Parallel Beam Search Algorithms for Domain-Independent Dynamic Programming

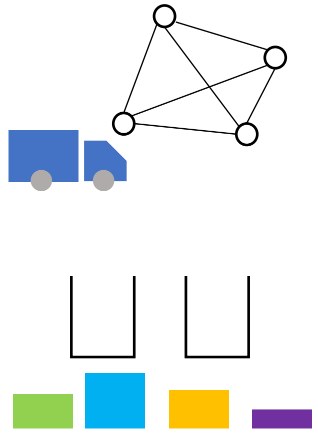
Ryo Kuroiwa and J. Christopher Beck
Toronto Intelligent Decision Engineering Laboratory (TIDEL)
Department of Mechanical and Industrial Engineering,
University of Toronto

Background

Domain-Independent Dynamic Programming (DIDP)

A user can solve a combinatorial optimization problem by formulating a Dynamic Programming (DP) model

Combinatorial Optimization Problem



DP Model

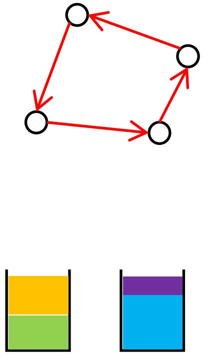
$$\text{compute } V(N \setminus \{0\}, 0)$$
$$V(U.i) = \min_{j \in U} c_{ij} + V(U \setminus \{j\}, j)$$
$$V(\emptyset, i) = c_{i0}$$

Modeling Interface (Python or YAML)

General-Purpose DP Solver

Current solvers (in Rust) use **heuristic search**

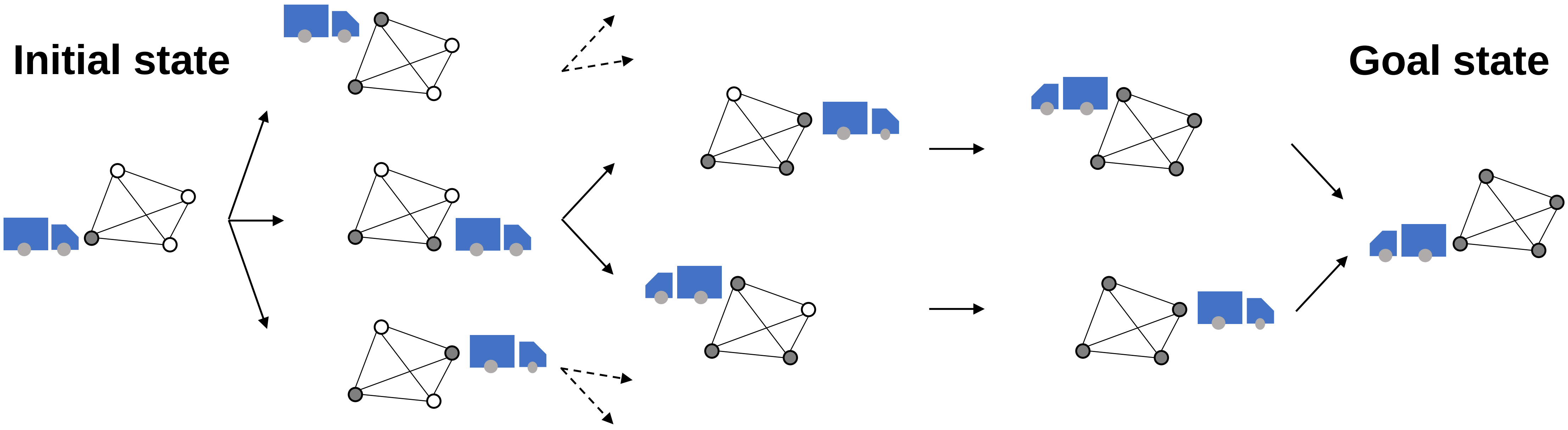
Solution



Recently proposed by us [Kuroiwa and Beck 2023b]

Example: DP Model for TSP

- TSP: Minimize the total travel cost to visit all customers and return
- DP: State space representation of the problem
  Customers are visited one by one in TSP



State transition graph for the DP model of TSP

Example of DIDP with Python

```python
import didppy as dp

model = dp.Model(maximize=False)
customer = model.add_object_type(number=4)
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])
u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)

for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=c[i, j] + dp.IntExpr.state_cost(),
        effects=[(u, u.remove(j)), (i, j)],
        preconditions=[u.contains(j)],
    )
    model.add_transition(visit)

model.add_base_case([u.is_empty()], cost=c[i, 0])

model.add_dual_bound(0)

solver = dp.CABS(model, threads=32)
solution = solver.search()
```

Define constants and state variables

Define transitions between states

Define goal conditions

Call a solver

Example of DIDP with Python

```
import didppy as dp

model = dp.Model(maximize=False)
customer = model.add_object_type(number=4)
c = model.add_int_table([[0, 3, 4, 5], [3, 0, 5, 4], [4, 5, 0, 3], [5, 4, 3, 0]])
u = model.add_set_var(object_type=customer, target=[1, 2, 3])
i = model.add_element_var(object_type=customer, target=0)

for j in range(1, 4):
    visit = dp.Transition(
        name="visit {}".format(j),
        cost=c[i, j] + dp.IntExpr.state_cost(),
        effects=[(u, u.remove(j)), (i, j)],
        preconditions=[u.contains(j)],
    )
    model.add_transition(visit)

model.add_base_case([u.is_empty()], cost=c[i, 0])

model.add_dual_bound(0)

solver = dp.CABS(model, threads=32)
solution = solver.search()
```

