Massively Parallel A* Search on a GPU

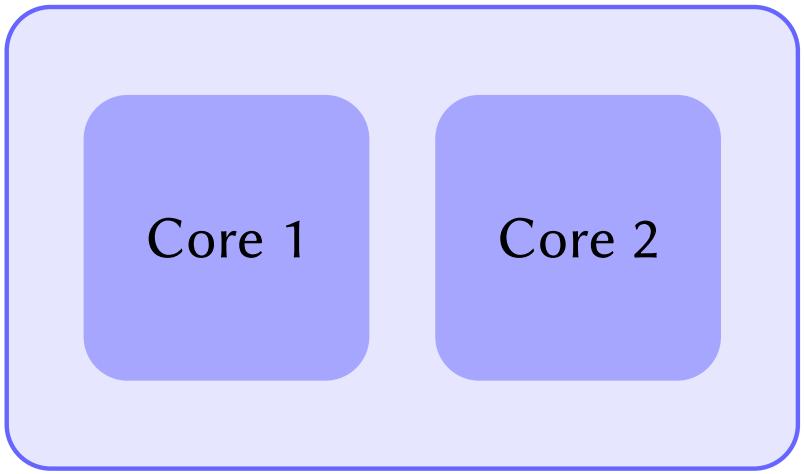
Yichao Zhou Jianyang Zeng

Institute for Interdisciplinary Information Sciences
Tsinghua University, Beijing, P. R. China

Jan, 2015

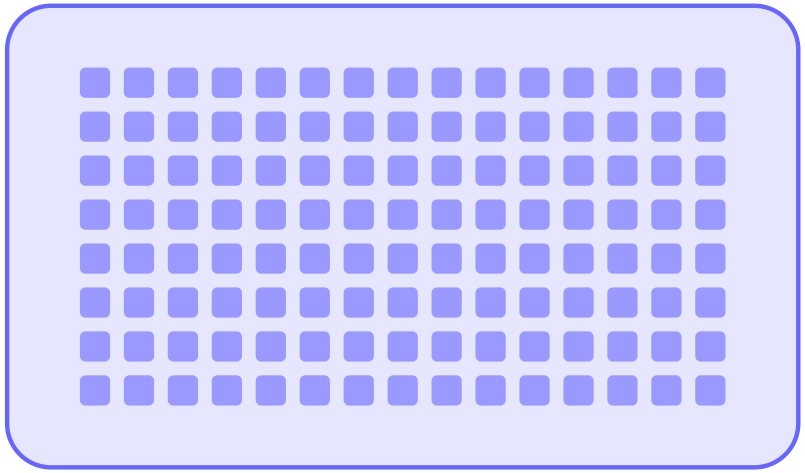
A Brief Concept to GPU Computation

CPU (several cores)



- ☺ Decent single thread performance
- ☹ Limited in parallelism

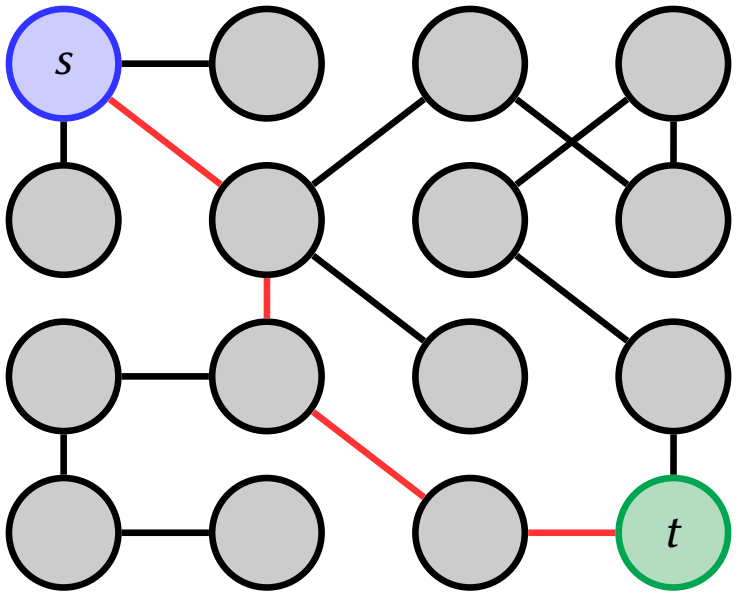
GPU (thousands of cores)



