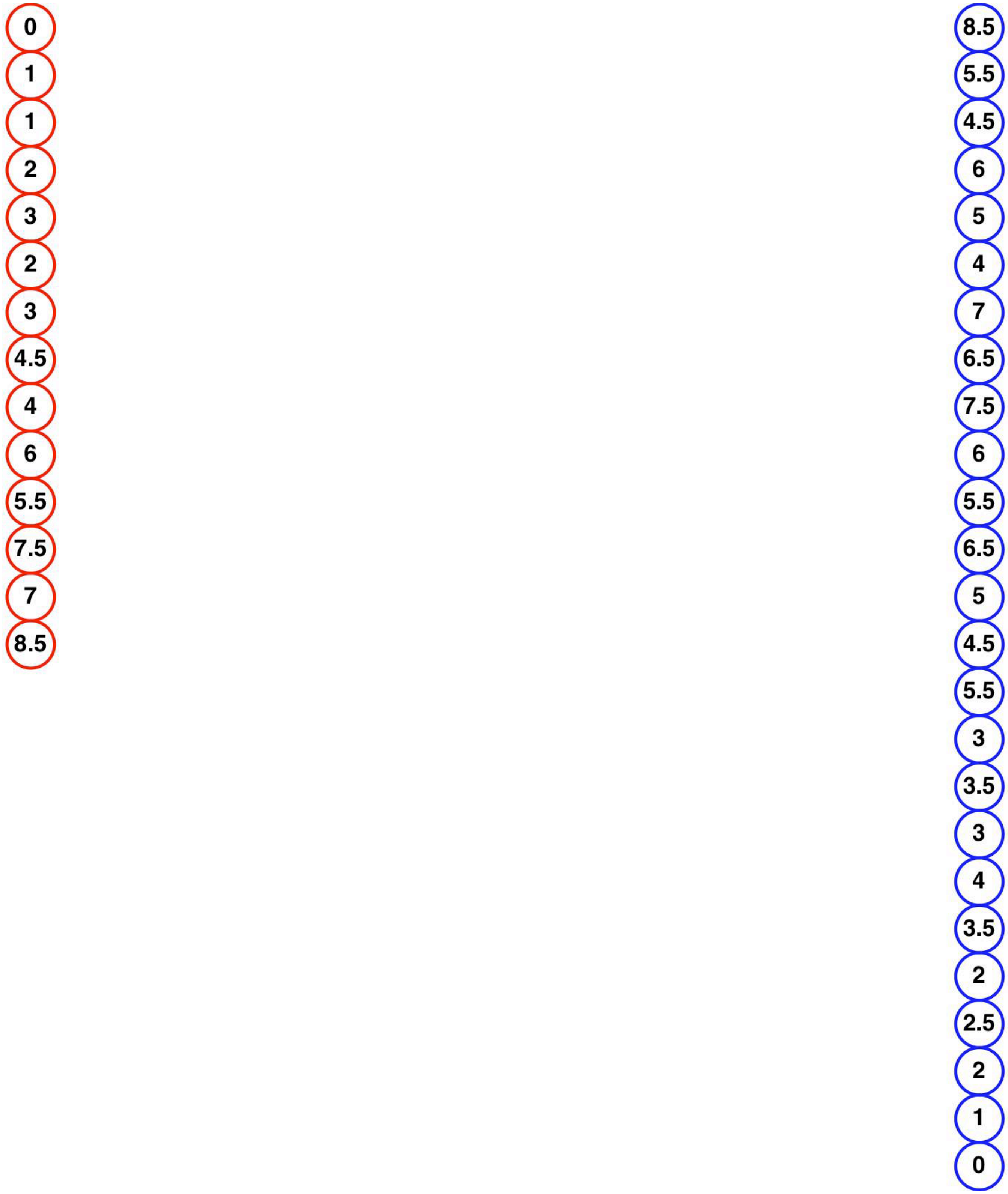
$$g_F(u) + g_B(v) < C^*$$

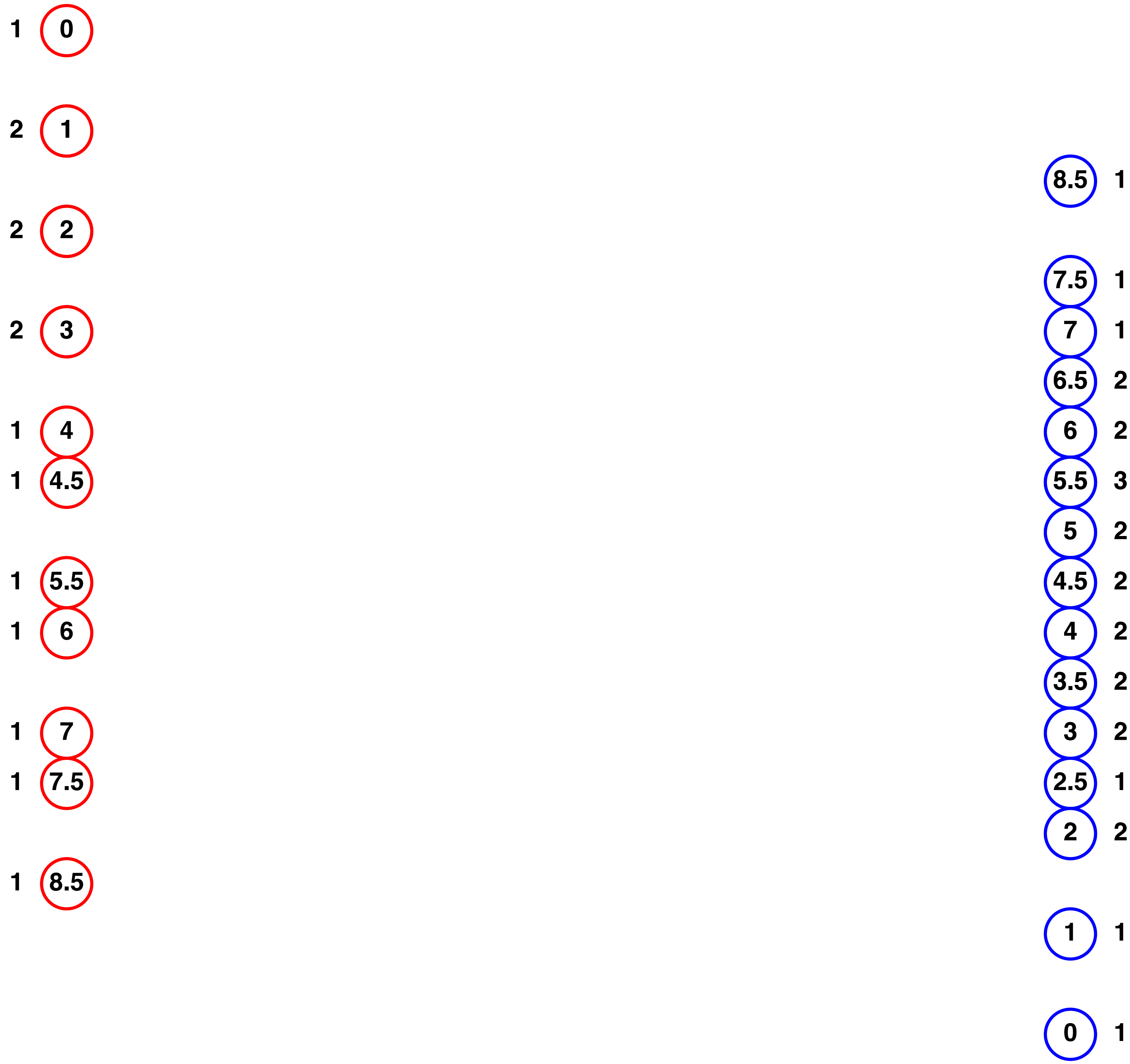
C* = 10.5



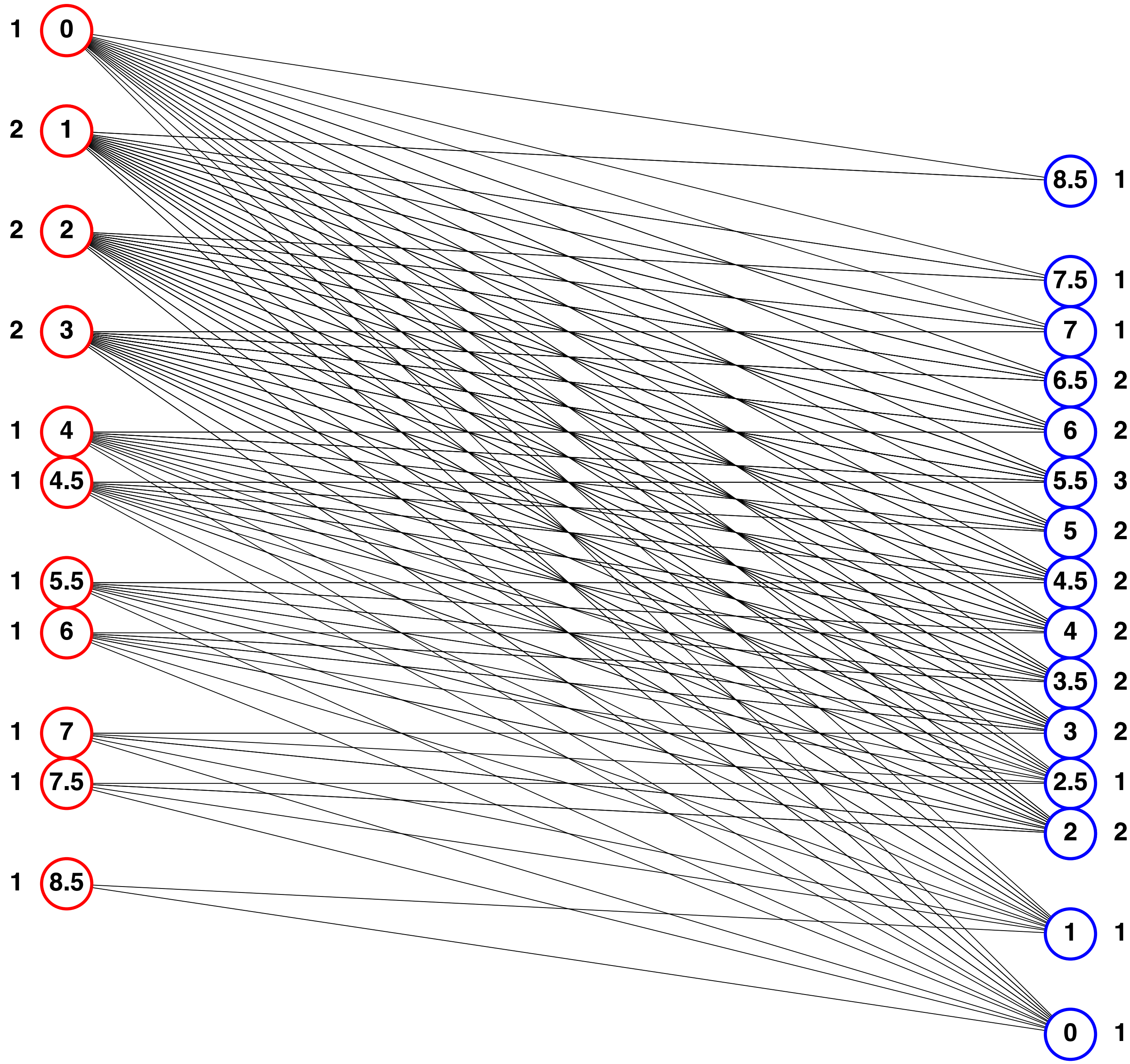
$$g_F(u) + g_B(v) < C^*$$


0
1
1
2
3
2
3
4.5
4
6
5.5
7.5
7
8.5
8.5
5.5
4.5
6
5
4
7
6.5
7.5
6
5.5
6.5
5
4.5
5.5
3
3.5
3
4
3.5
2
2.5
2
1
0

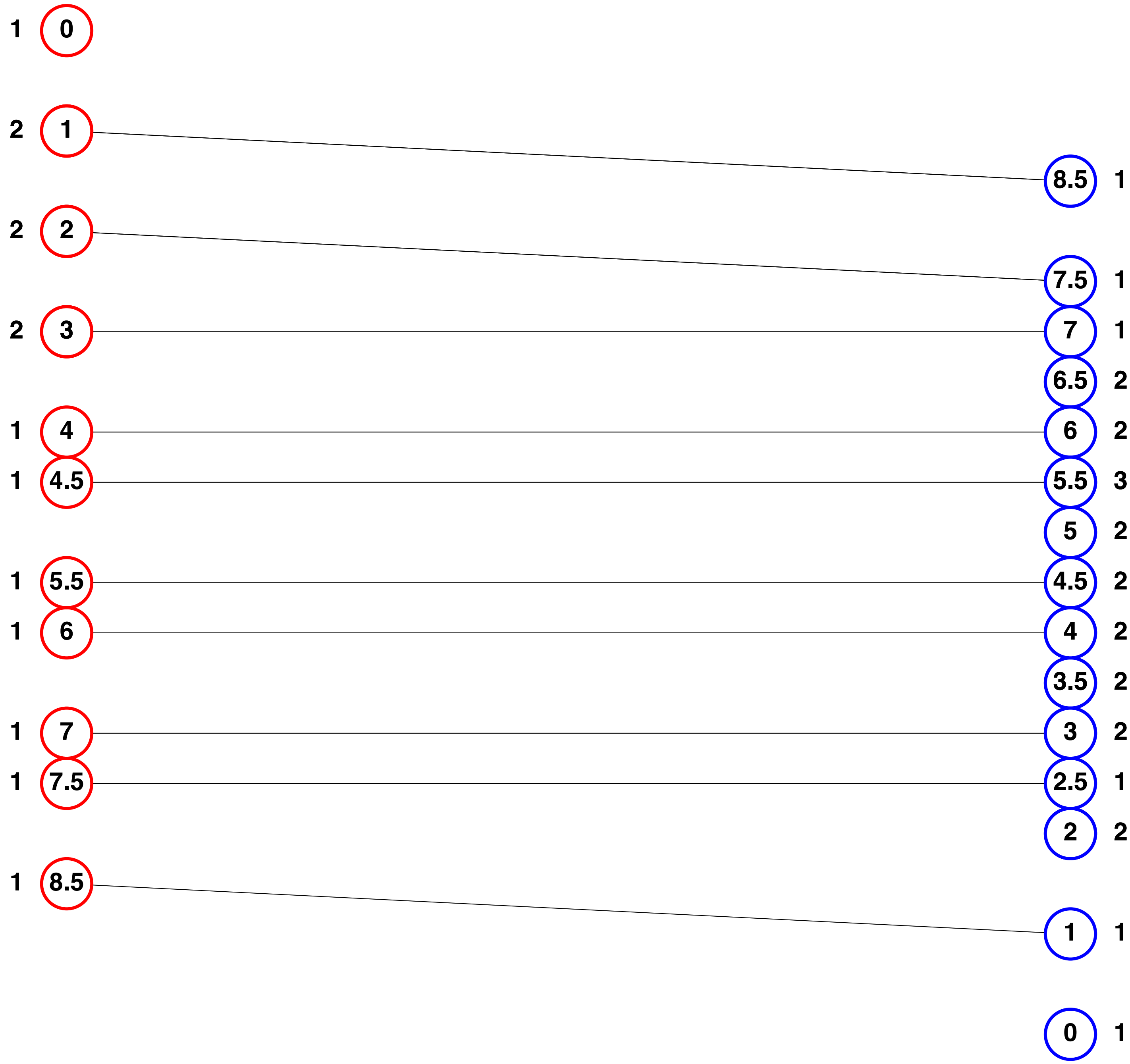

0
1
1
2
3
2
3
4.5
4
6
5.5
7.5
7
8.5
8.5
5.5
4.5
6
5
4
7
6.5
7.5
6
5.5
6.5
5
4.5
5.5
3
3.5
3
4
3.5
2
2.5
2
1
0