Contribution of this paper

Define constants and state variables

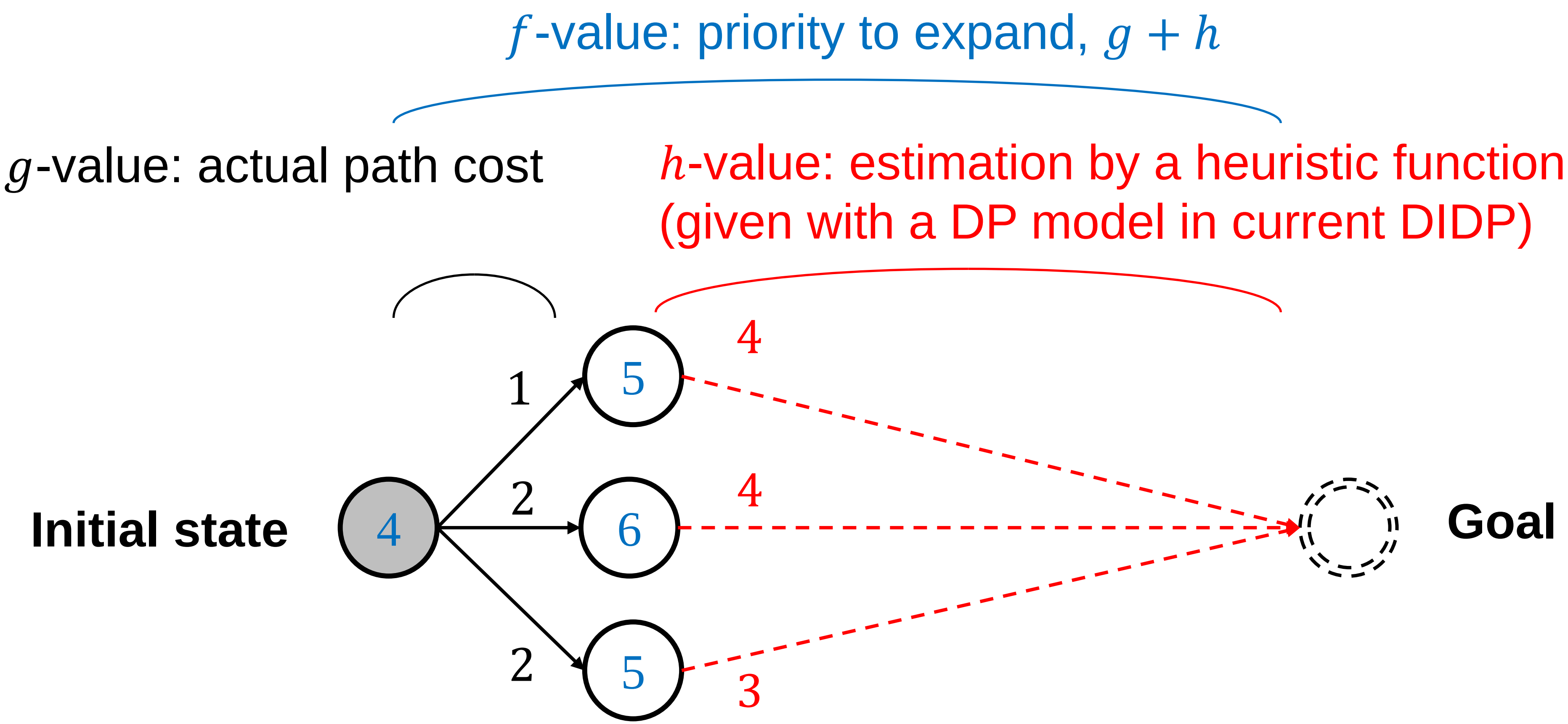
Define transitions between states

Define goal conditions

Call a solver

Solving DP with Heuristic Search

Solving the DP model by finding a path in the state transition graph

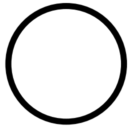


CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

$b = 2$

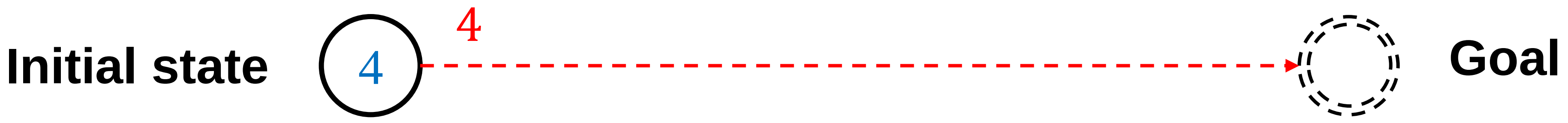
Initial state



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

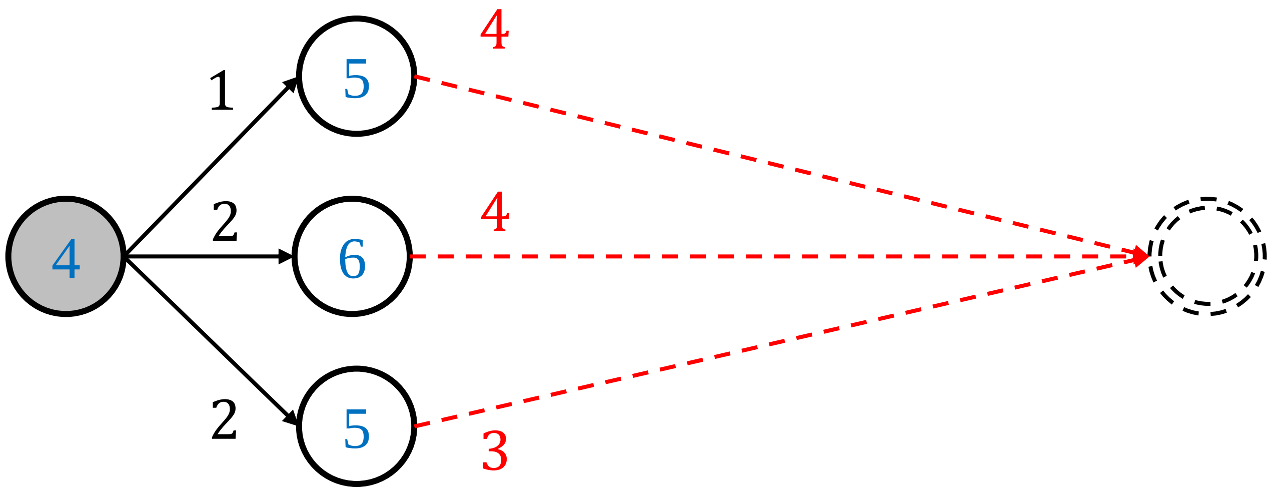
$b = 2$



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

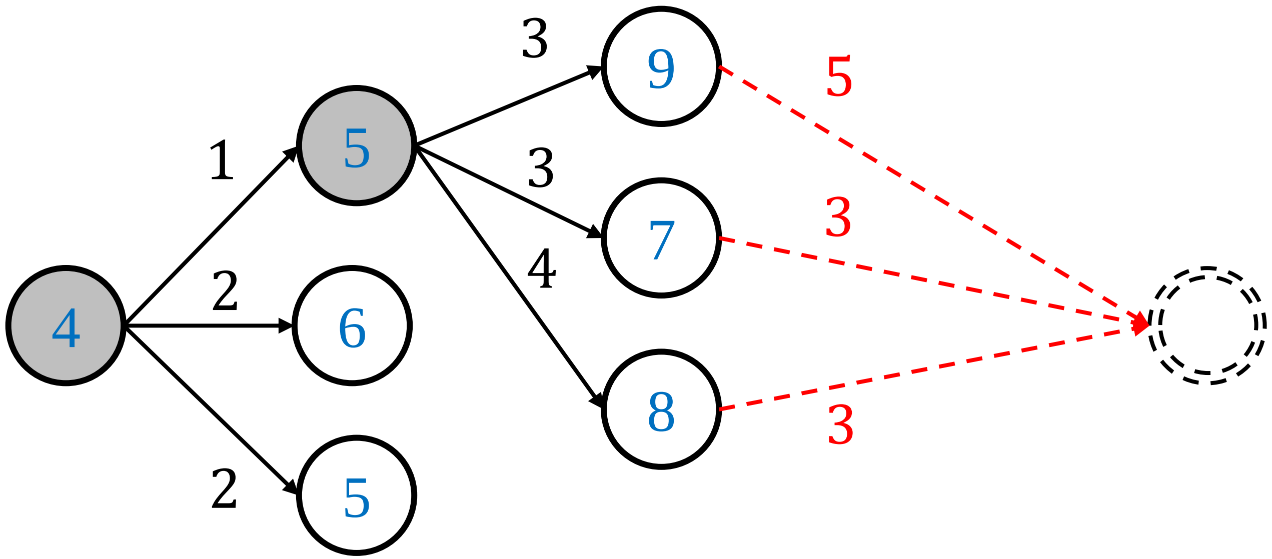
$b = 2$



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

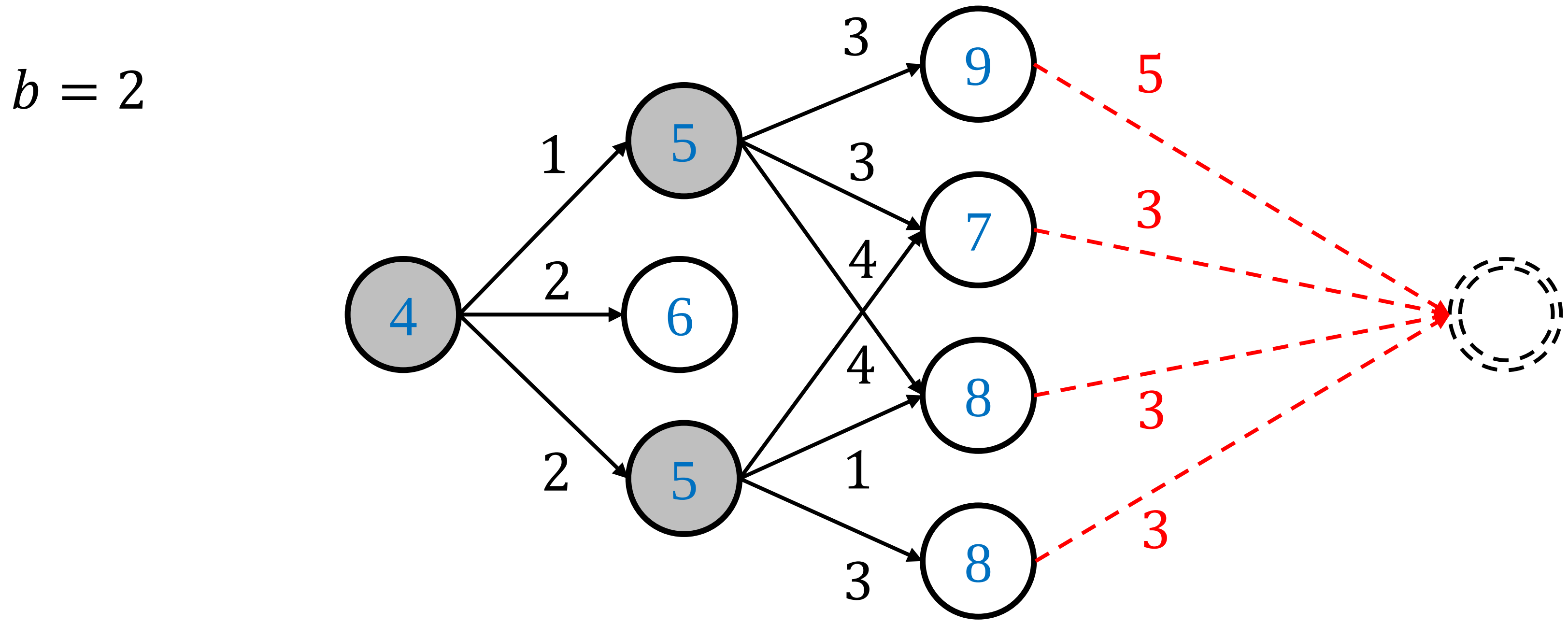
- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

$b = 2$



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

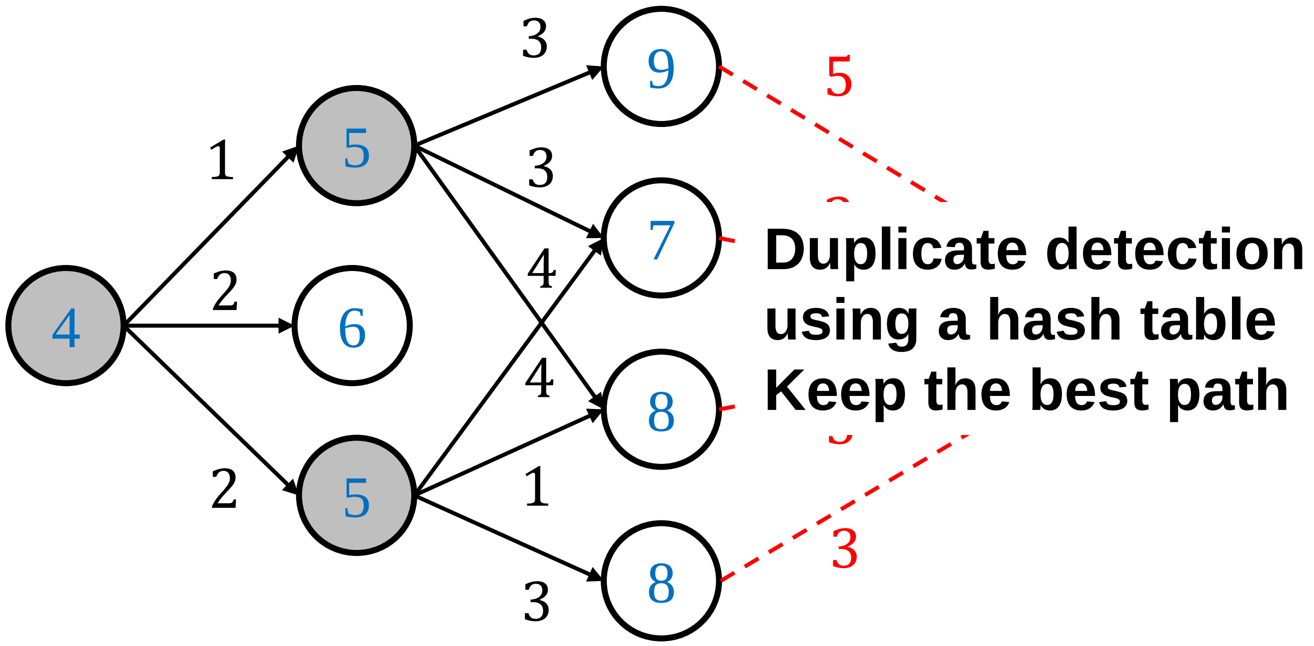
- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