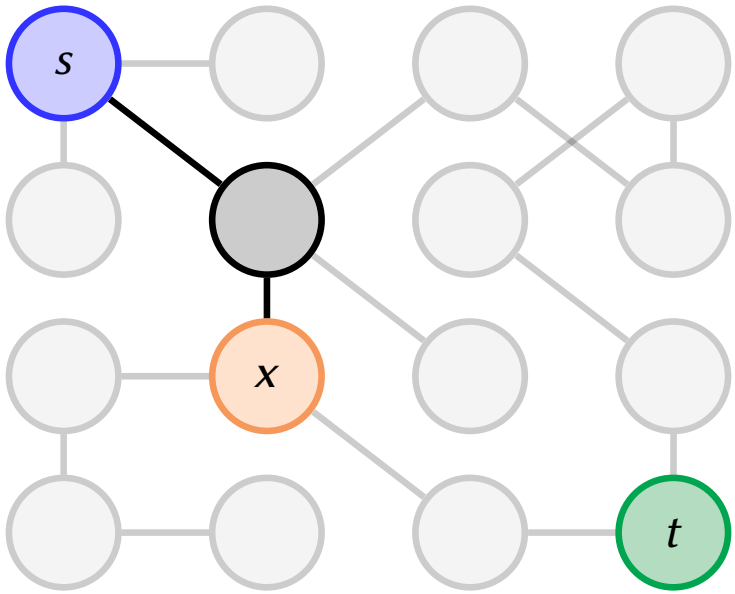
- ☺ More computational units
- ☺ Energy efficient
- ☹ Need massive parallelism

Shortest Path Search Problem

Find the shortest path from the starting node s to the goal node t.



Heuristic Function

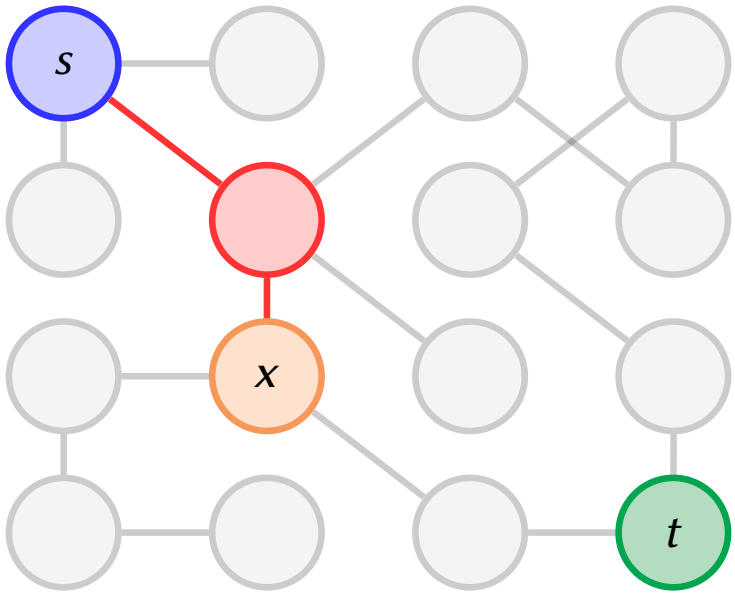


Heuristic Function

$$f(x) = g(x) + h(x)$$

- $g(x)$: distance from the starting node s to node x;
- $h(x)$: *estimated* distance from node x to the goal node t

Heuristic Function

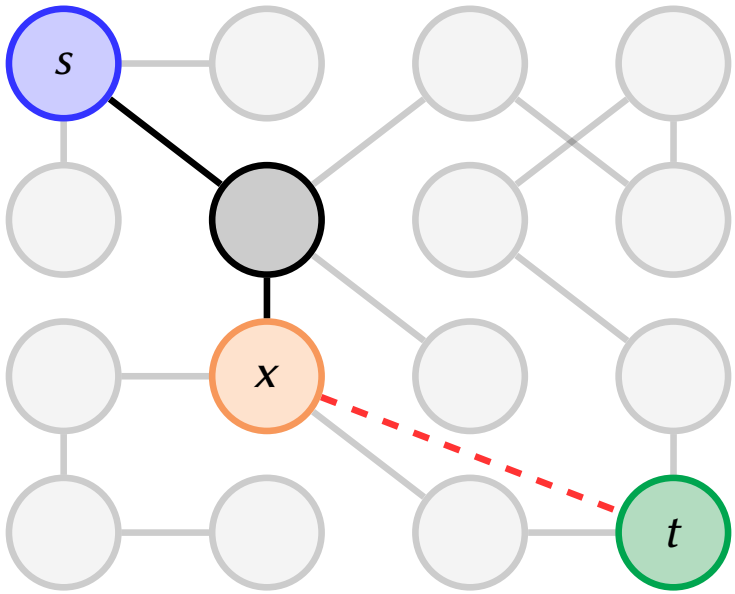


Heuristic Function

$$f(x) = g(x) + h(x)$$

- $g(x)$: distance from the starting node s to node x;
- $h(x)$: *estimated* distance from node x to the goal node t