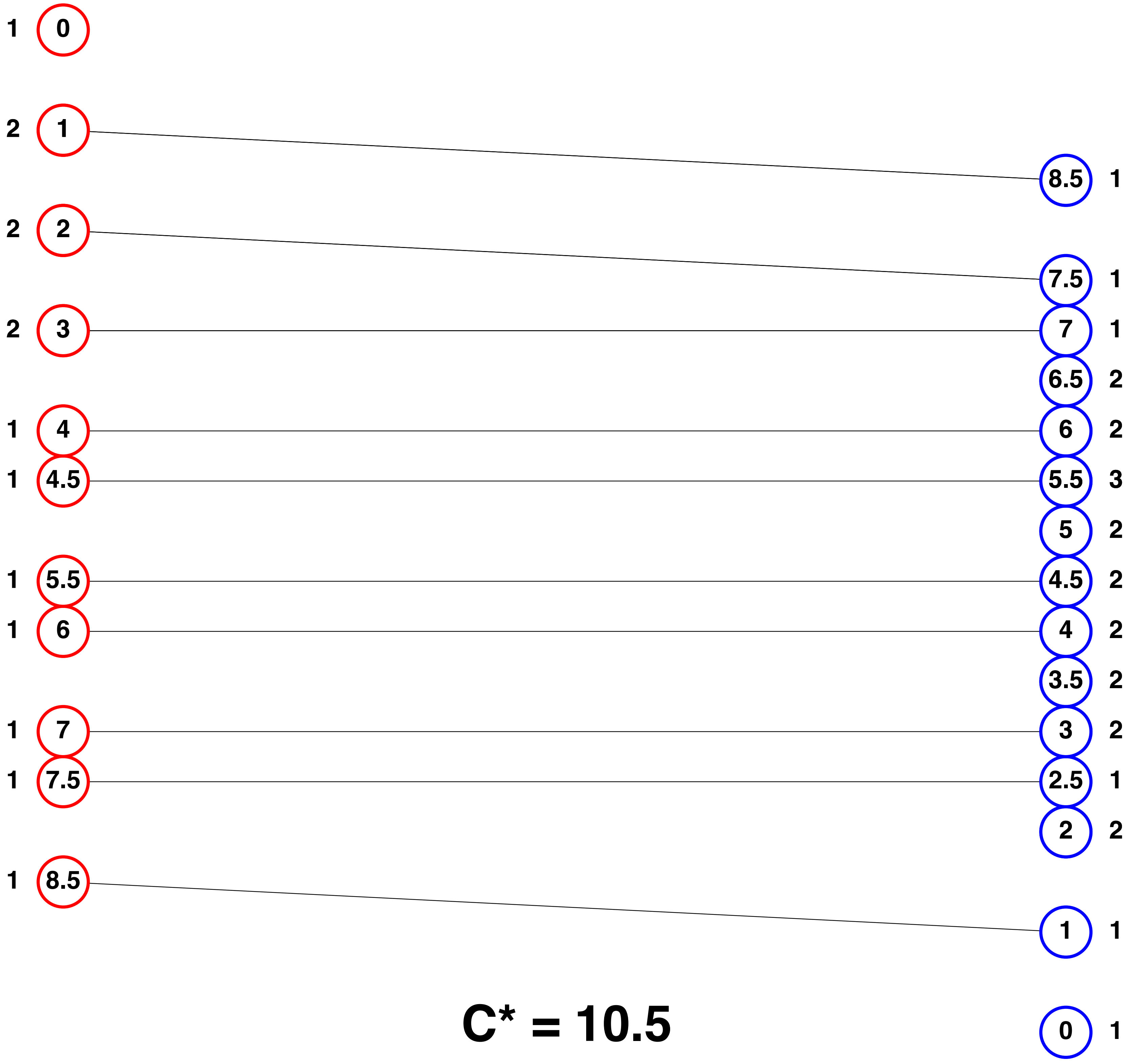
1 0
2 1
2 2
2 3
1 4
1 4.5
1 5.5
1 6
1 7
1 7.5
1 8.5
8.5 1
7.5 1
7 1
6.5 2
6 2
5.5 3
5 2
4.5 2
4 2
3.5 2
3 2
2.5 1
2 2
1 1
0 1



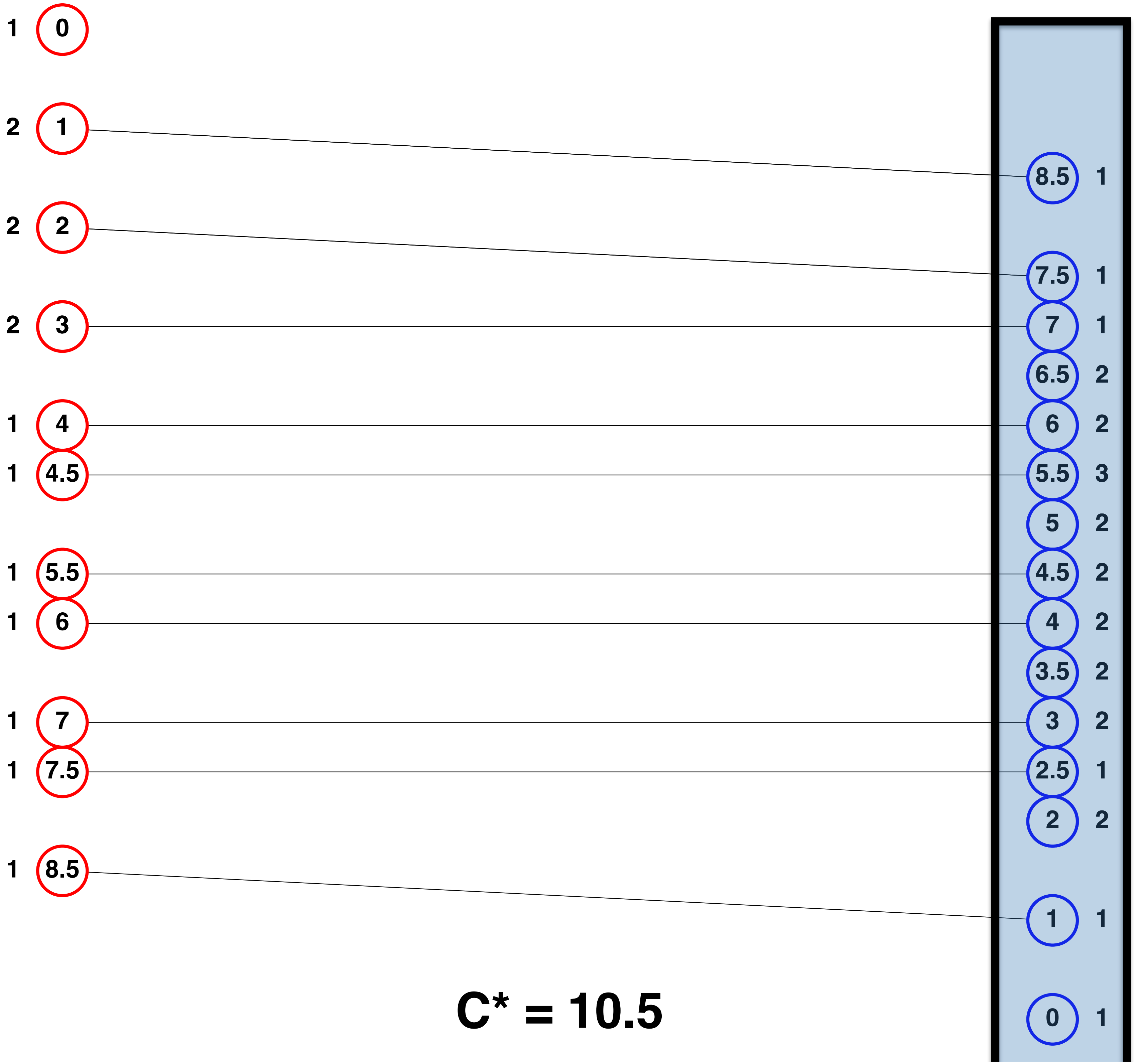
1 0
2 1
2 2
2 3
1 4
1 4.5
1 5.5
1 6
1 7
1 7.5
1 8.5
8.5 1
7.5 1
7 1
6.5 2
6 2
5.5 3
5 2
4.5 2
4 2
3.5 2
3 2
2.5 1
2 2
1 1
0 1


1 0
2 1
8.5 1
2 2
7.5 1
2 3
7 1
6.5 2
1 4
6 2
1 4.5
5.5 3
5 2
1 5.5
4.5 2
1 6
4 2
3.5 2
1 7
3 2
1 7.5
2.5 1
2 2
1 8.5
1 1
0 1

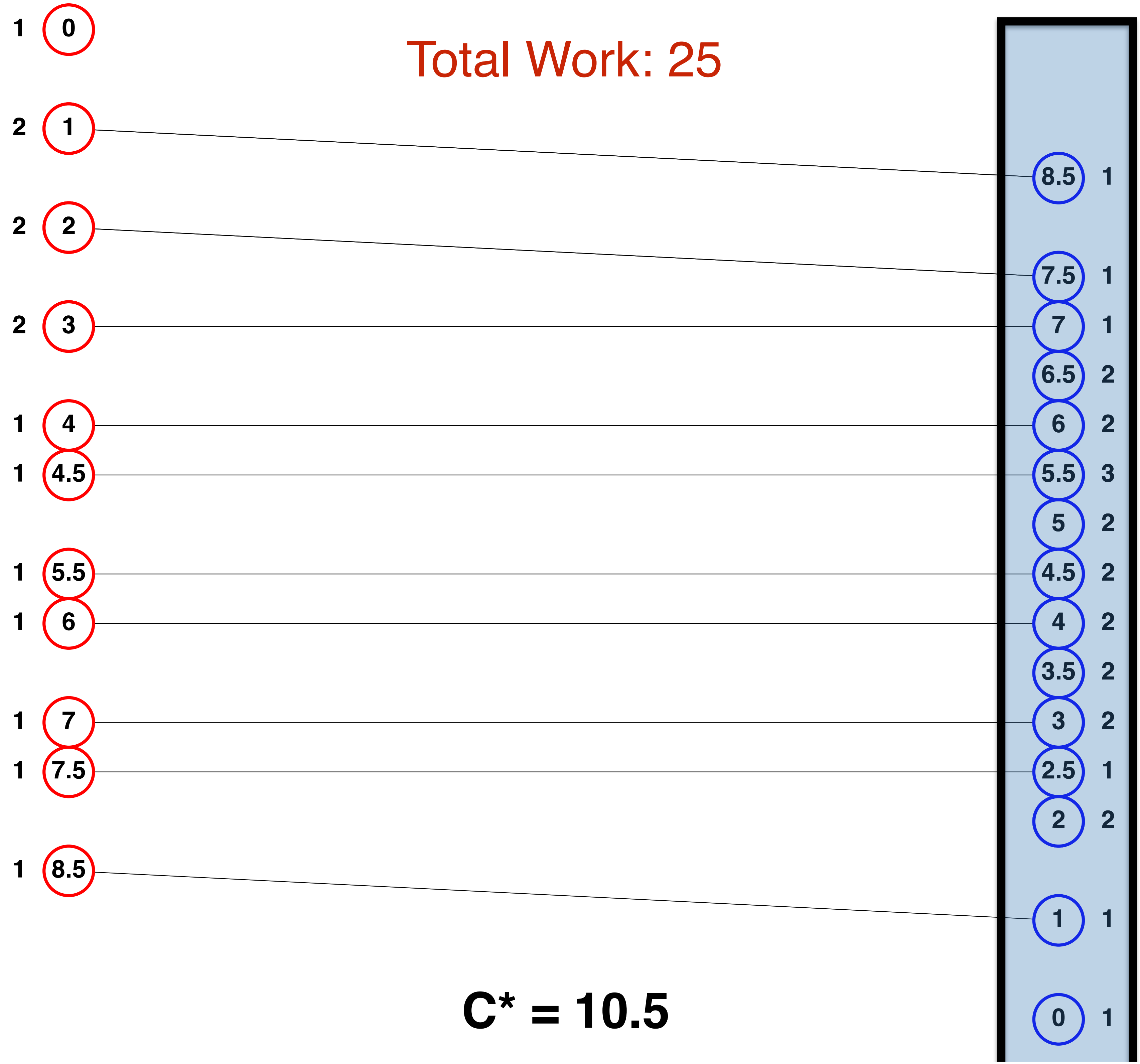
$$C^* = 10.5$$


1
0
2
1
2
2
2
3
1
4
1
4.5
1
5.5
1
6
1
7
1
7.5
1
8.5
8.5
1
7.5
1
7
1
6.5
2
6
2
5.5
3
5
2
4.5
2
4
2
3.5
2
3
2
2.5
1
2
2
1
1
0
1