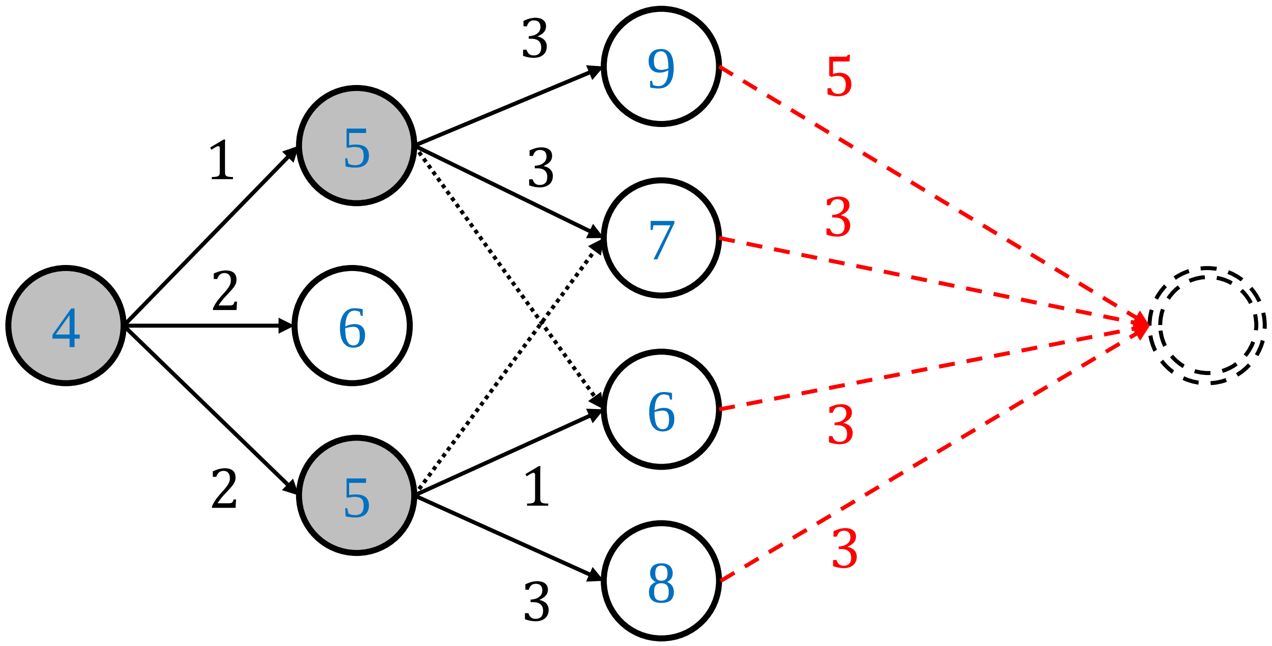
$b = 2$



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

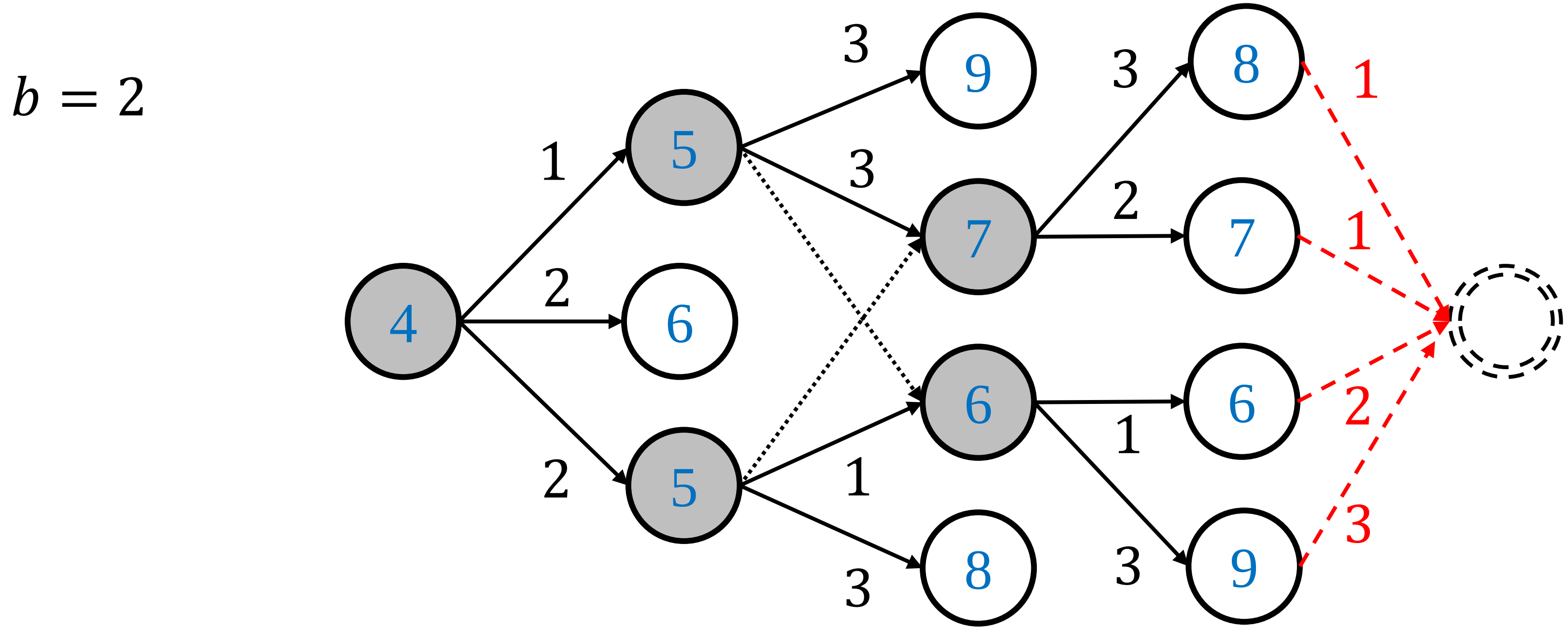
- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

$b = 2$



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

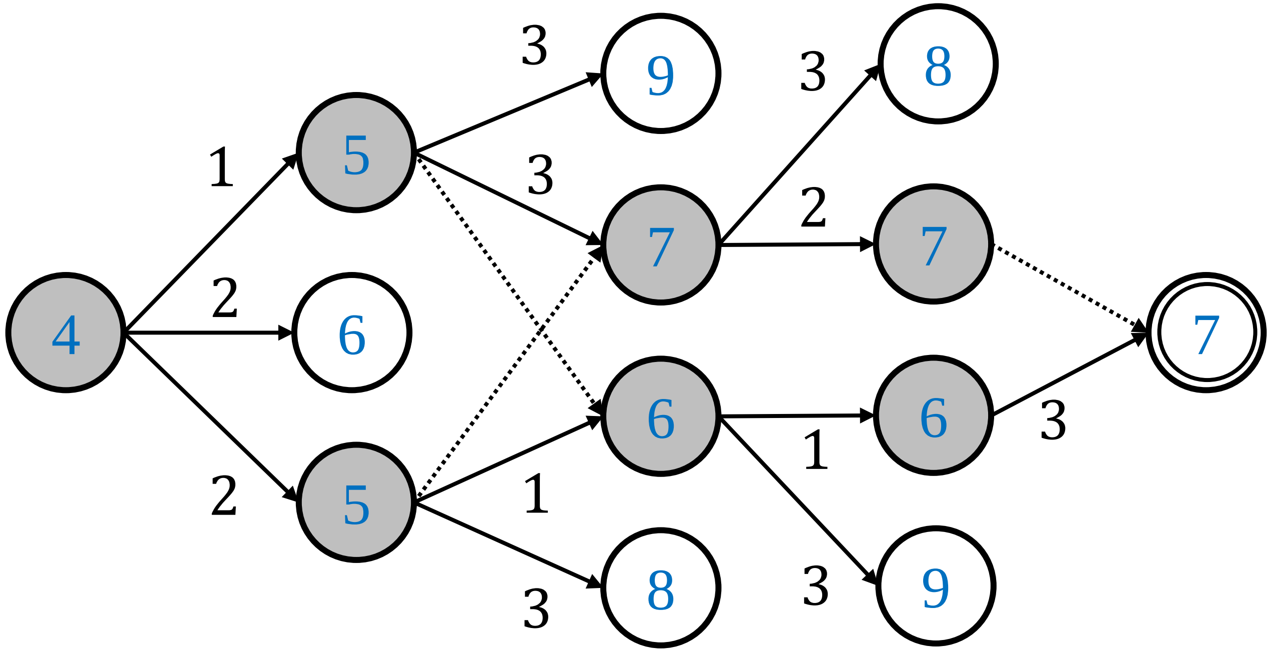
- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