Heuristic Function

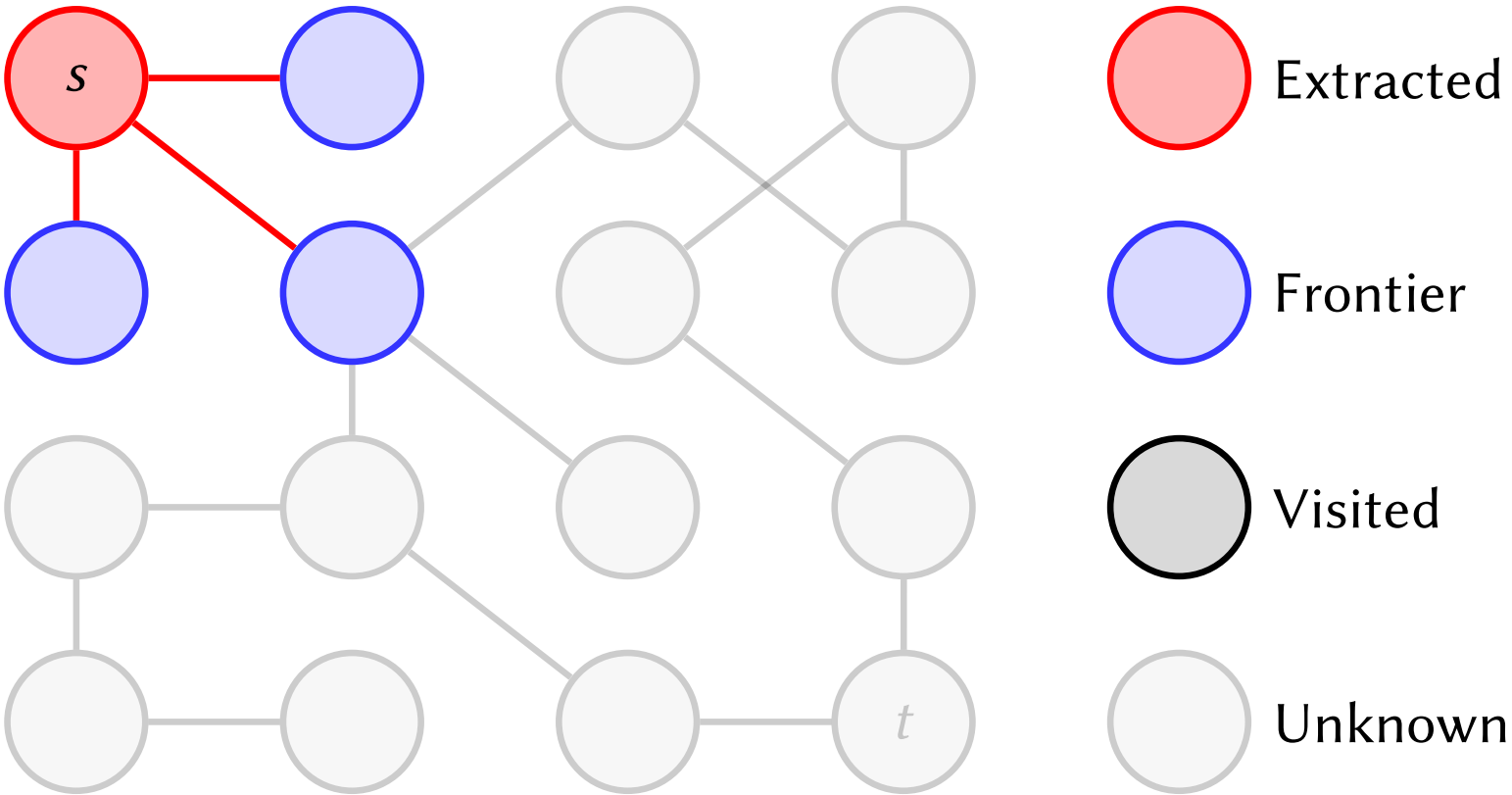


Heuristic Function

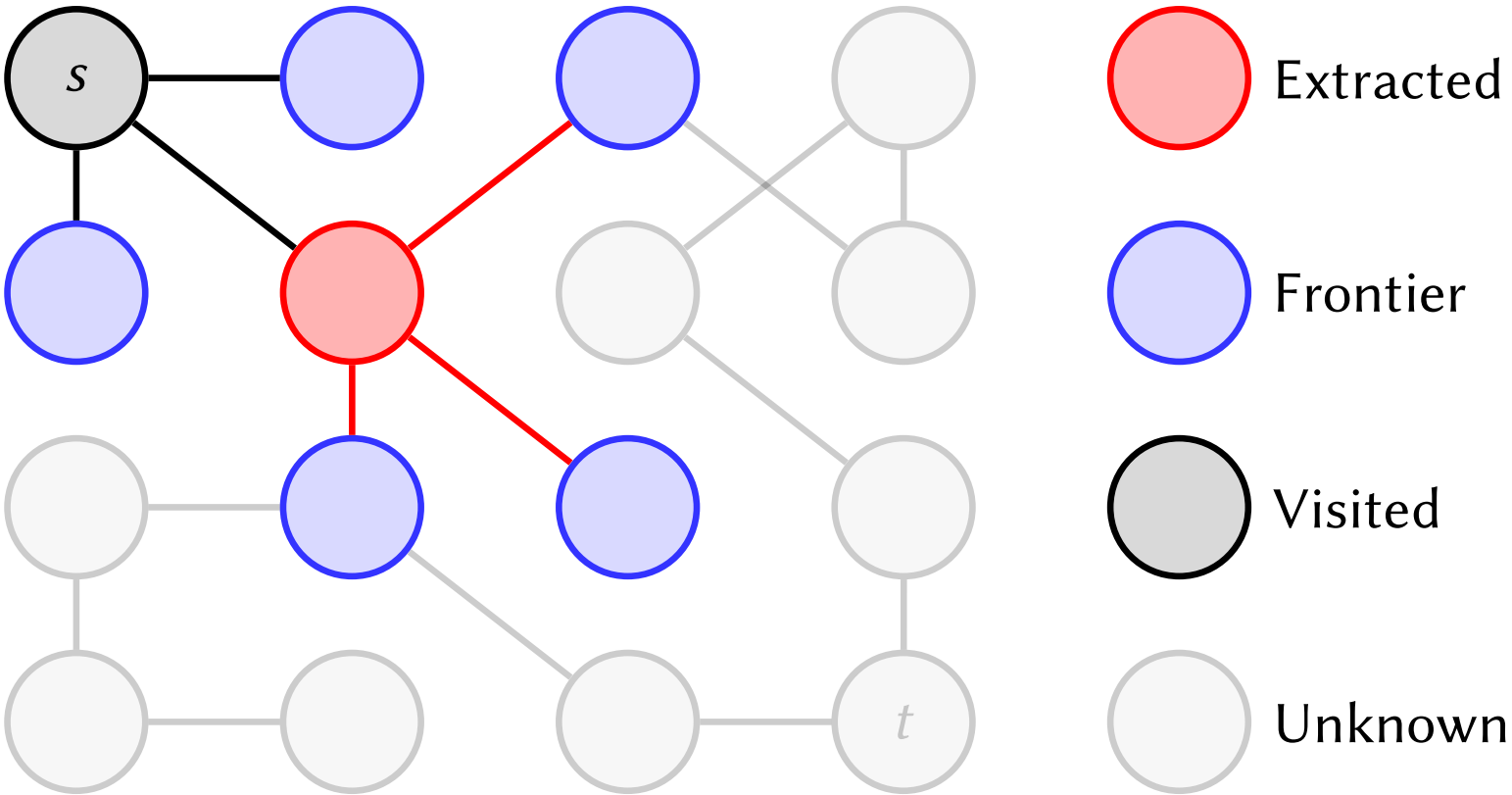
$$f(x) = g(x) + h(x)$$

- $g(x)$: distance from the starting node s to node x;
- $h(x)$: *estimated* distance from node x to the goal node t