$C^* = 10.5$

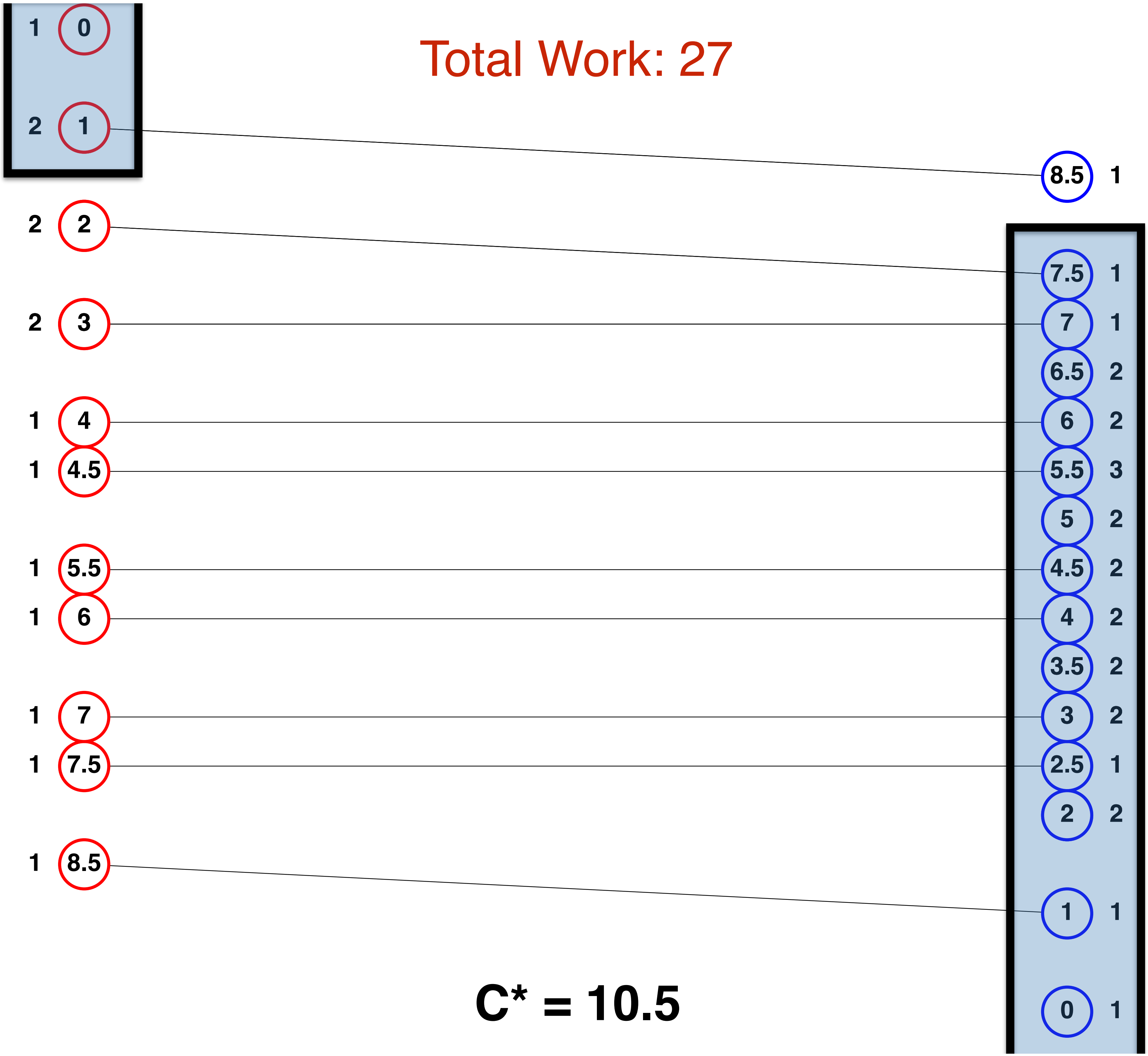

1
0
2
1
2
2
2
3
1
4
1
4.5
1
5.5
1
6
1
7
1
7.5
1
8.5
8.5
1
7.5
1
7
1
6.5
2
6
2
5.5
3
5
2
4.5
2
4
2
3.5
2
3
2
2.5
1
2
2
1
1
0
1

$C^* = 10.5$



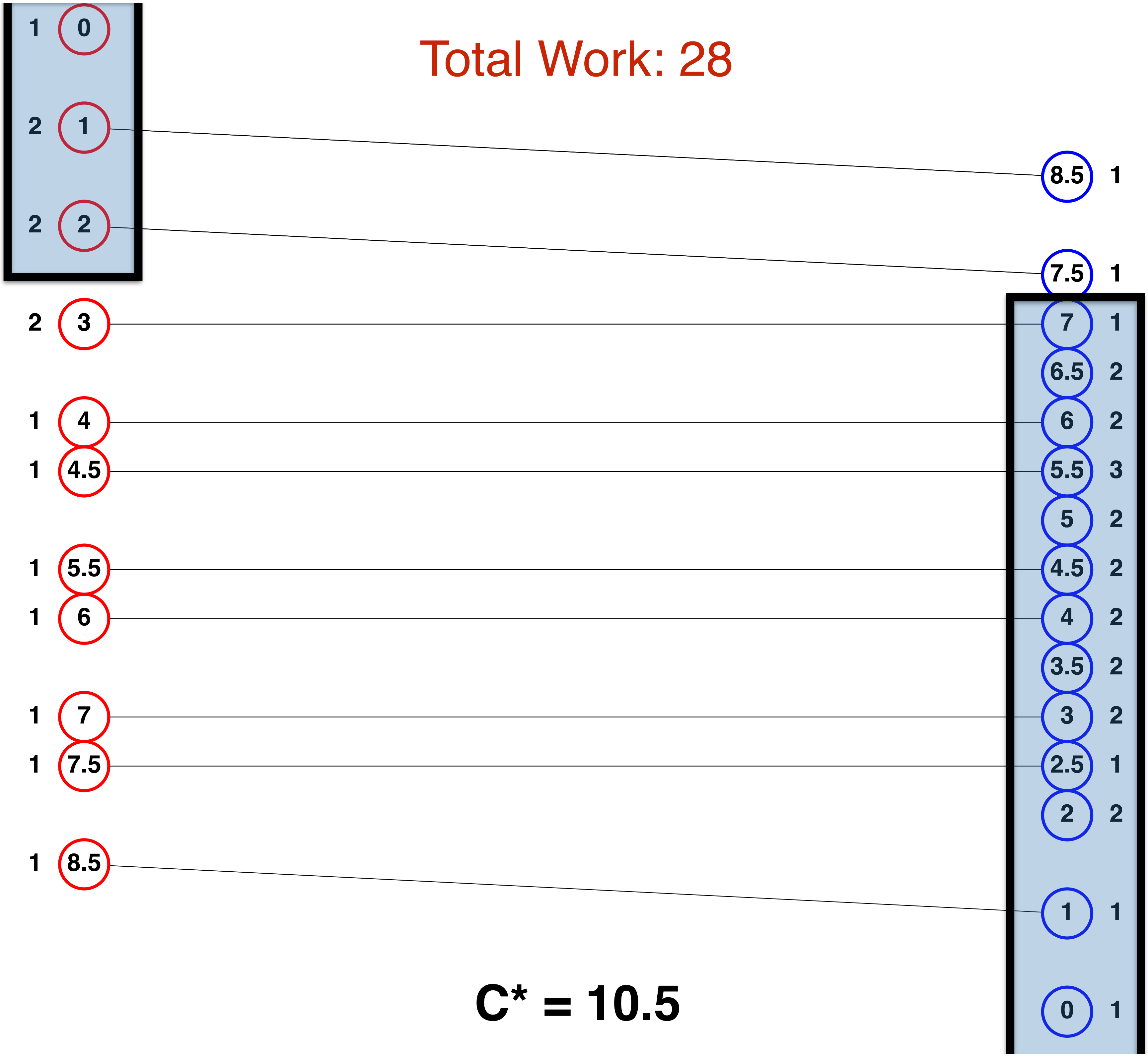
Total Work: 25
1 0
2 1
2 2
2 3
1 4
1 4.5
1 5.5
1 6
1 7
1 7.5
1 8.5
8.5 1
7.5 1
7 1
6.5 2
6 2
5.5 3
5 2
4.5 2
4 2
3.5 2
3 2
2.5 1
2 2
1 1
0 1
C* = 10.5