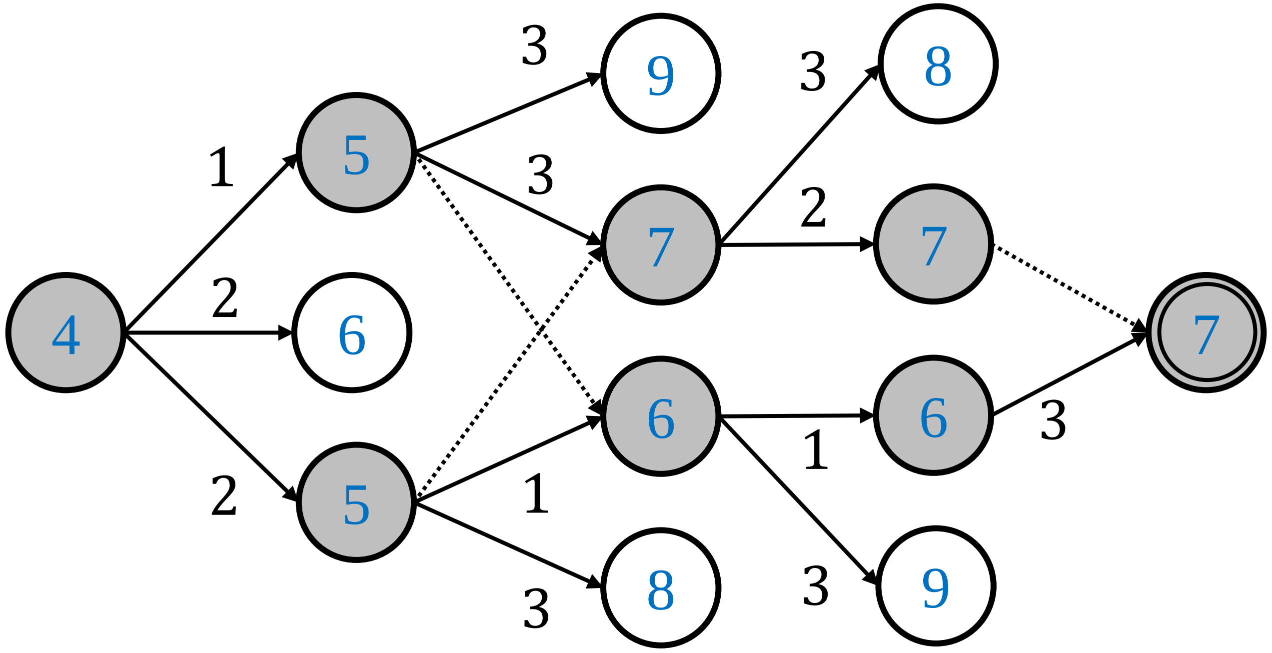
$b = 2$



CABS: SOTA DIDP Solver [Kuroiwa and Beck 2023c]

- Beam search expands the b states minimizing f-values in each layer
- Complete Anytime Beam Search (CABS) repeats beam search with increasing b until finding an optimal solution [Zhang 1998]

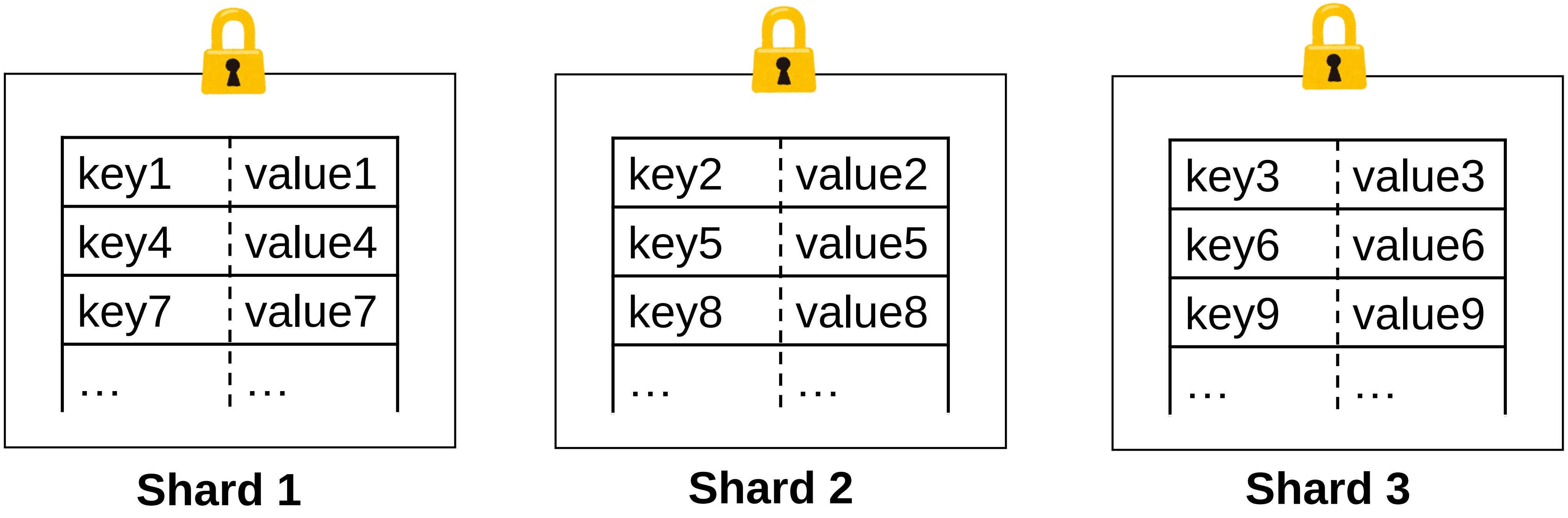
$b = 2$



Parallel Beam Search Algorithms

Approach 1: Shared Beam Search (SBS)

- Expand the best b states (obtained by **parallel sort**) in parallel
- Use a **concurrent hash table** for duplicate detection
  Divided into multiple shards, and each shard has a lock



Similar to problem-specific parallel beam search by Frohner+ (2023)