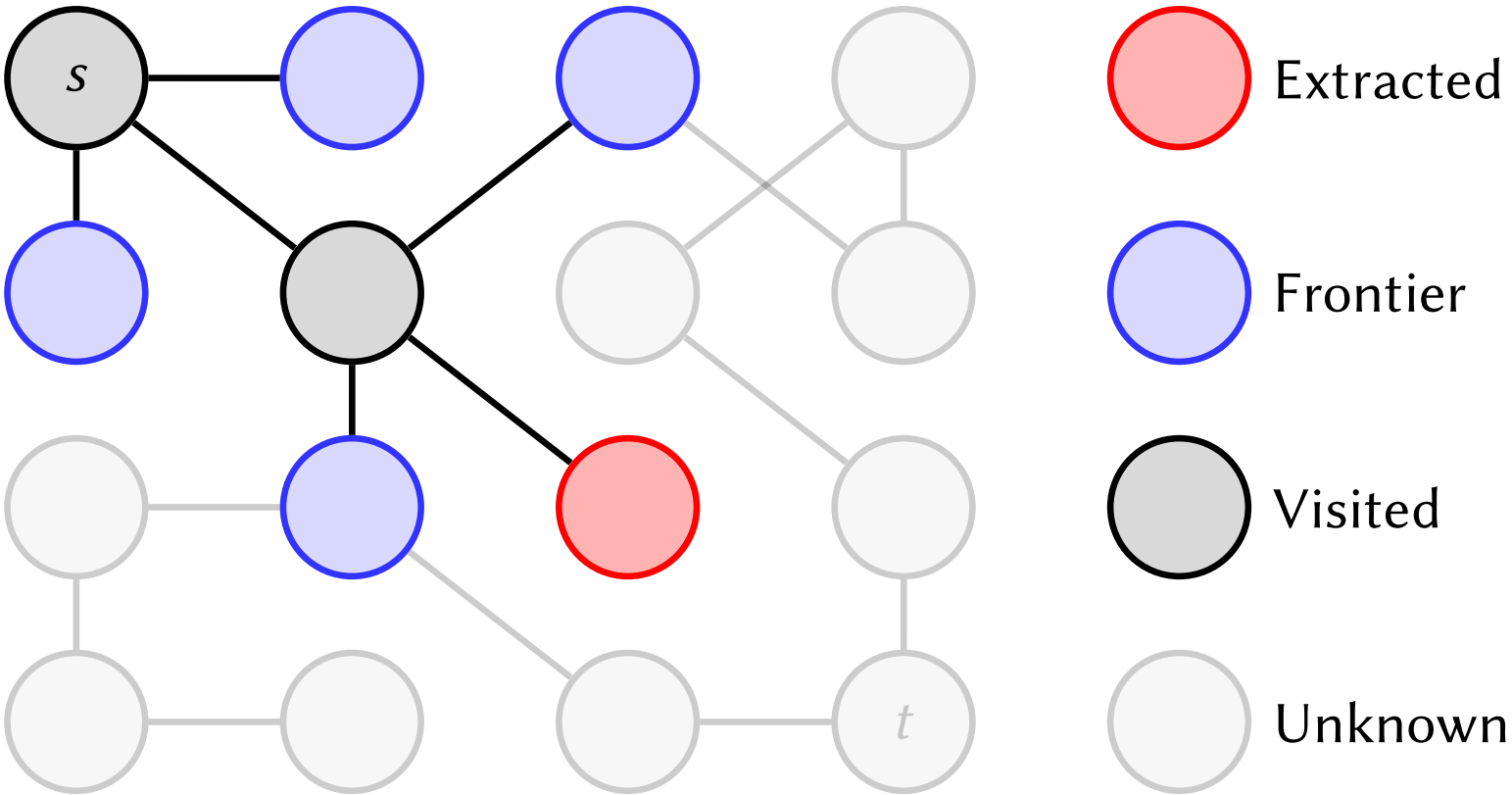
A* Search Example



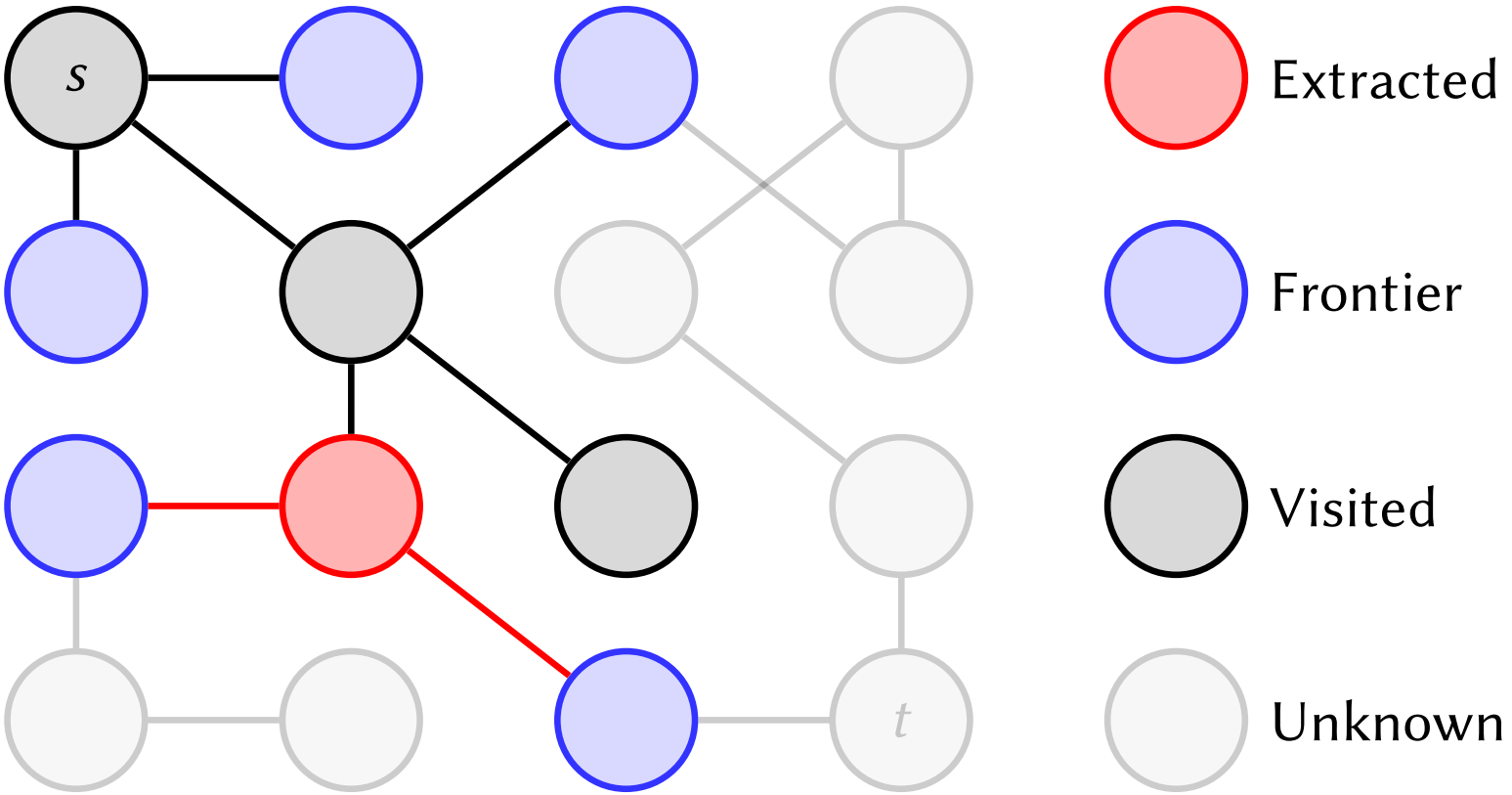
A* Search Example



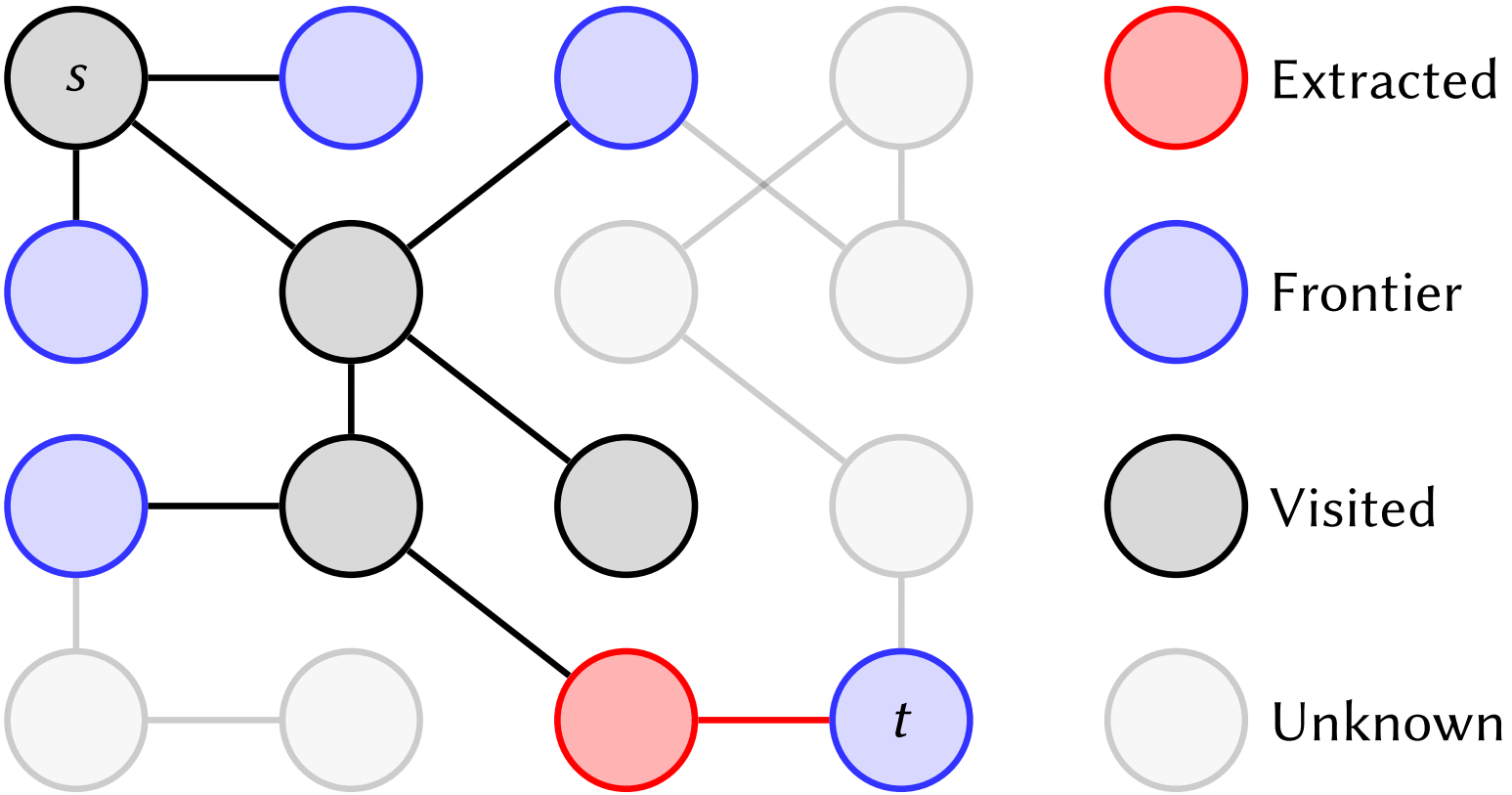
A* Search Example



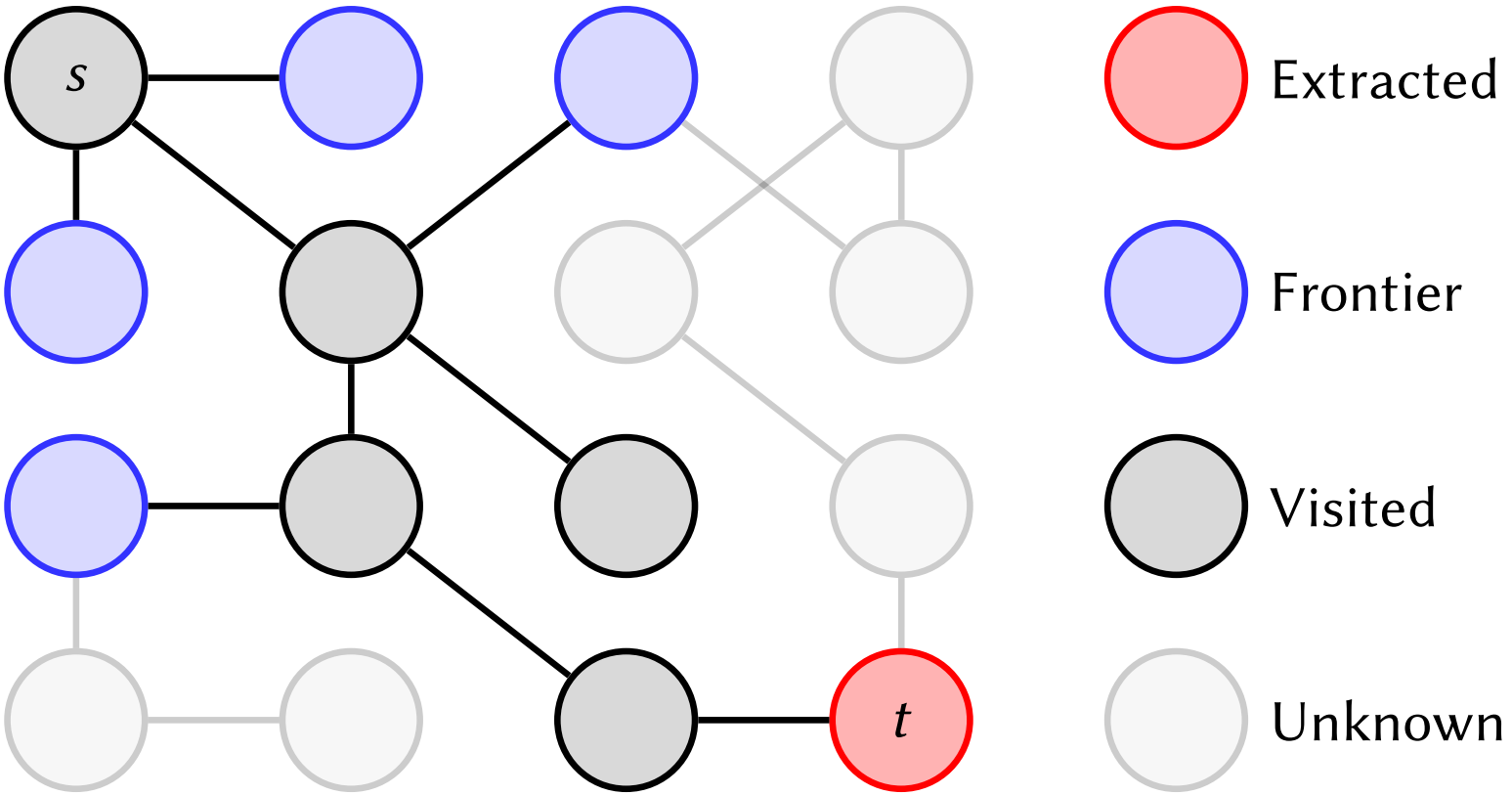
A* Search Example



A* Search Example



A* Search Example



Related Work (Parallelization of A* Search)

- Solve multiple small A* search problems simultaneously on a GPU (Bleiweiss 2008)
- Parallelize A* on a CPU cluster (Kishimoto, Fukunaga, and Botea 2013)
- Node expansion on a GPU for Dijkstra (Sulewski, Edelkamp, and Kissmann 2011)
- None of them makes general A* search work on a GPU!

Related Work (Parallelization of A* Search)

- Solve multiple small A* search problems simultaneously on a GPU (Bleiweiss 2008)
- Parallelize A* on a CPU cluster (Kishimoto, Fukunaga, and Botea 2013)
- Node expansion on a GPU for Dijkstra (Sulewski, Edelkamp, and Kissmann 2011)
- None of them makes general A* search work on a GPU!