Total Work: 27



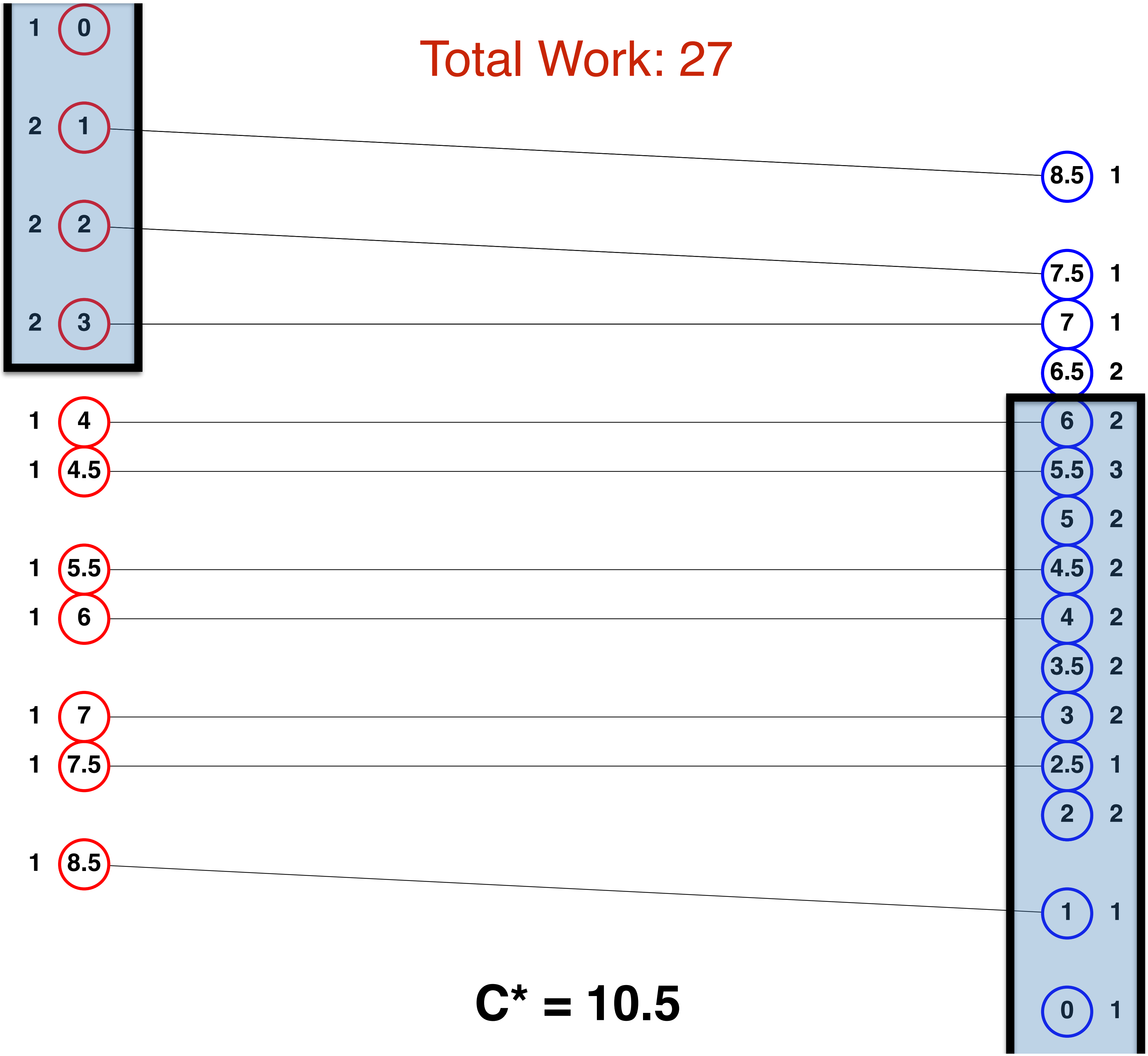
$C^* = 10.5$

Total Work: 28



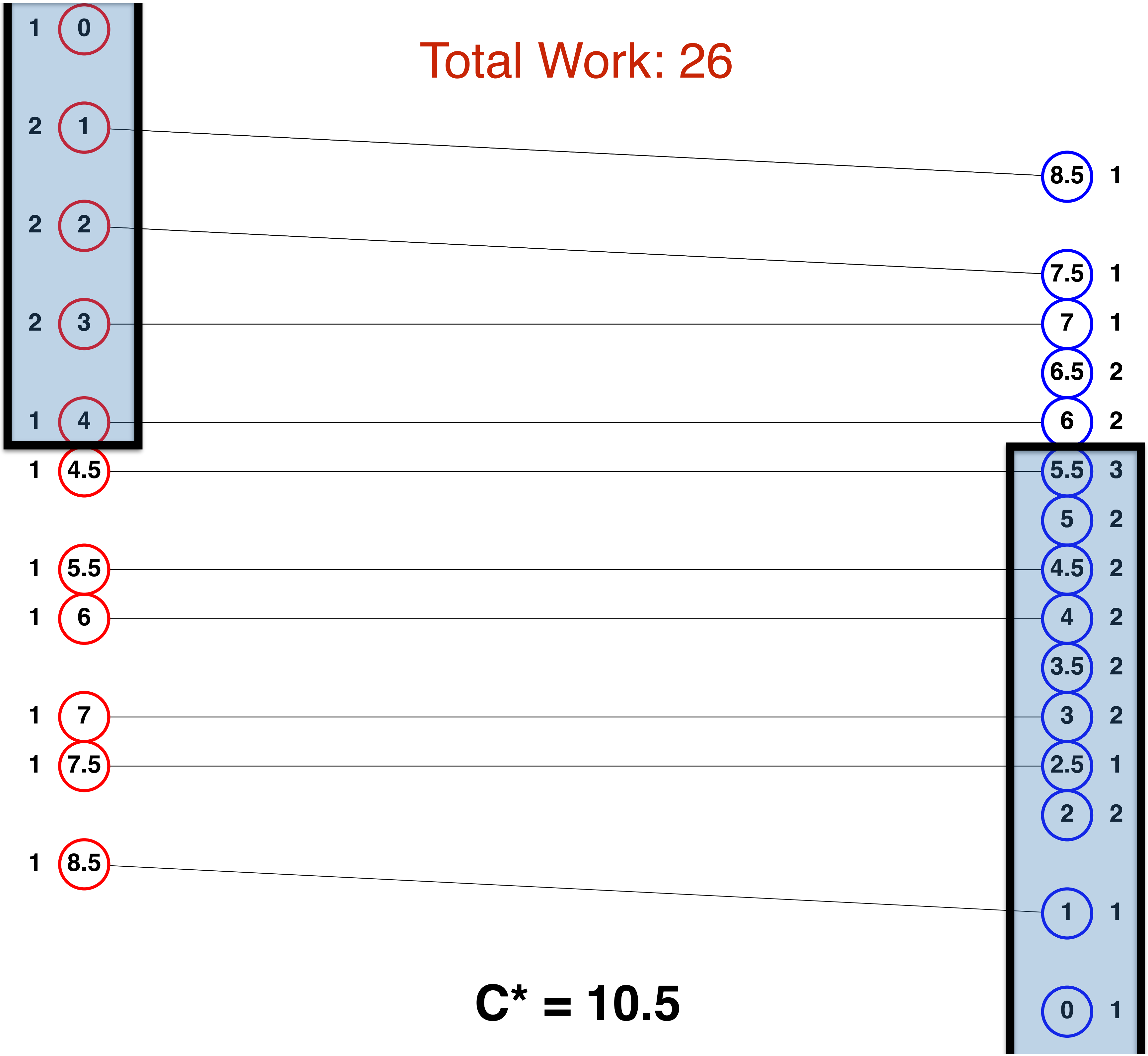
$C^* = 10.5$

Total Work: 27



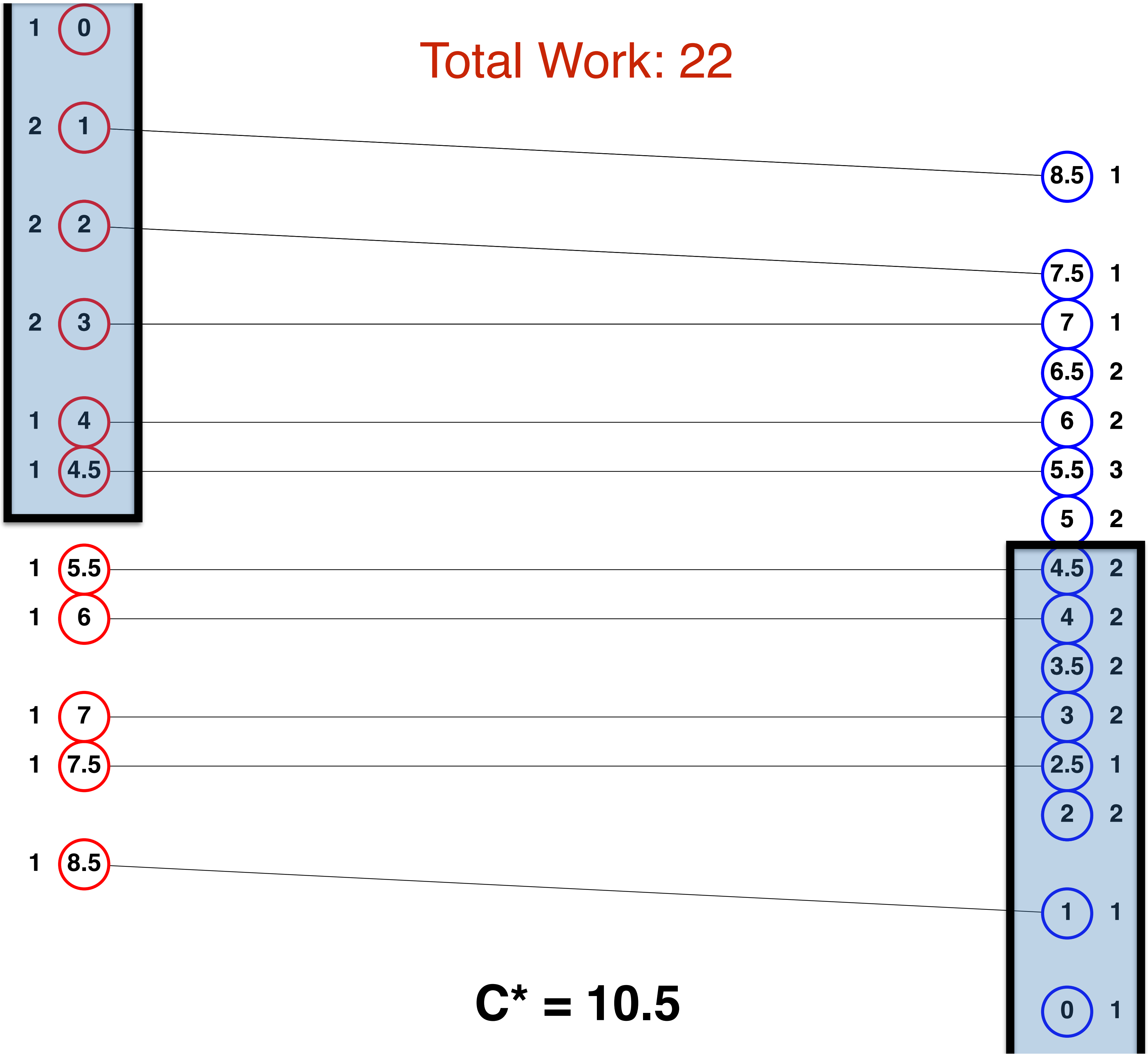
$C^* = 10.5$

Total Work: 26



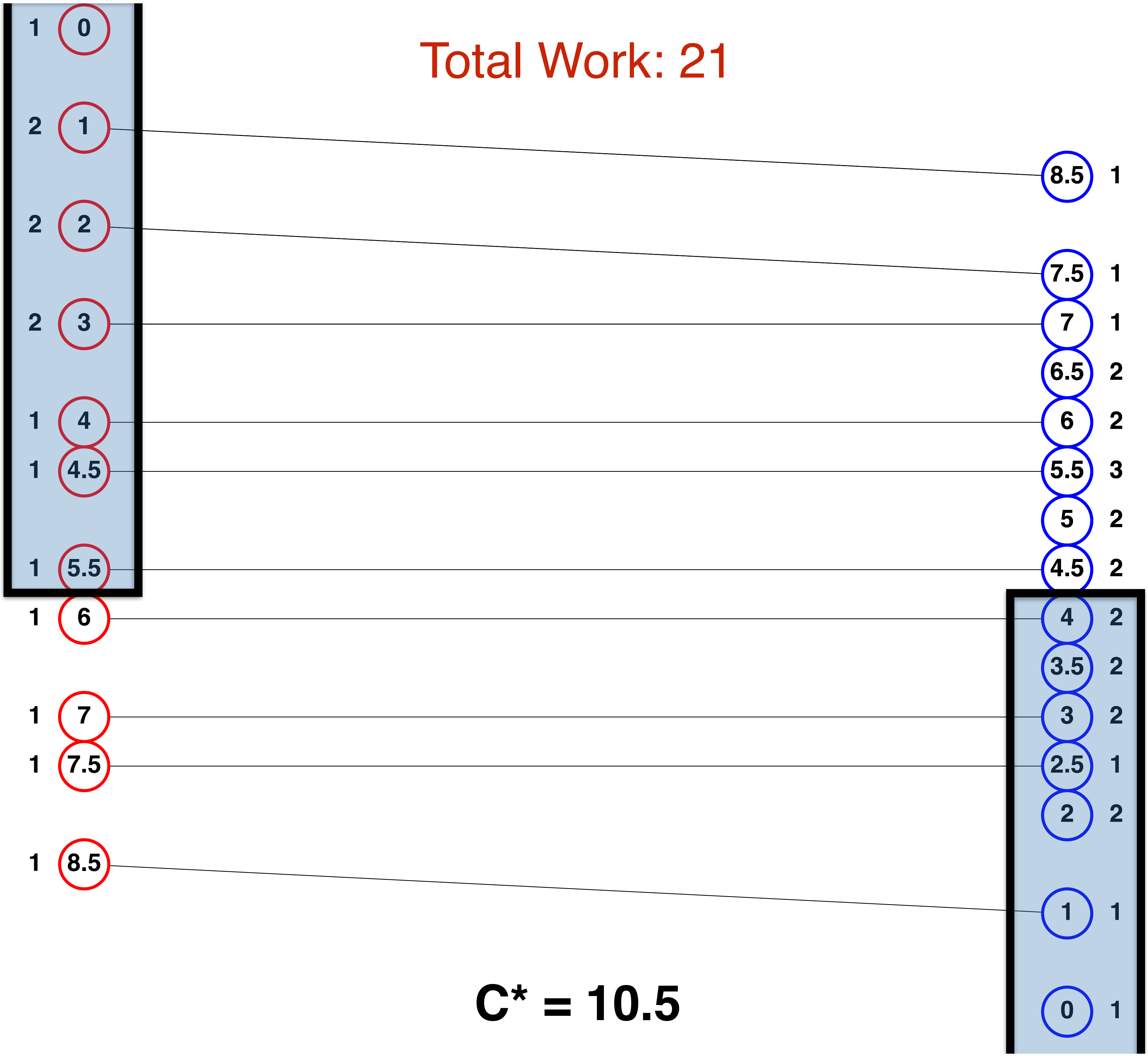
$C^* = 10.5$

Total Work: 22


1 0
2 1
2 2
2 3
1 4
1 4.5
1 5.5
1 6
1 7
1 7.5
1 8.5
8.5 1
7.5 1
7 1
6.5 2
6 2
5.5 3
5 2
4.5 2
4 2
3.5 2
3 2
2.5 1
2 2
1 1
0 1