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

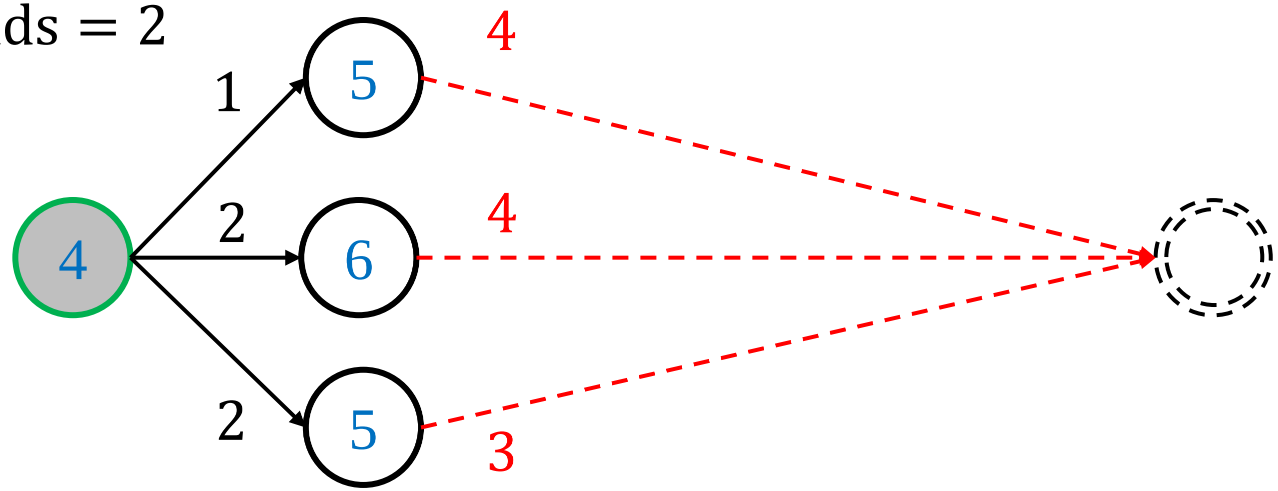
Assigned to a thread

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

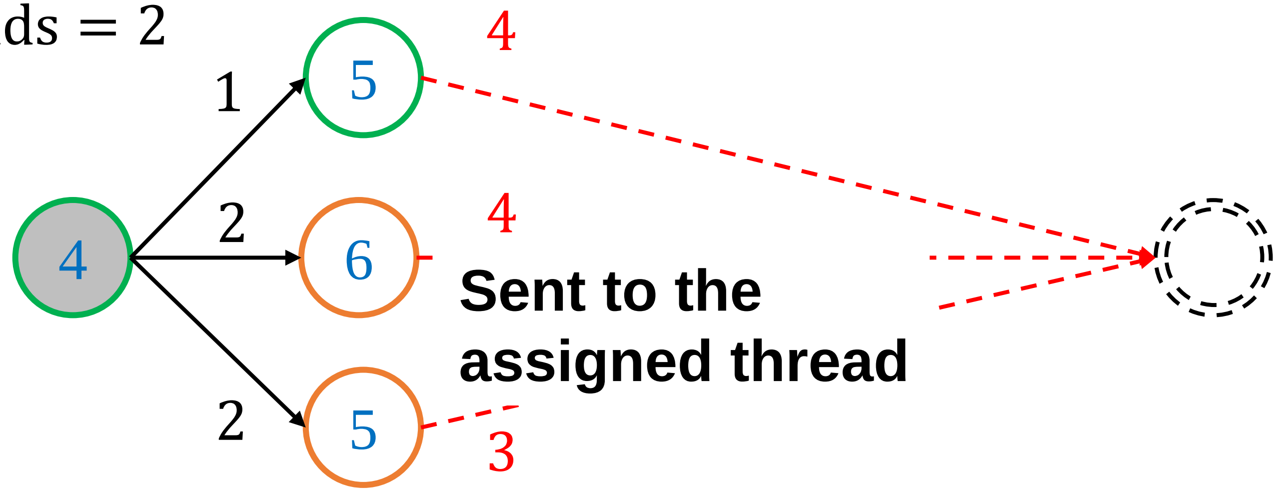


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

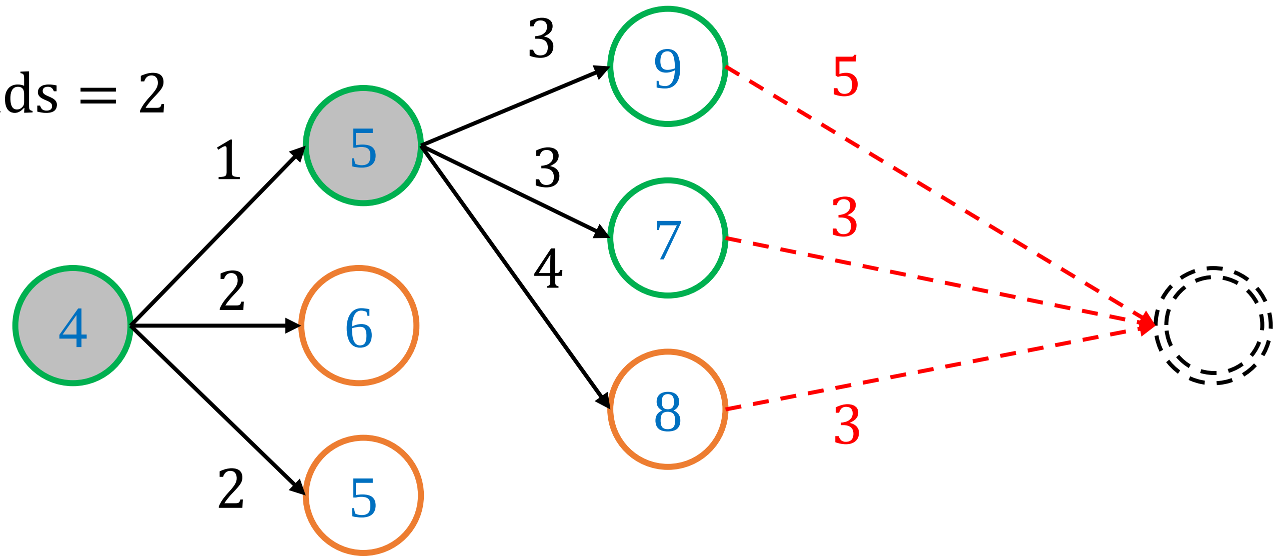


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

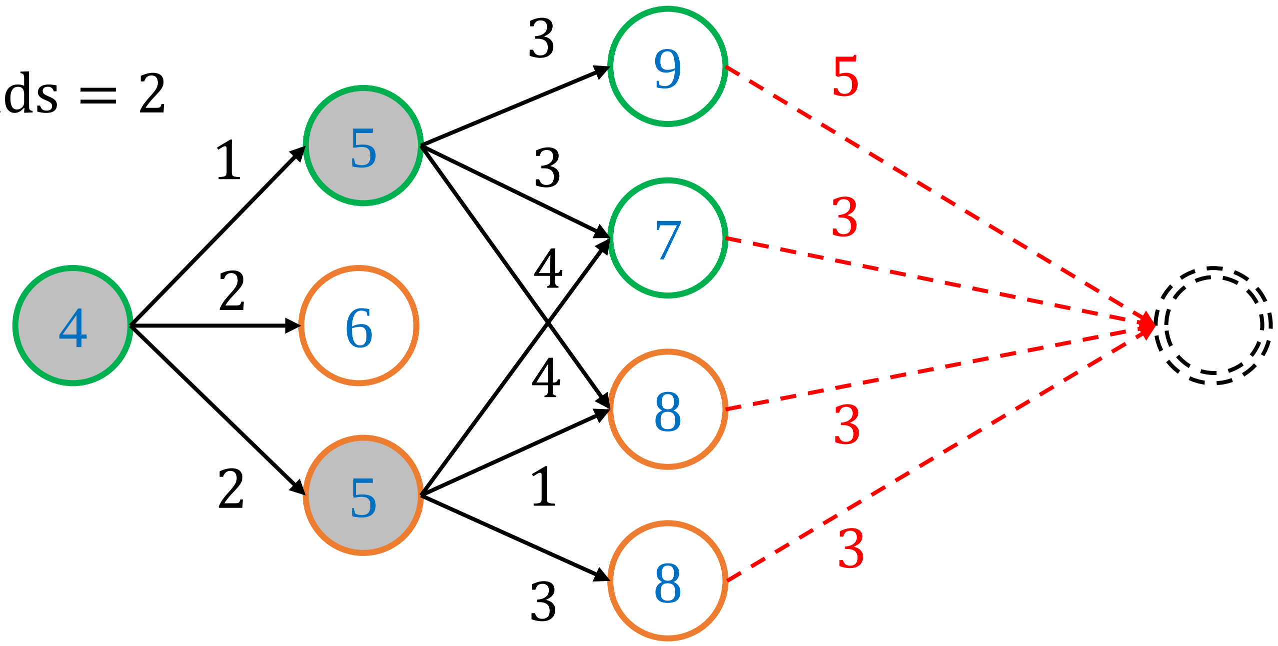


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\mathrm{threads}}$ states

$b = 2, \#\mathrm{threads} = 2$

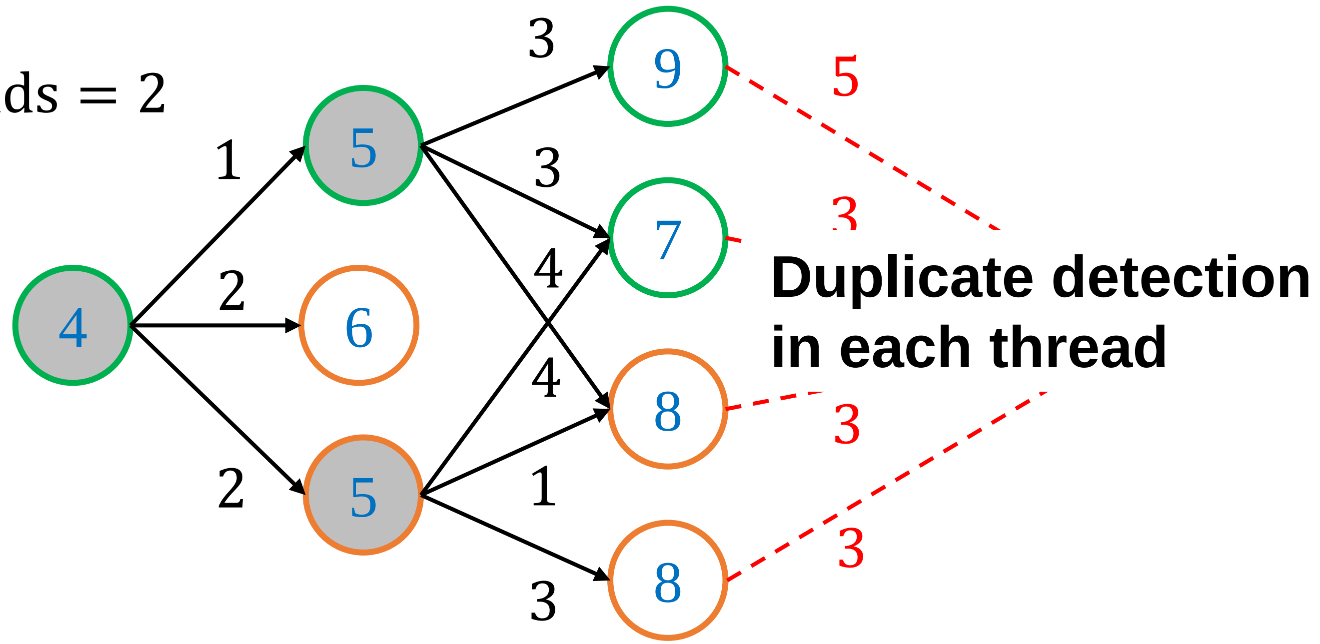


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

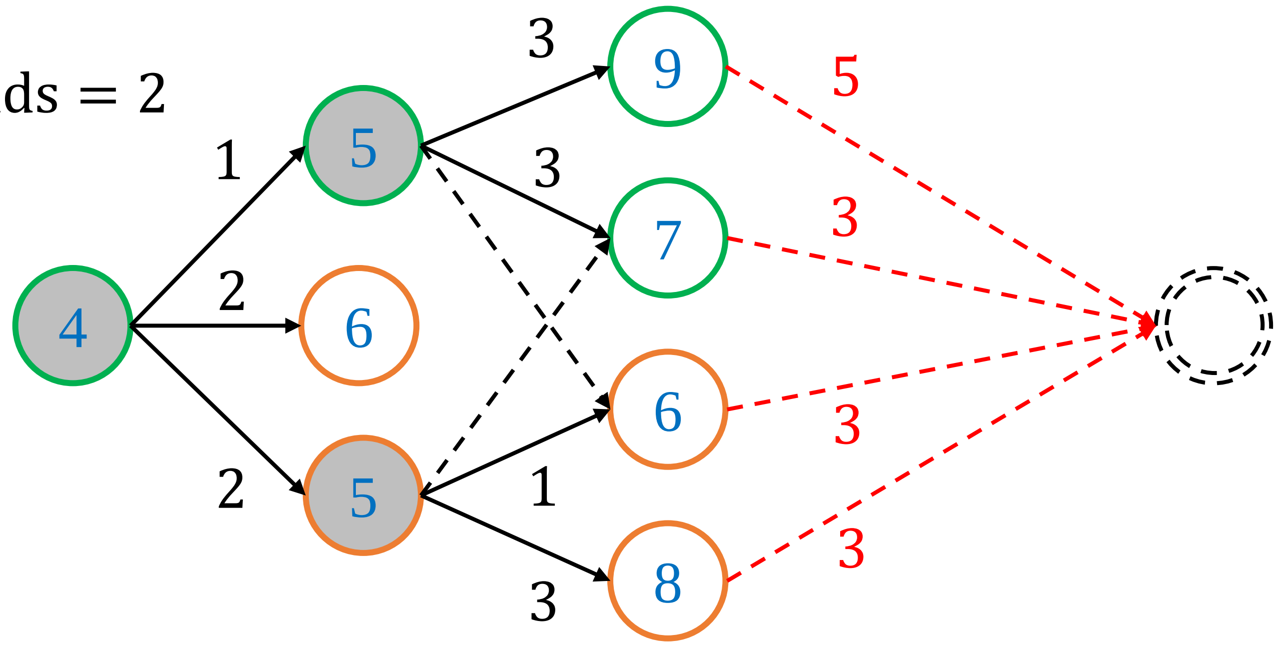


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\mathrm{threads}}$ states

$b = 2, \#\mathrm{threads} = 2$

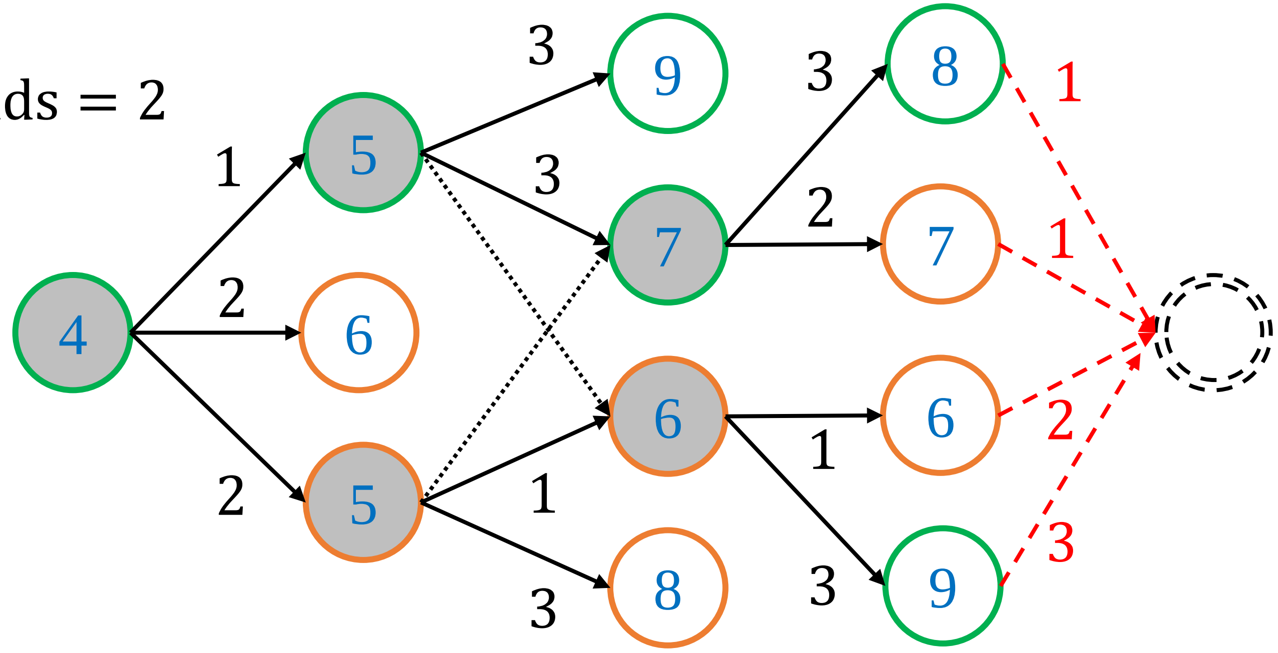


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

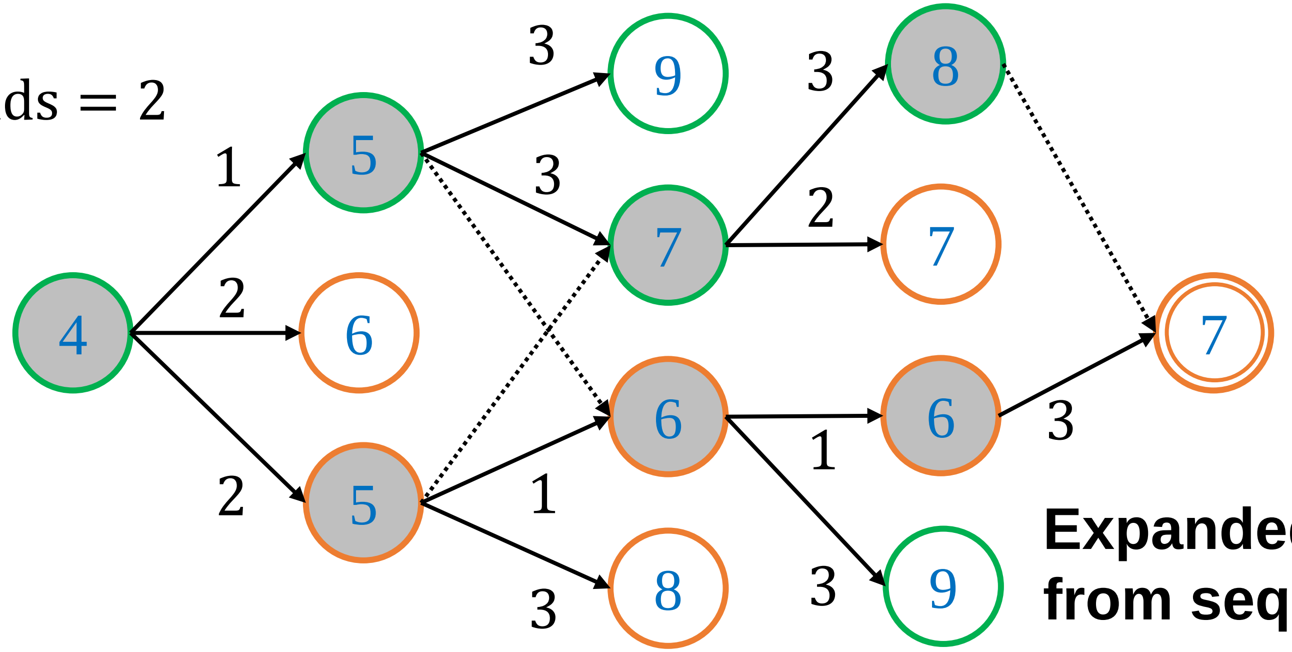


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$



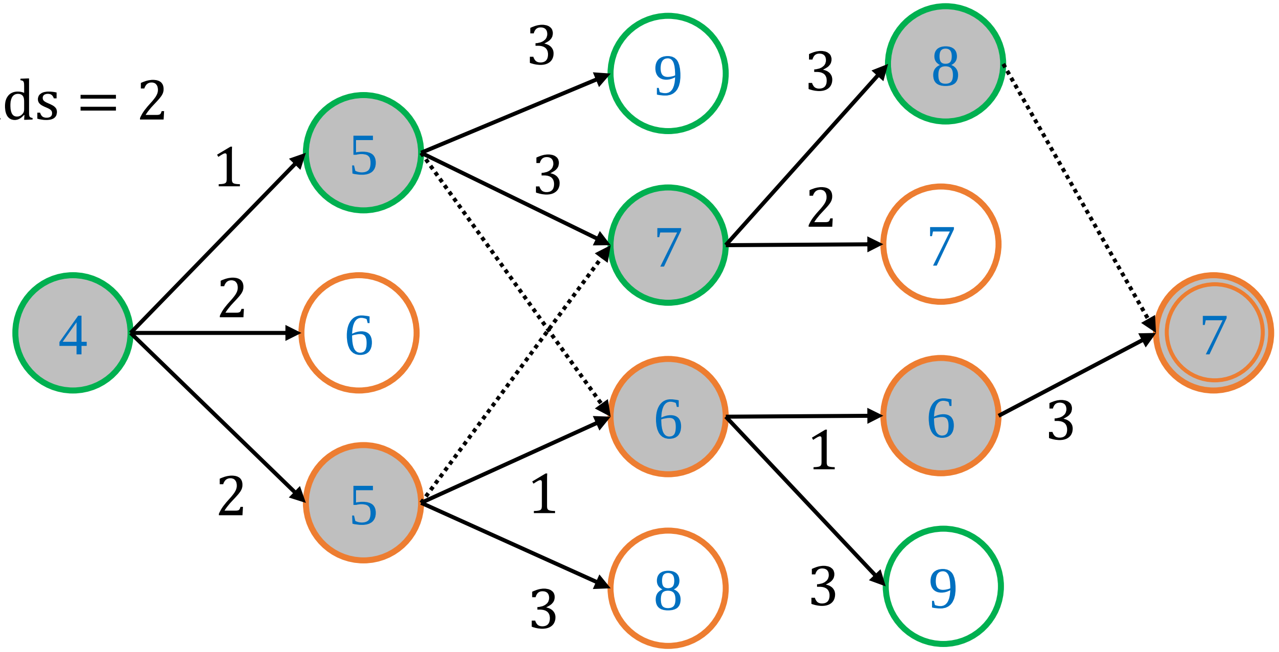
Expanded states are different from sequential beam search

Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

Approach 2: Hash Distributed Beam Search (HDBS)