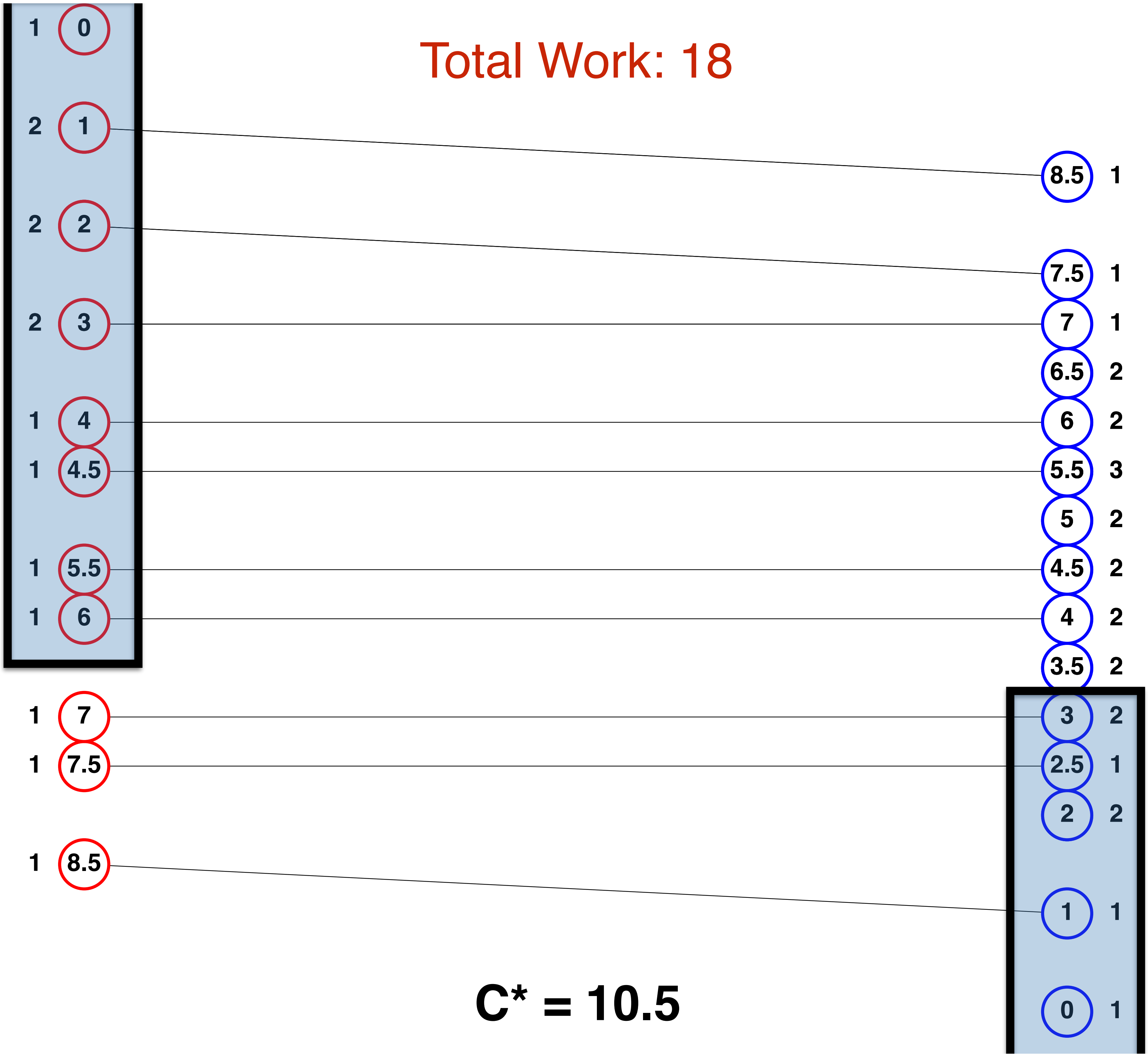
$C^* = 10.5$

Total Work: 21

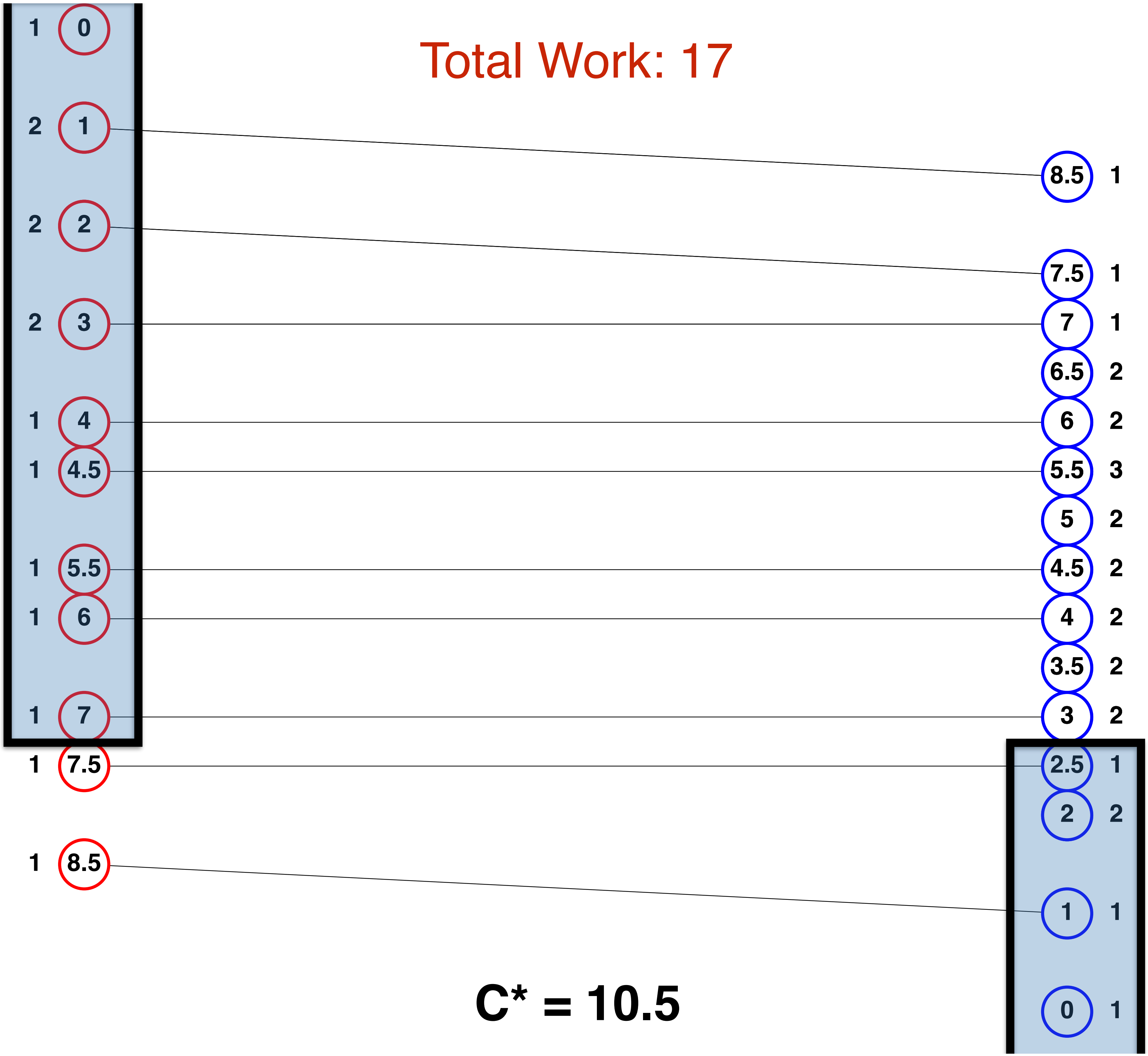


$$C^* = 10.5$$

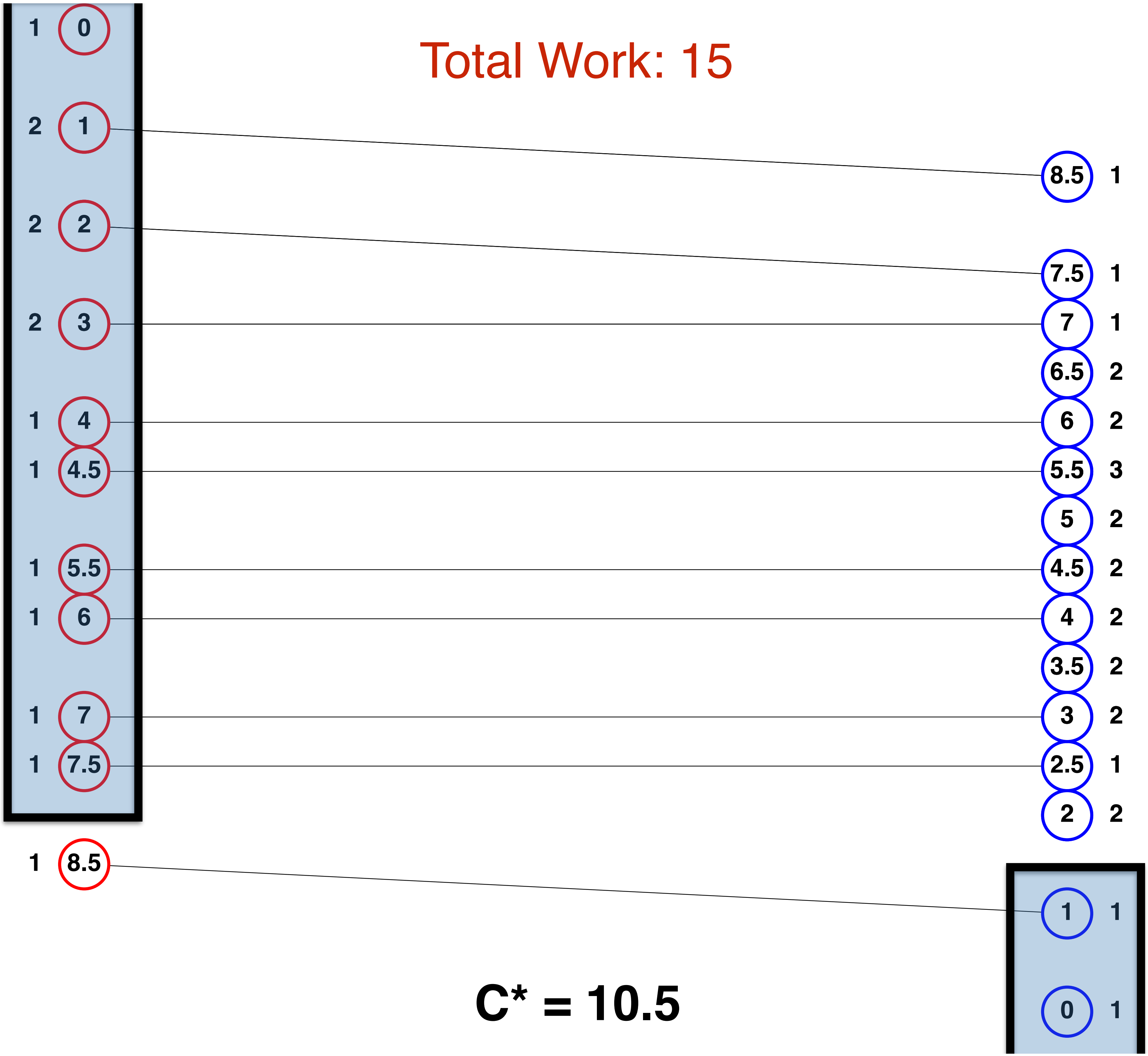
Total Work: 18



$C^* = 10.5$

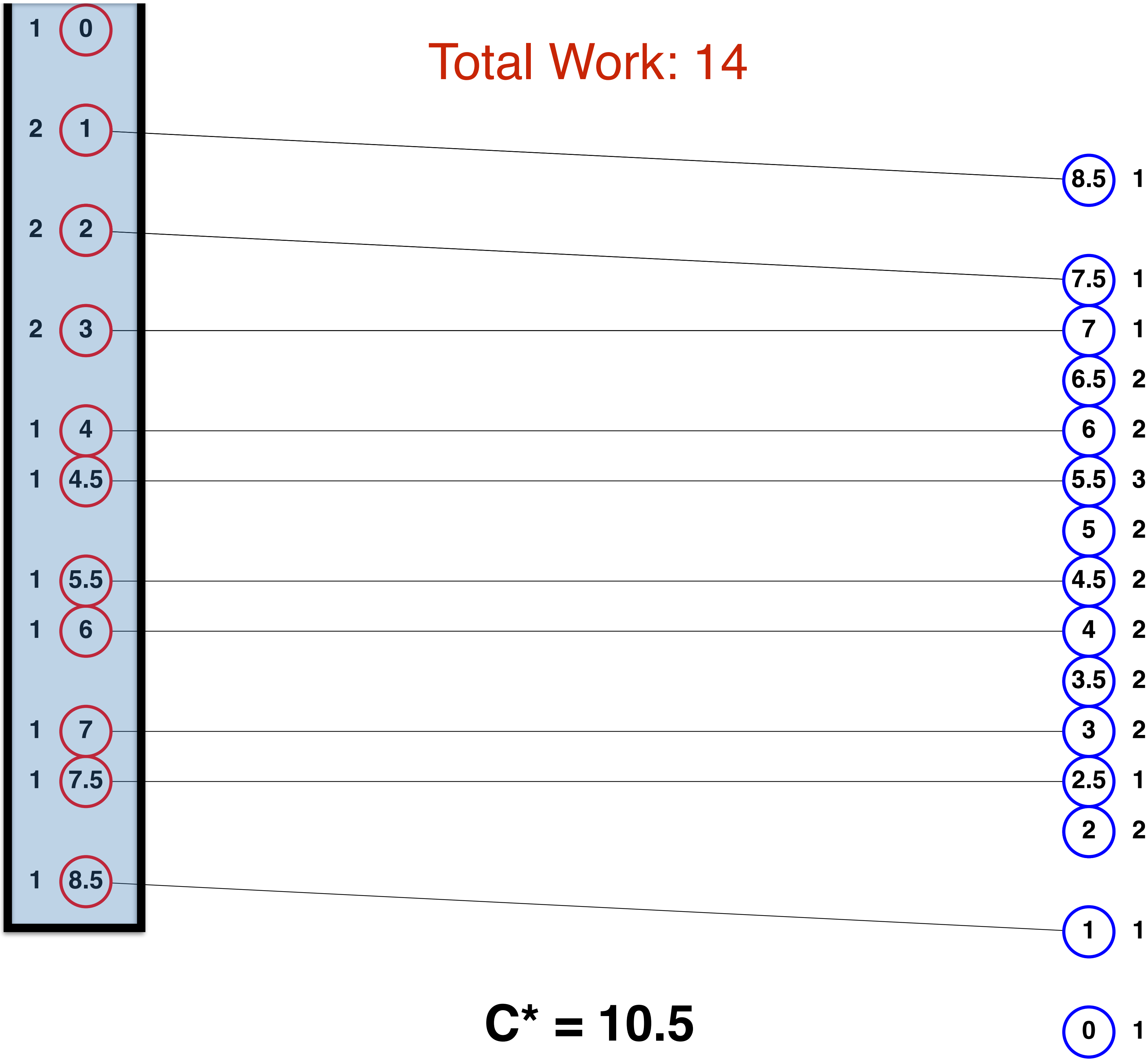

Total Work: 17
1 0
2 1
2 2
2 3
1 4
1 4.5
1 5.5
1 6
1 7
1 7.5
1 8.5
8.5 1
7.5 1
7 1
6.5 2
6 2
5.5 3
5 2
4.5 2
4 2
3.5 2
3 2
2.5 1
2 2
1 1
0 1
C* = 10.5