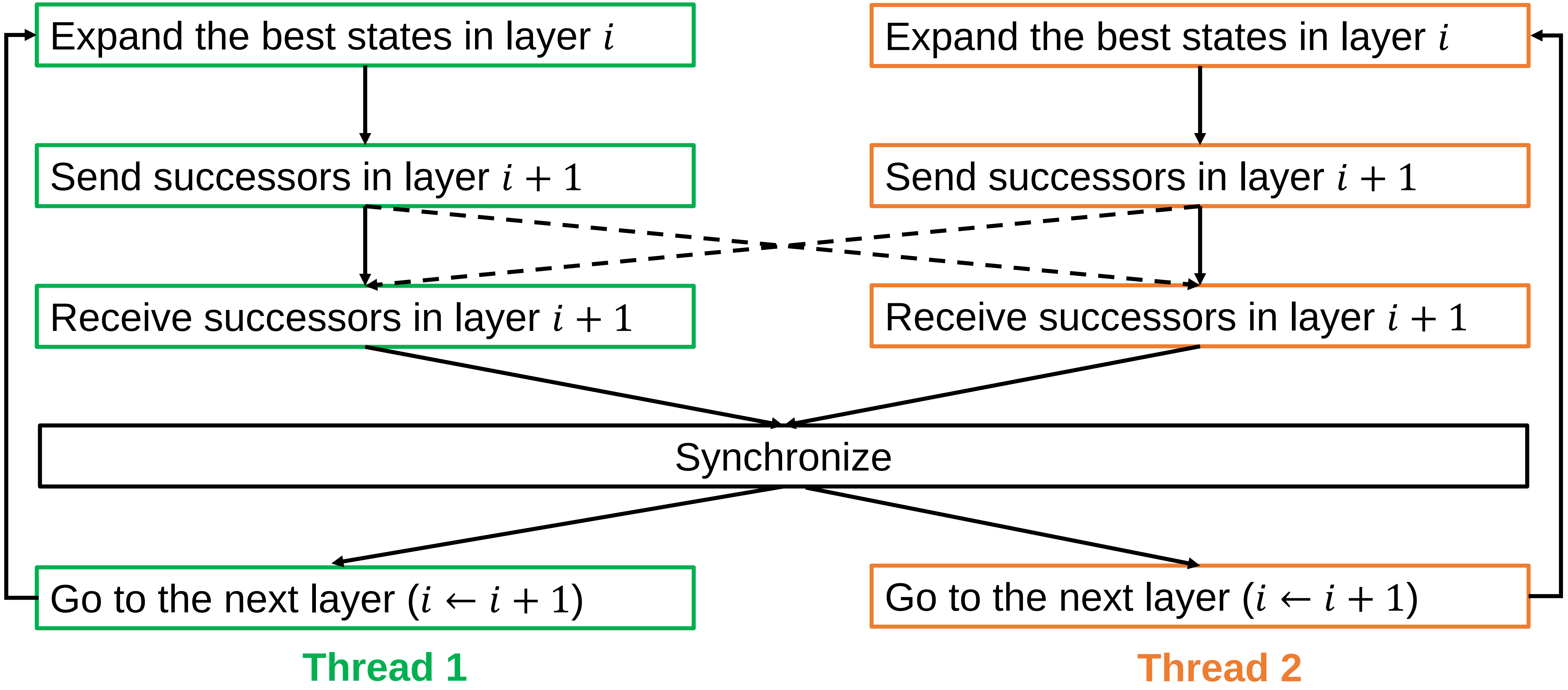
- Send a state to a thread determined by its **hash value** using **message passing** (duplicate states sent to the same thread)
- Each thread **locally detects duplicates** and expands $\frac{b}{\#\text{threads}}$ states

$b = 2, \#\text{threads} = 2$

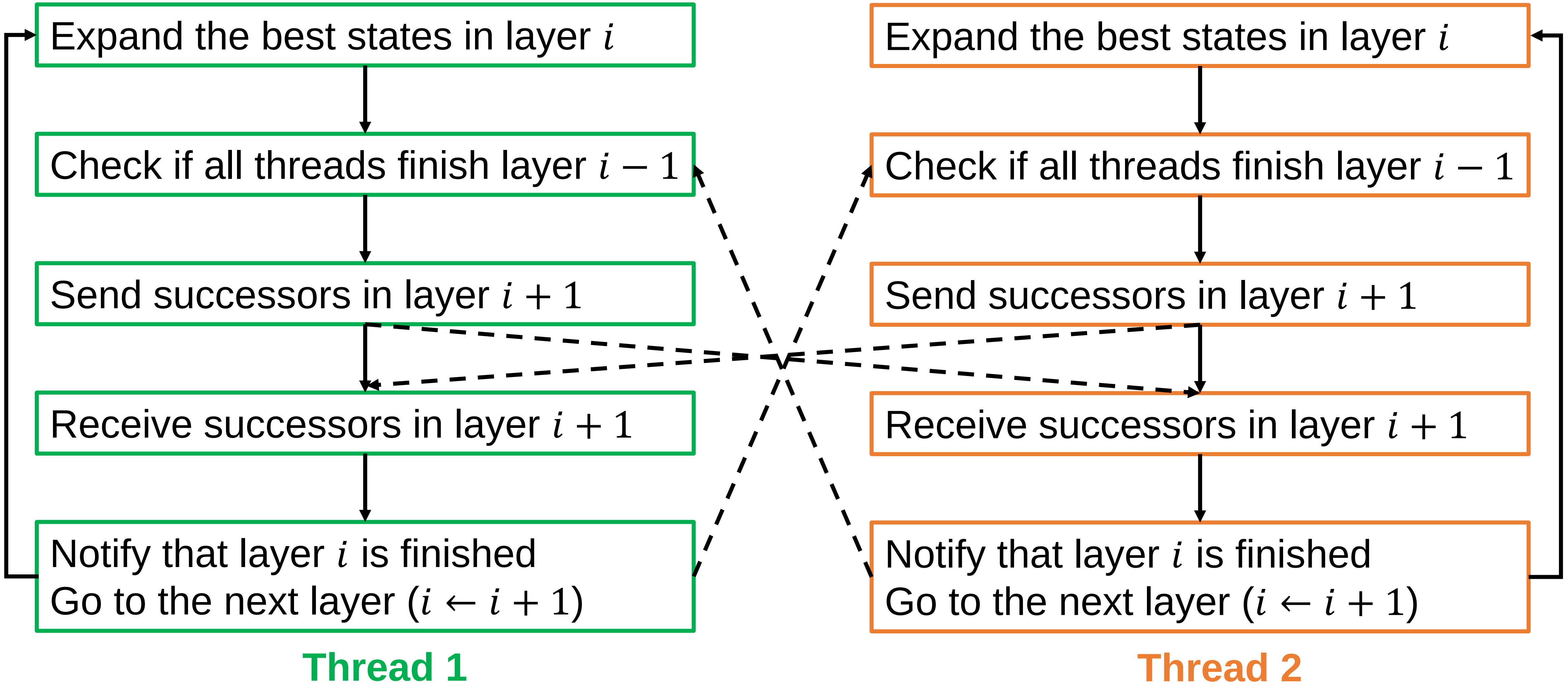


Adaptation of Hash Distributed A* [Kishimoto+ 2013] to beam search

HDBS1: Immediate Layer Synchronization

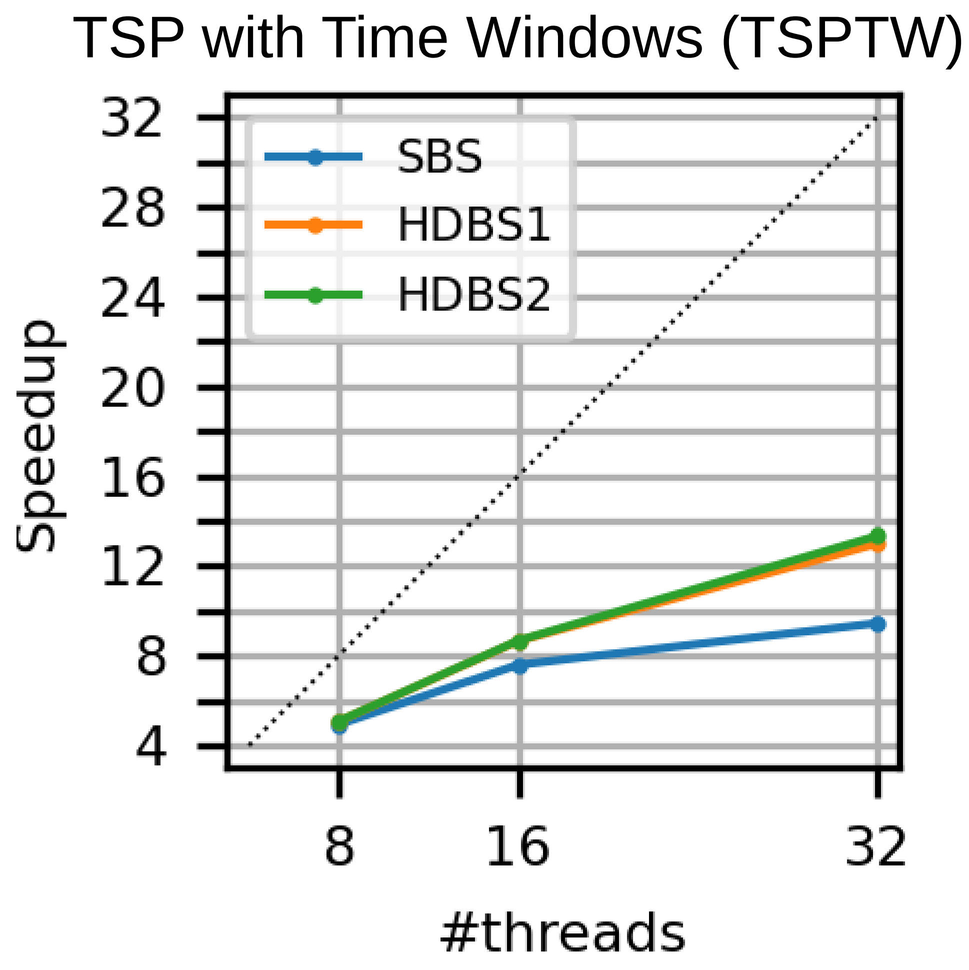


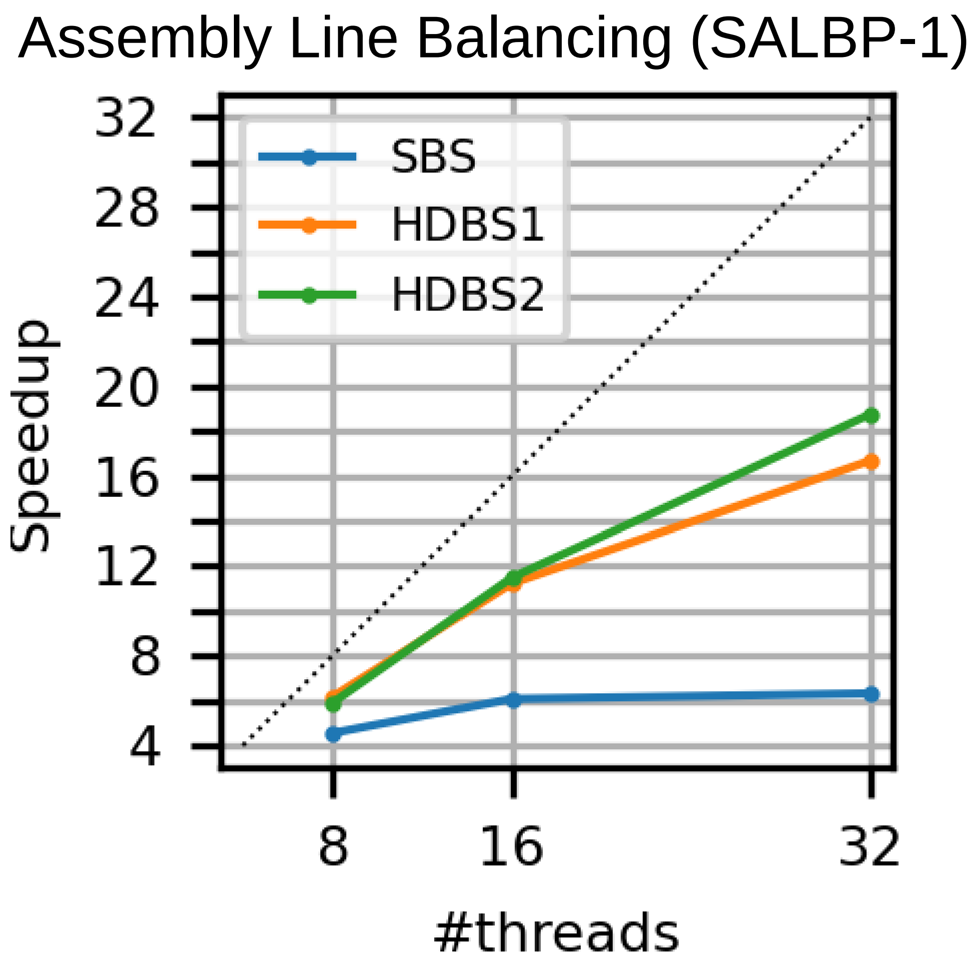
HDBS2: Delayed Layer Synchronization



Experimental Evaluation

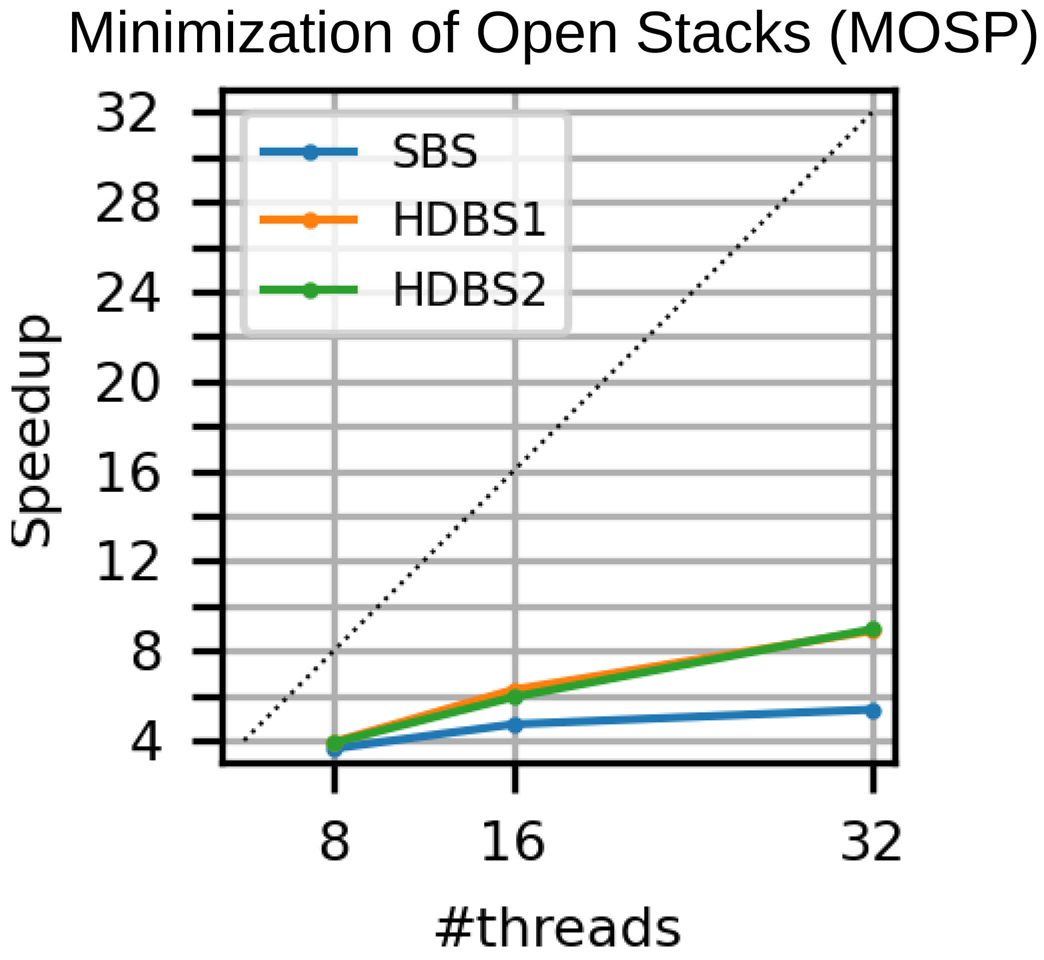
SBS vs. HDBS: Mean Speedup against Single Thread

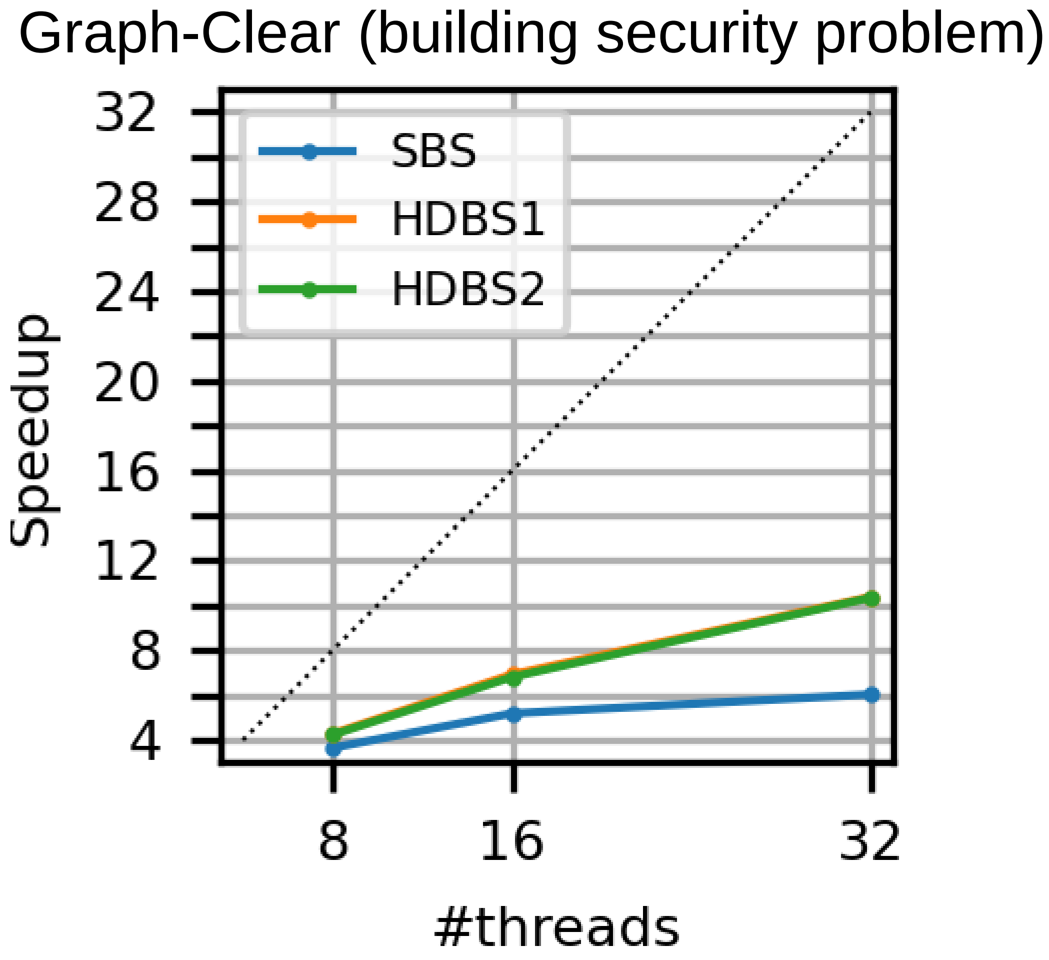




Used with CABS and measure the time to solve optimally (limits: 5-min and 188GB)

SBS vs. HDBS: Mean Speedup against Single Thread





Used with CABS and measure the time to solve optimally (limits: 5-min and 188GB)

DIDP vs. Commercial Parallel Optimization Solvers

Problem	Description	Gurobi	CPO	DIDP (HDBS2)
TSPTW (340)	TSP with time	239/4.2	27/0.1	**262**/13.3
CVRP (207)	vehicle routing	**29**/5.3	0/ -	8/ 9.3
SALBP-1 (2100)	line balancing	1351/1.3	1581/1.4	**1826**/18.8
Bin Packing (1615)	bin packing	1192/6.4	**1251**/9.2	1239/39.6
MOSP (570)	manufacturing	238/3.1	397/0.3	**531**/ 9.0
Graph-Clear (135)	building security	16/2.0	4/3.2	**113**/10.3

#optimally solved / mean speed up

- Resources: 32 threads, 5-min, and 188GB
- Gurobi: mixed-integer programming solver
- CPO: IBM ILOG CP Optimizer (constraint programming solver)

Conclusion

- A parallel beam search algorithm, HDBS2, shows good speedup and yields a high-performance parallel combinatorial optimization solver
- Start DIDP with Python: `pip install didppy`

Tutorials and API References

Project Page

GitHub Repo