Total Work: 15

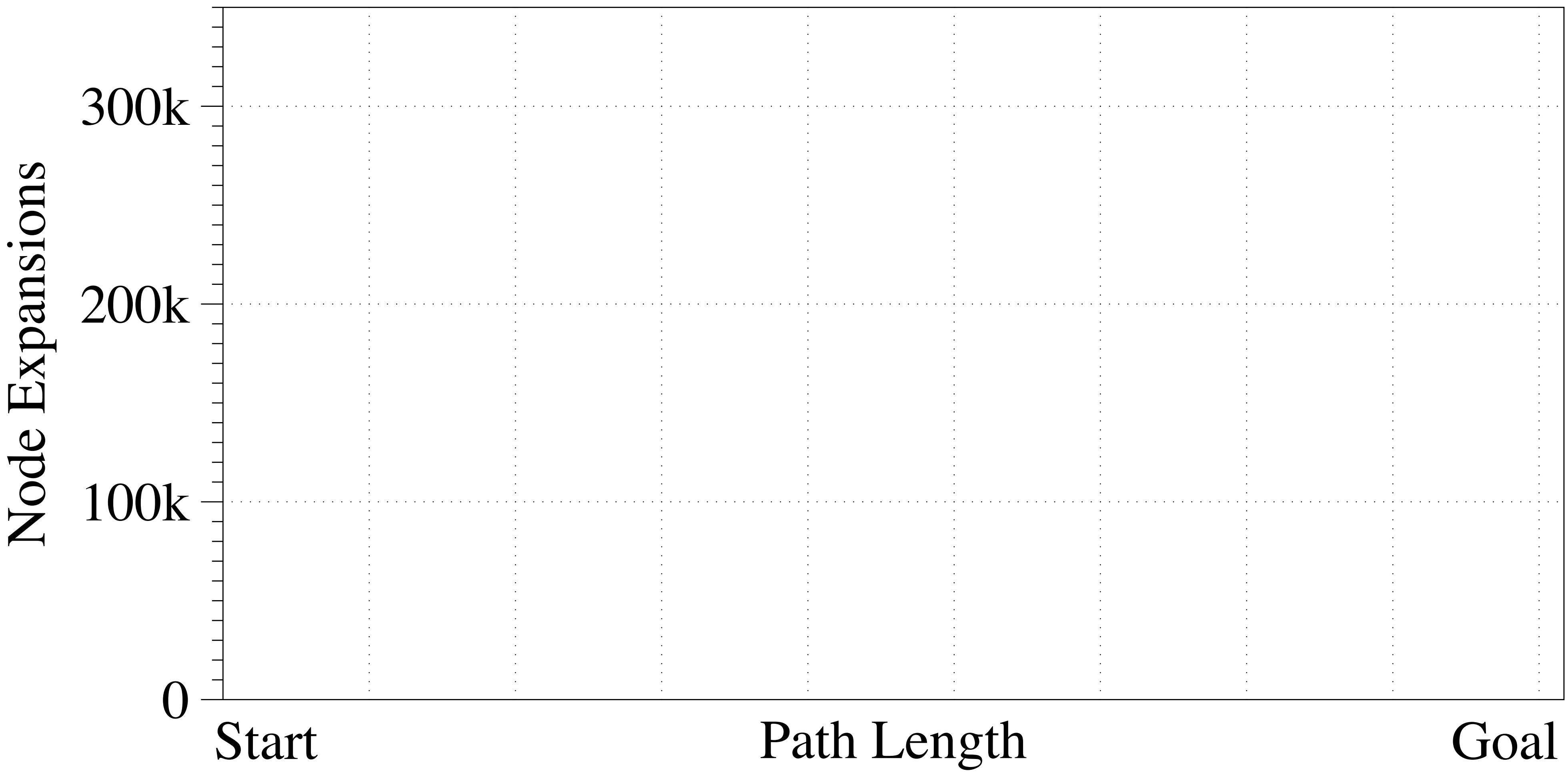


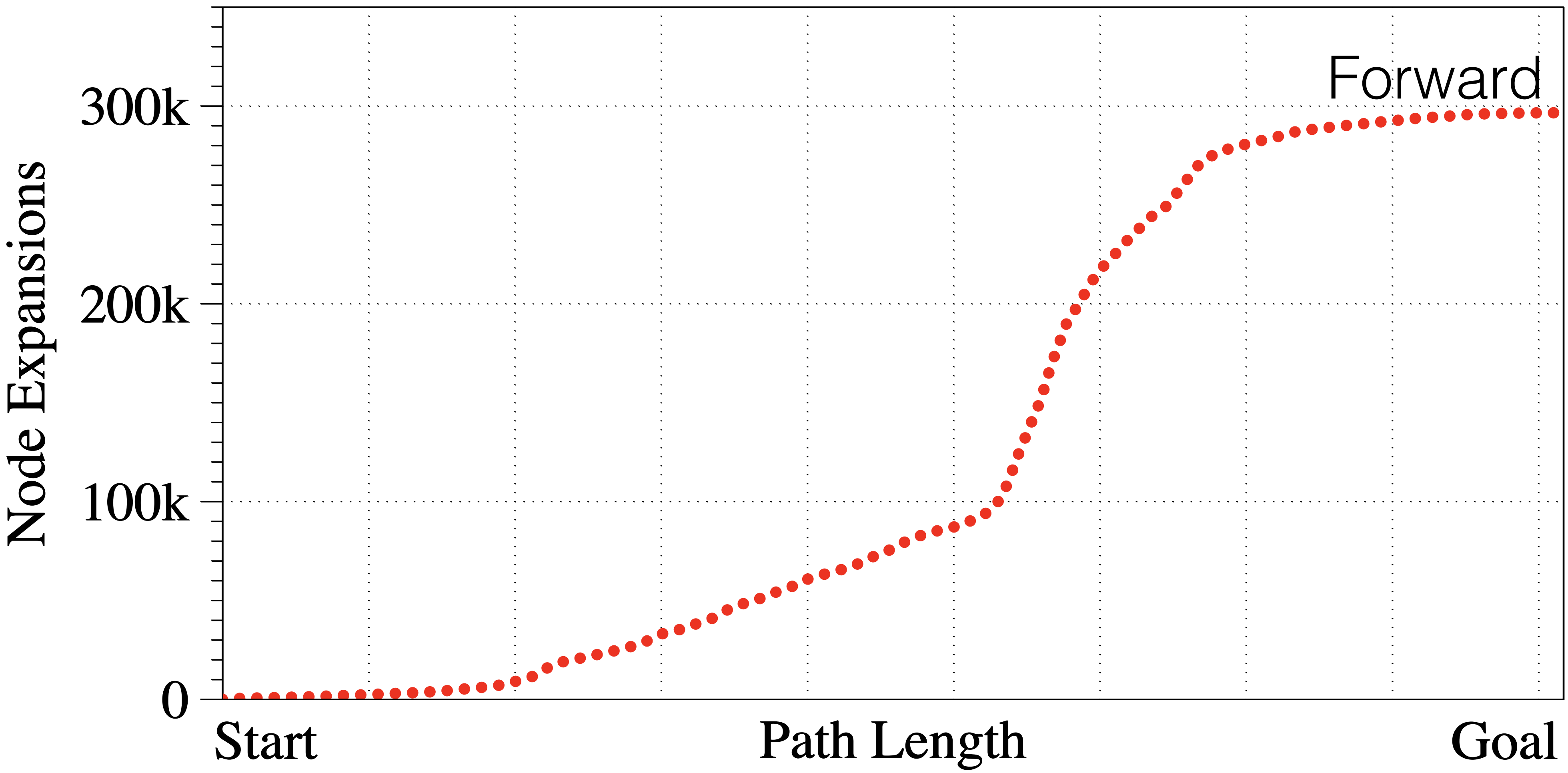
C* = 10.5

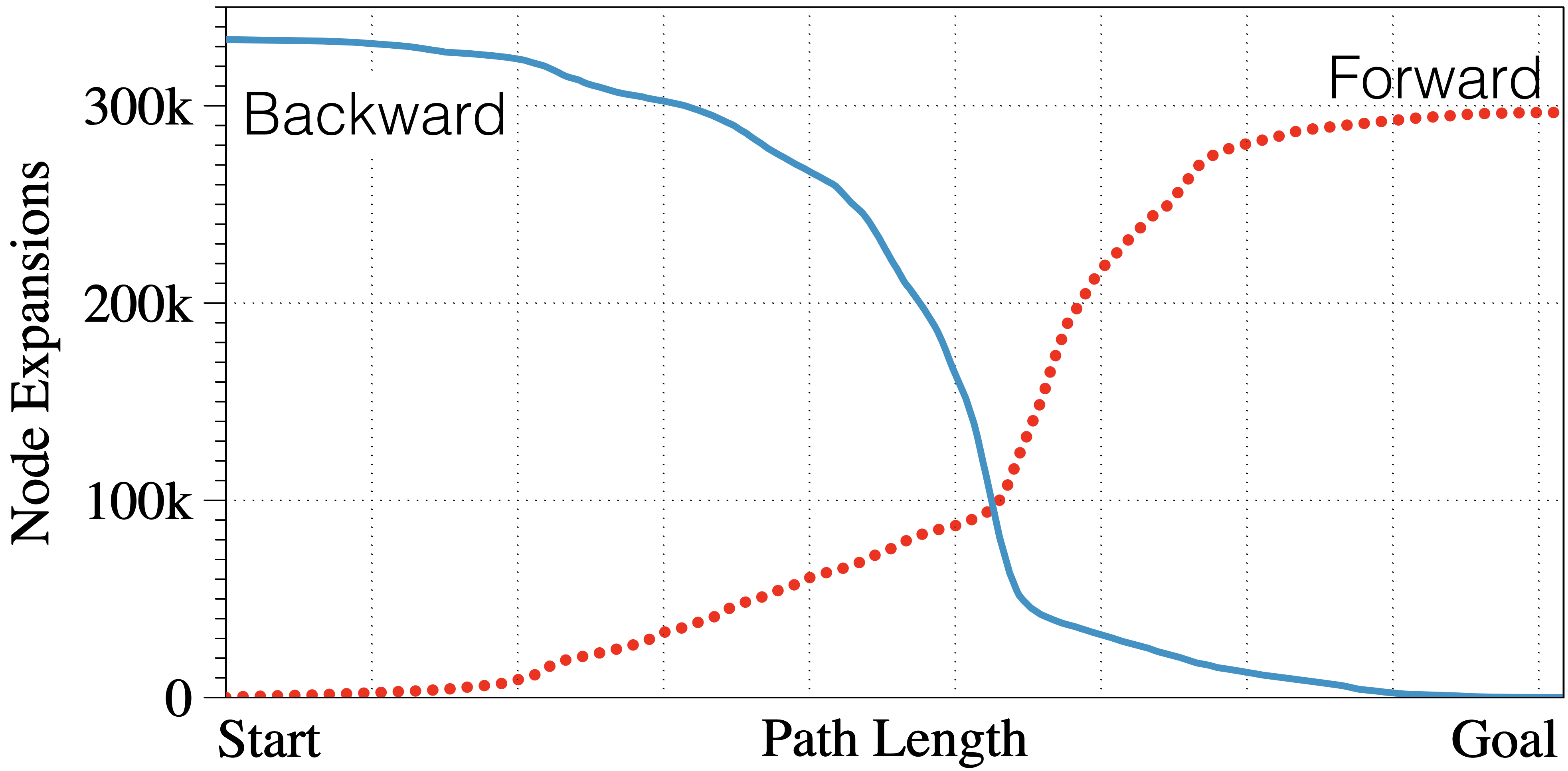
Total Work: 14



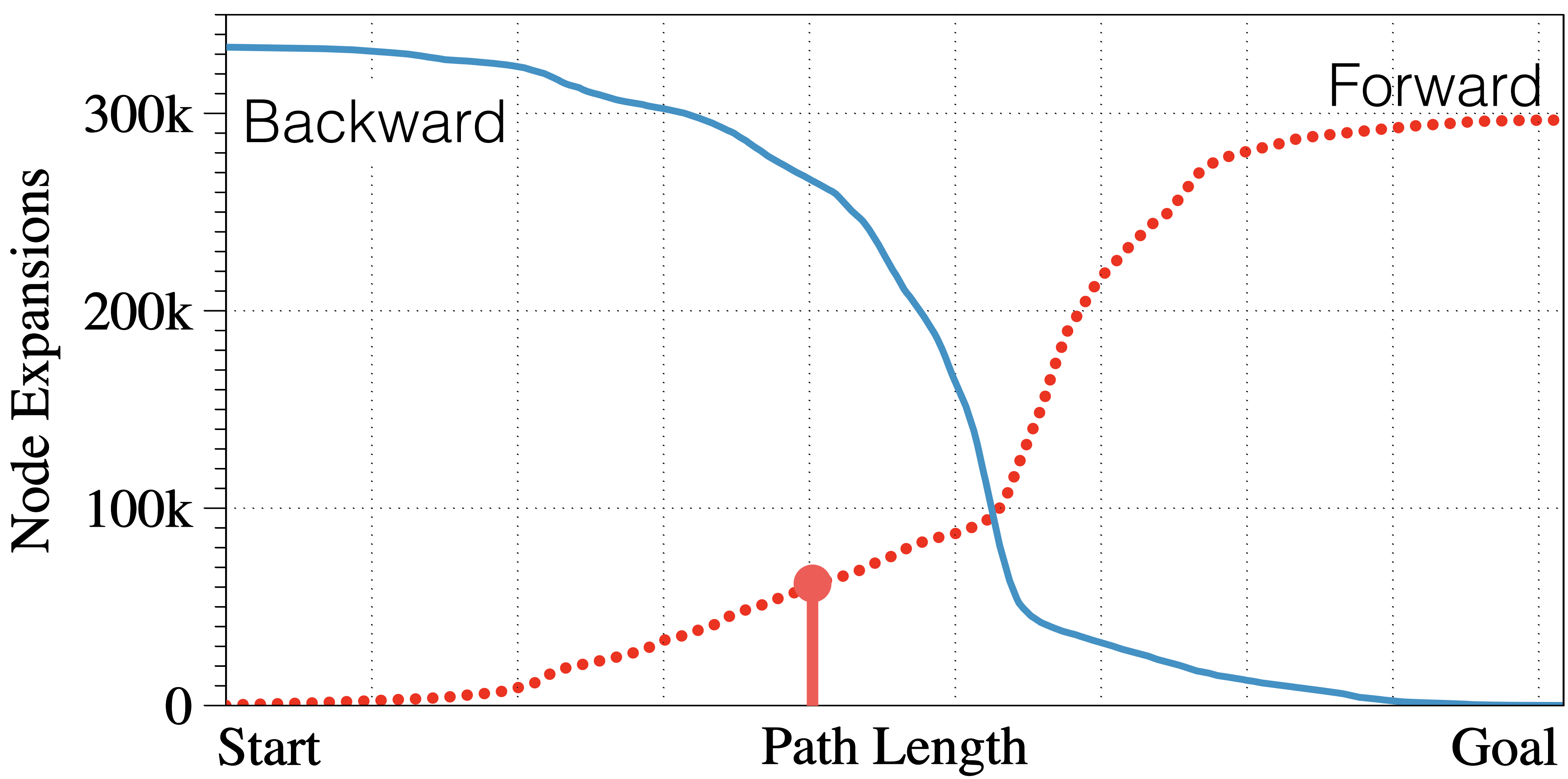
$$C^* = 10.5$$


Node Expansions
300k
200k
100k
0
Start
Path Length
Goal

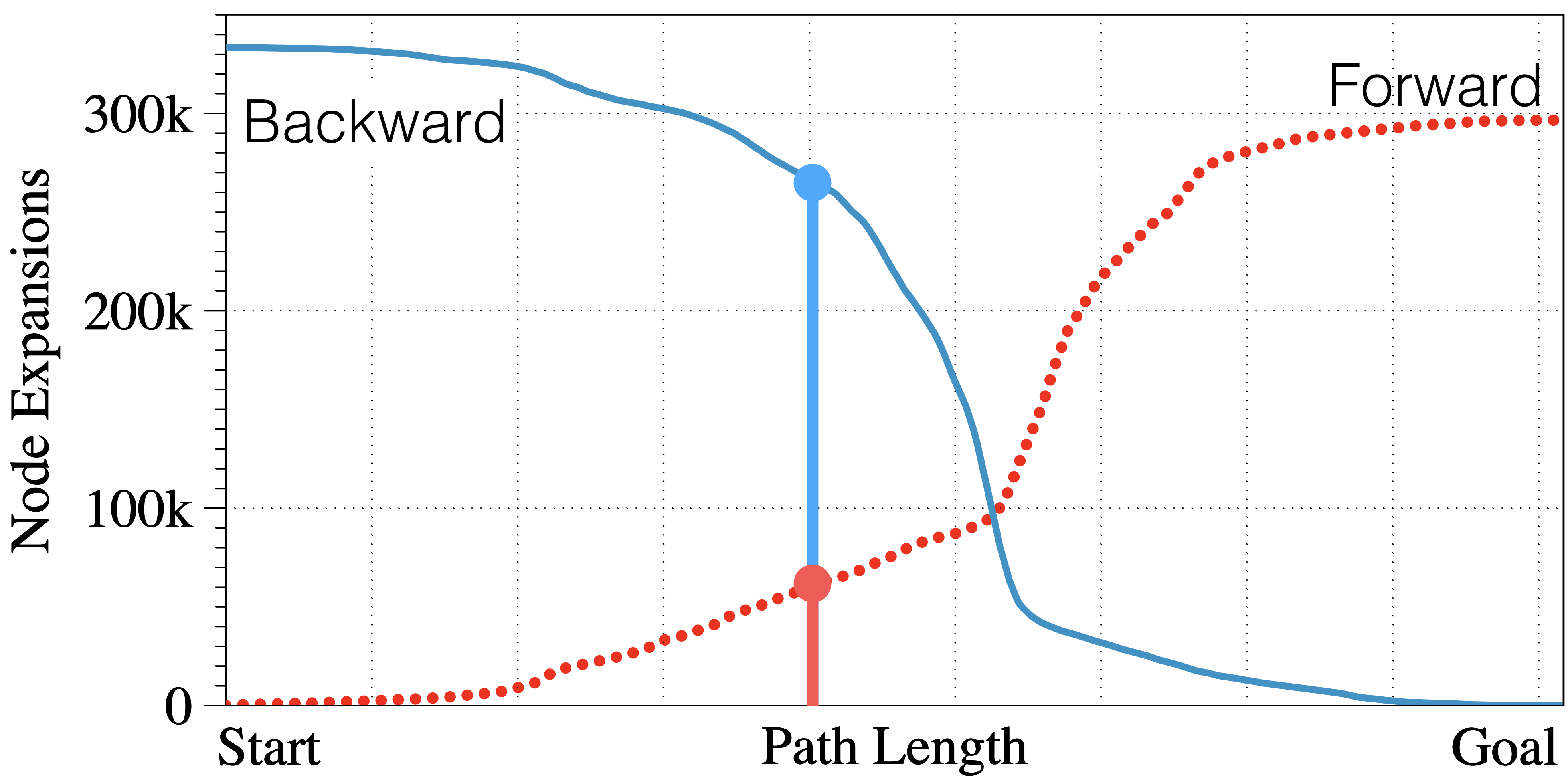

Forward
Node Expansions
300k
200k
100k
0
Start
Path Length
Goal



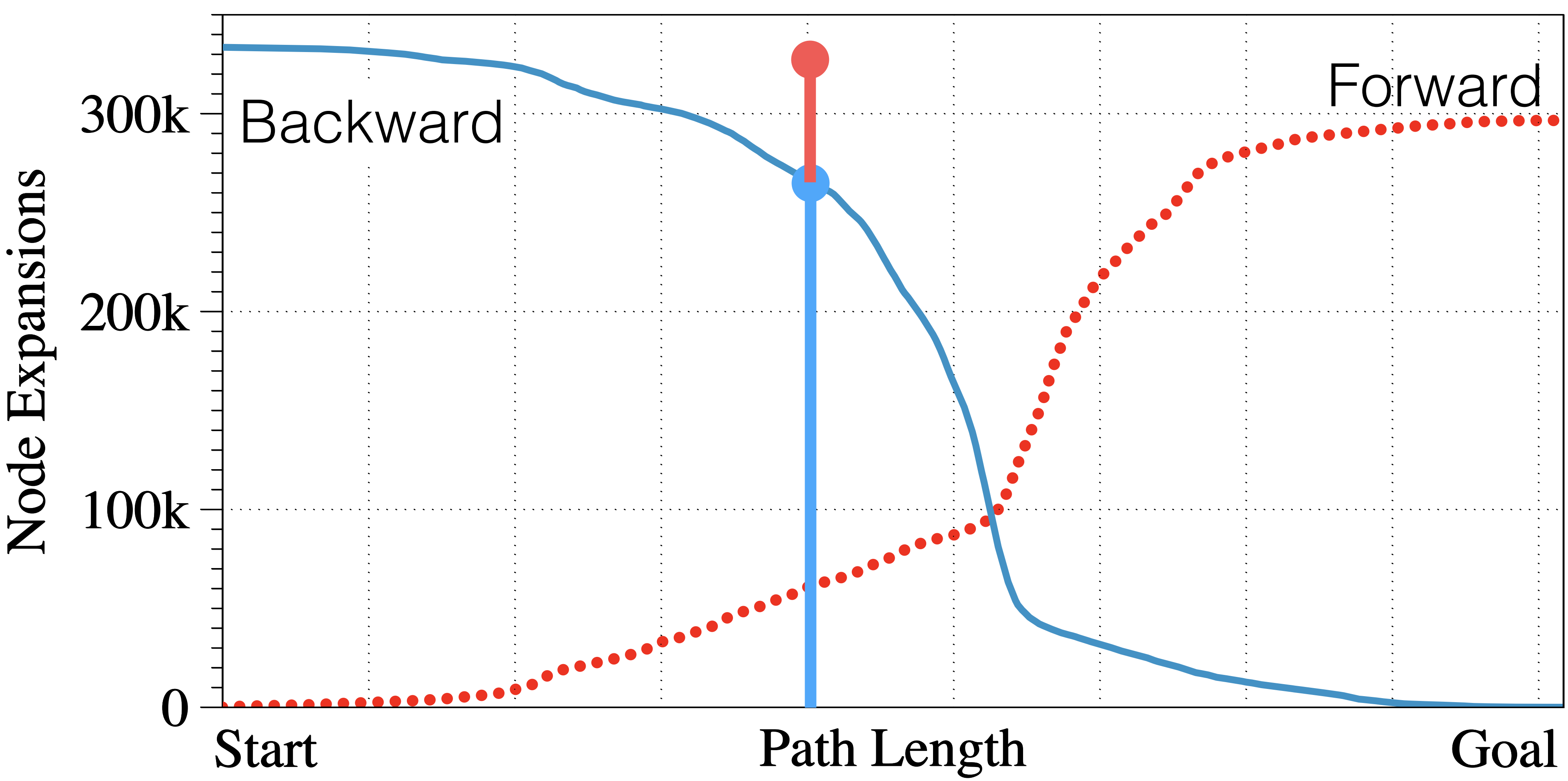
Node Expansions
300k
200k
100k
0
Backward
Forward
Start
Path Length
Goal



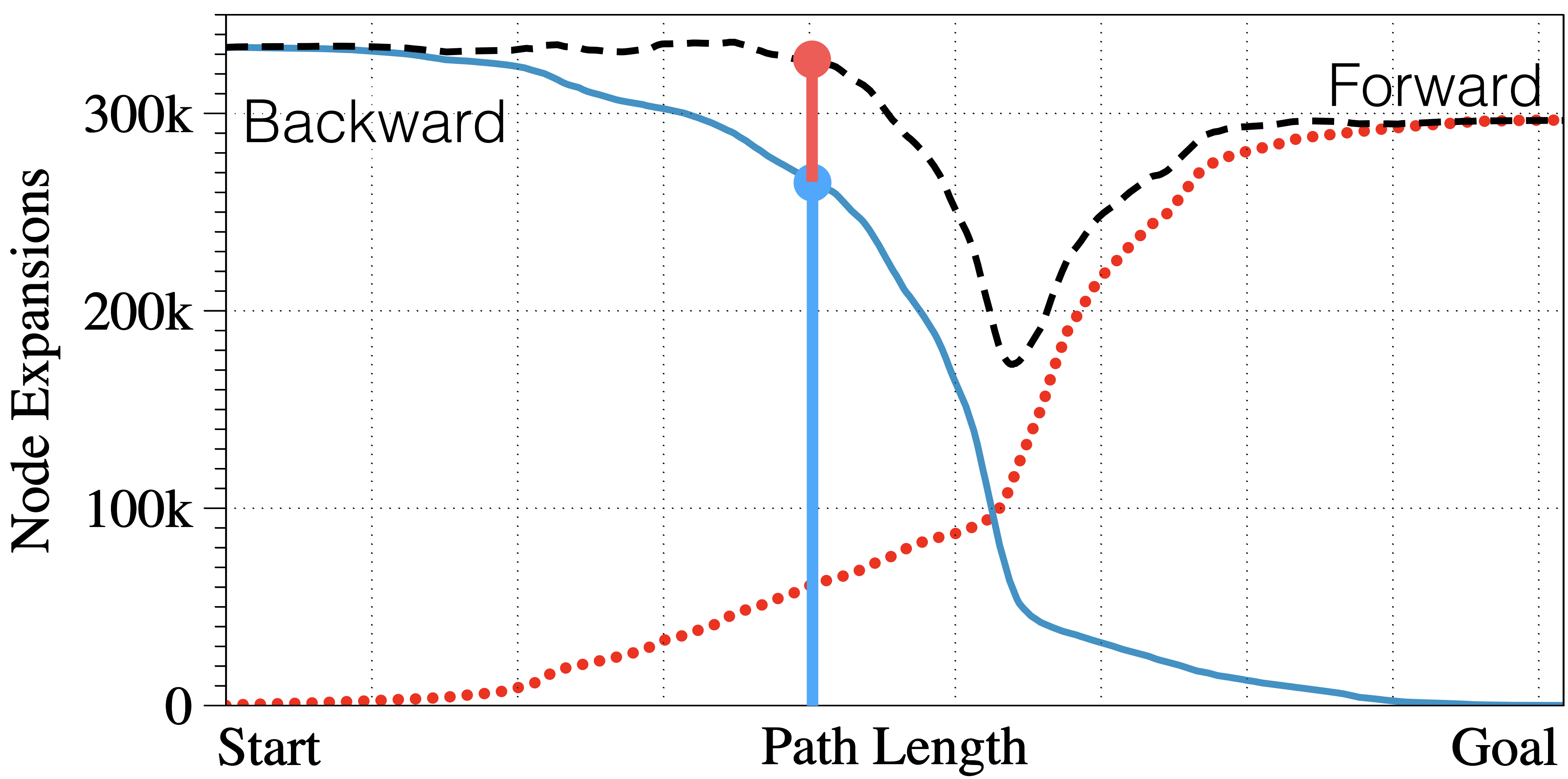
Node Expansions
300k
200k
100k
0
Backward
Forward
Start
Path Length
Goal

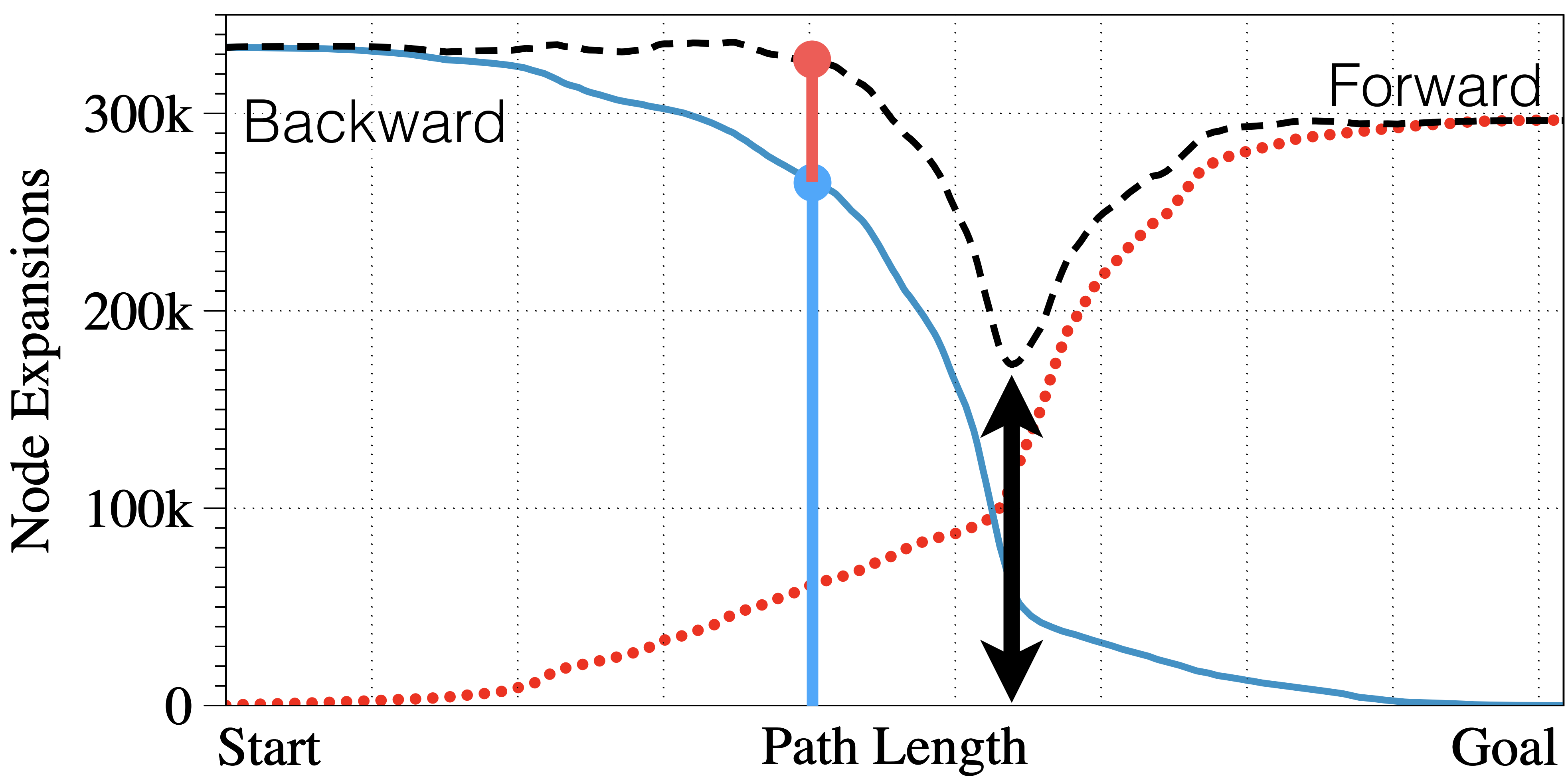


Node Expansions
300k
200k
100k
0
Backward
Forward
Start
Path Length
Goal



Node Expansions
300k
200k
100k
0
Backward
Forward
Start
Path Length
Goal


Node Expansions
300k
200k
100k
0
Backward
Forward
Start
Path Length
Goal



Node Expansions
300k
200k
100k
0
Backward
Forward
Start
Path Length
Goal

Fractional MM

- Takes a parameter f
  - Cost of the state space to explore in each direction
  - Costs correspond to different vertex covers

- *We can (offline) compute the best algorithm for a given search problem*

The Minimal Set of States that Must be Expanded in a Front-to-end Bidirectional Search,
Eshed Shaham, Ariel Felner, Jingwei Chen and Nathan R. Sturtevant,
Symposium on Combinatorial Search (SoCS), **2017**

Vertex Cover on a Bipartite Graph

- Approximation algorithm:
  - Repeat until all vertices covered
    - Choose any edge/line with uncovered vertex
    - Place both states into vertex cover

- Gives 2x approximation to optimal vertex cover
  - (Papadimitriou & Steiglitz, 1982)

Using this algorithm

- We don’t know the full graph ahead of time

Using this algorithm

- We don't know the full graph ahead of time
  - Build the graph as we go

Using this algorithm

- We don’t know the full graph ahead of time
  - Build the graph as we go
- We don’t know the optimal solution cost

Using this algorithm

- We don't know the full graph ahead of time
  - Build the graph as we go
- We don't know the optimal solution cost
  - Must estimate C*

Using this algorithm

- We don’t know the full graph ahead of time
  - Build the graph as we go
- We don’t know the optimal solution cost
  - Must estimate C*
- We must avoid re-expanding states

Using this algorithm

- We don’t know the full graph ahead of time
  - Build the graph as we go
- We don’t know the optimal solution cost
  - Must estimate C*
- We must avoid re-expanding states
  - Carefully order state expansions

Using this algorithm

- We don’t know the full graph ahead of time
  - Build the graph as we go
- We don’t know the optimal solution cost
  - Must estimate C*
- We must avoid re-expanding states
  - Carefully order state expansions
- Computing lb(*u*, *v*) could be expensive

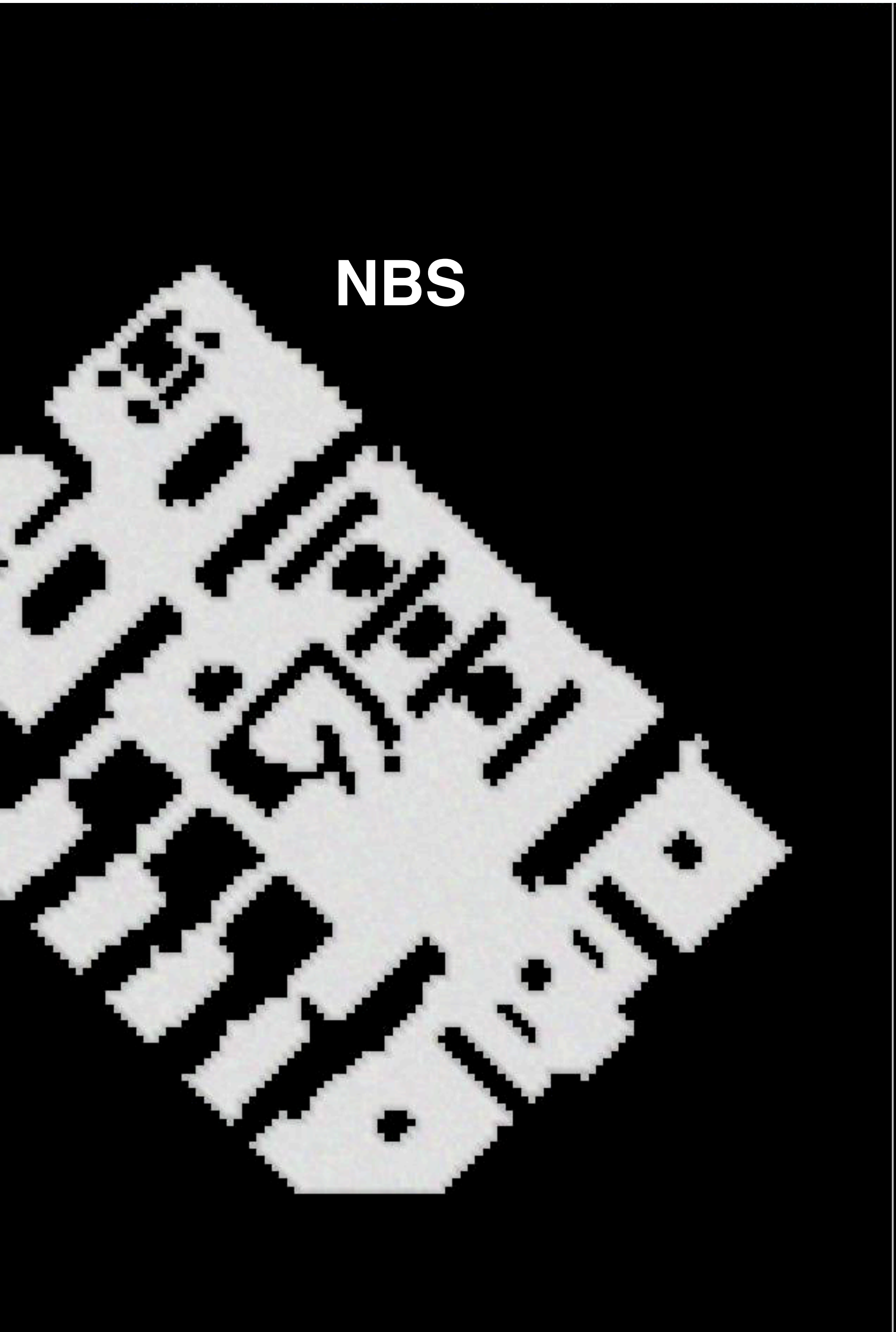
Using this algorithm

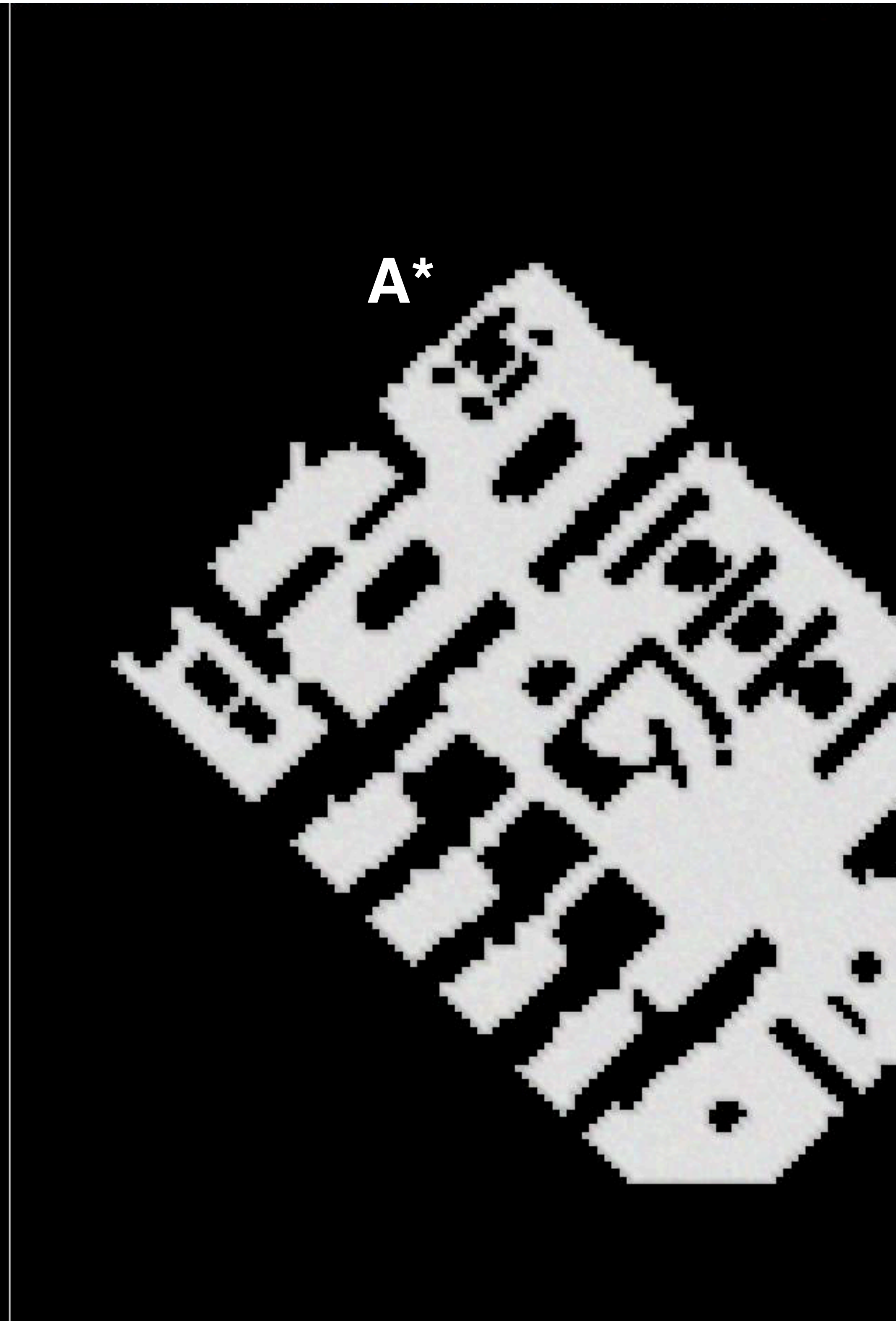
- We don’t know the full graph ahead of time
  - Build the graph as we go
- We don’t know the optimal solution cost
  - Must estimate C*
- We must avoid re-expanding states
  - Carefully order state expansions
- Computing lb(*u*, *v*) could be expensive
  - Efficient data structures

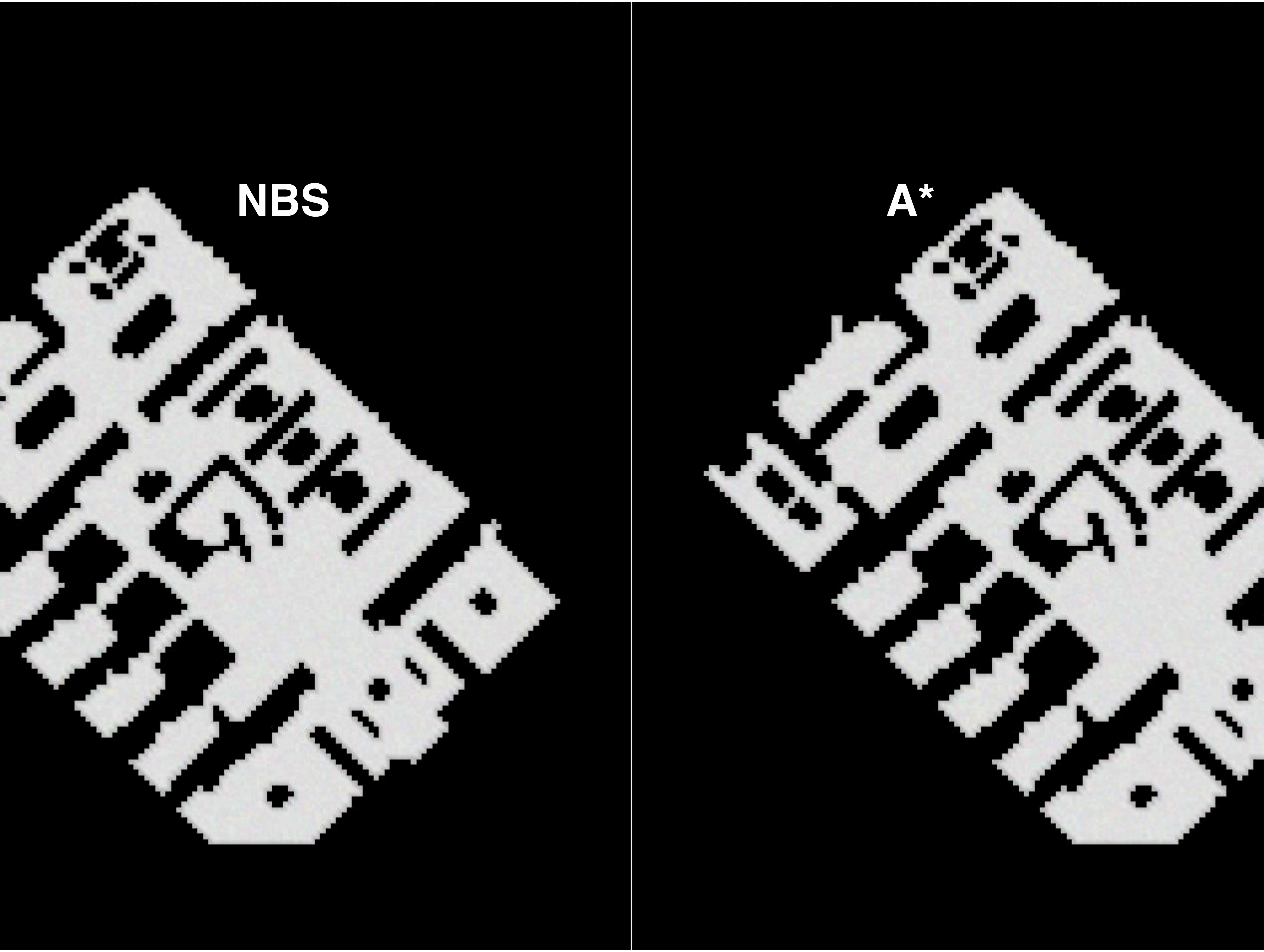
NBS

- Put start/goal onto forward/backward priority queues
- While forward/backward not empty
  - Among all state on queues:
    - Select the pair with lowest *lb*
    - Expand **both** of them
  - Terminate when *lb* ≥ best path
- Gives 2x bound on optimal number of expansions
  - Bound is tight

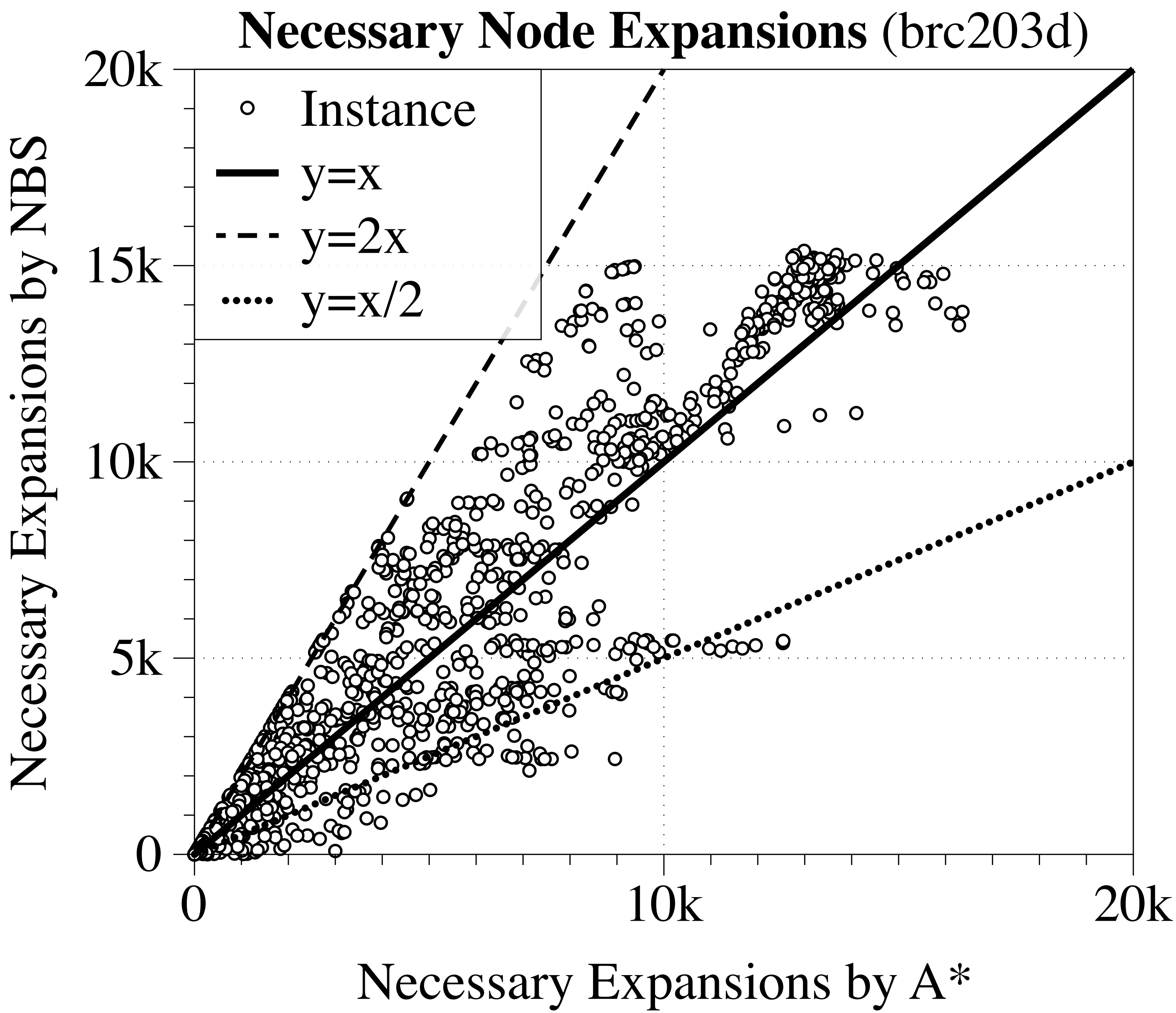
Front-to-End Bidirectional Heuristic Search with Near-Optimal Node Expansions, Jingwei Chen, Robert C. Holte, Sandra Zilles and Nathan R. Sturtevant, International Joint Conference on Artificial Intelligence (IJCAI), **2017**


NBS

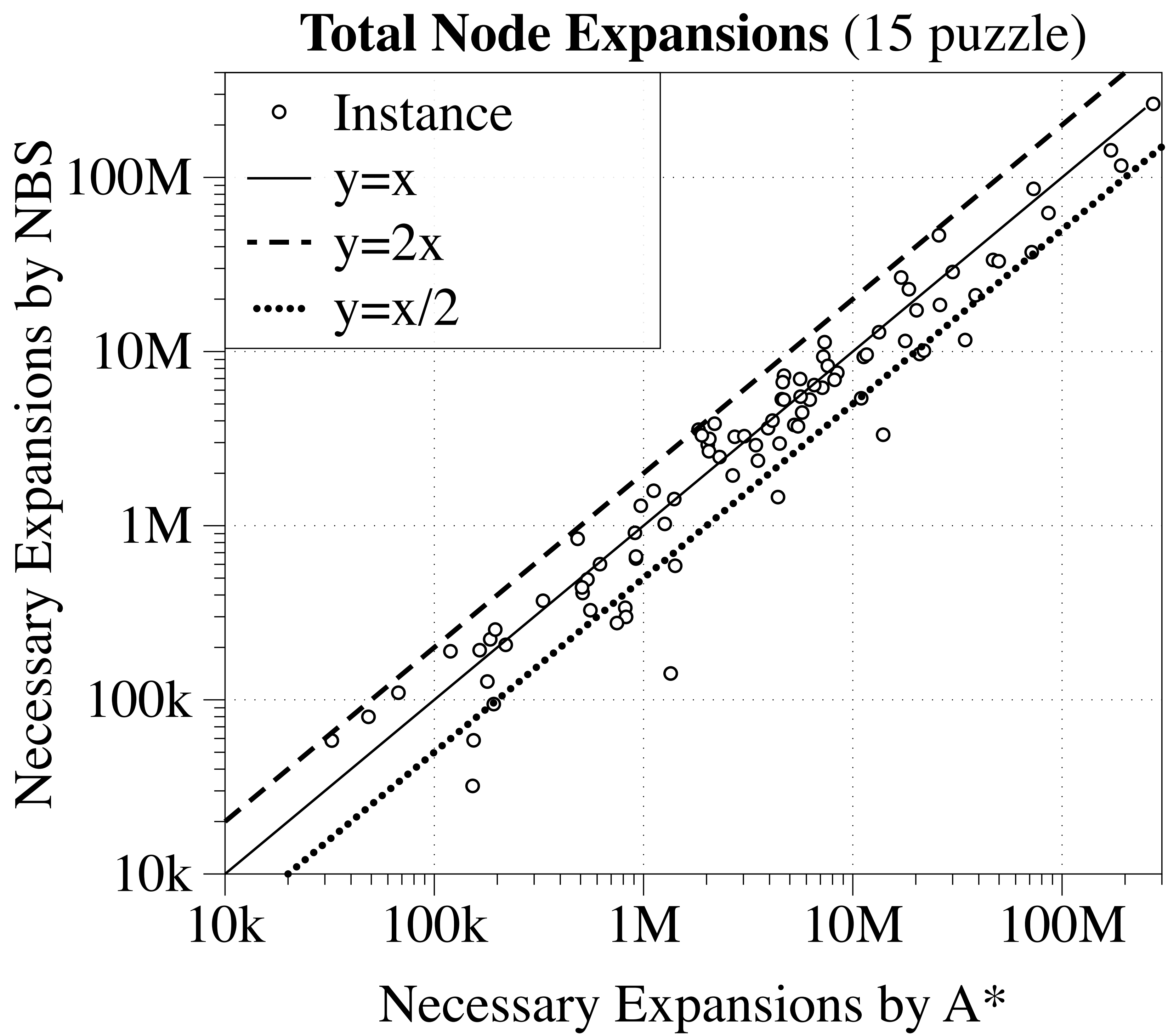

A*



NBS
A*



Necessary Node Expansions (brc203d)
Instance
y=x
y=2x
y=x/2
Necessary Expansions by NBS
Necessary Expansions by A*
0
5k
10k
15k
20k
0
10k
20k


Total Node Expansions (15 puzzle)
Instance
y=x
y=2x
y=x/2
Necessary Expansions by NBS
10k
100k
1M
10M
100M
Necessary Expansions by A*

Summary

- Theory
  - First definition of necessary node expansions
  - fMM - implements optimal bidirectional search
- Practice
  - Near-optimal approach (NBS)
  - Node expansions are bounded by 2x optimal
- Demos & videos will appear at:
  - https://www.movingai.com

Open Questions

- What can we learn about bidirectional search from the minimum vertex cover?
- Is there an algorithm with better average performance?
- Efficient front-to-front search?