Our Contribution

- The first GPU-based A* search framework
- Massively Parallel Algorithm
- "Pure" GPU Algorithm
  - Data structures are stored on the GPU
  - Minimize data transmission overhead
- General A* Search Algorithm
  - Efficient for different problems
  - Guarantee to find the global optimal solution

Our Contribution

- The first GPU-based A* search framework
- Massively Parallel Algorithm
- "Pure" GPU Algorithm
  - Data structures are stored on the GPU
  - Minimize data transmission overhead
- General A* Search Algorithm
  - Efficient for different problems
  - Guarantee to find the global optimal solution

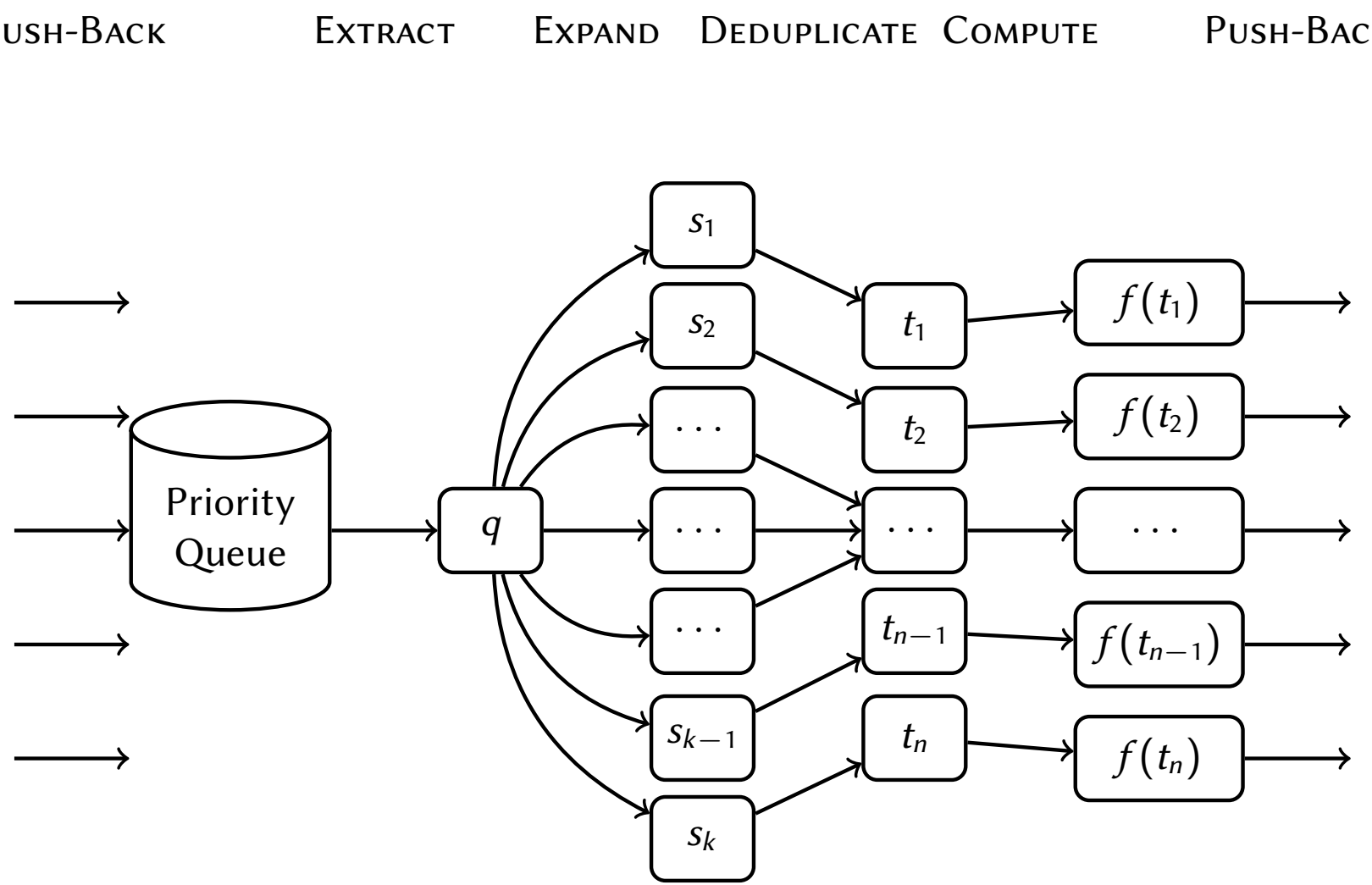
Our Contribution

- The first GPU-based A* search framework
- Massively Parallel Algorithm
- "Pure" GPU Algorithm
  - Data structures are stored on the GPU
  - Minimize data transmission overhead
- General A* Search Algorithm
  - Efficient for different problems
  - Guarantee to find the global optimal solution

Our Contribution

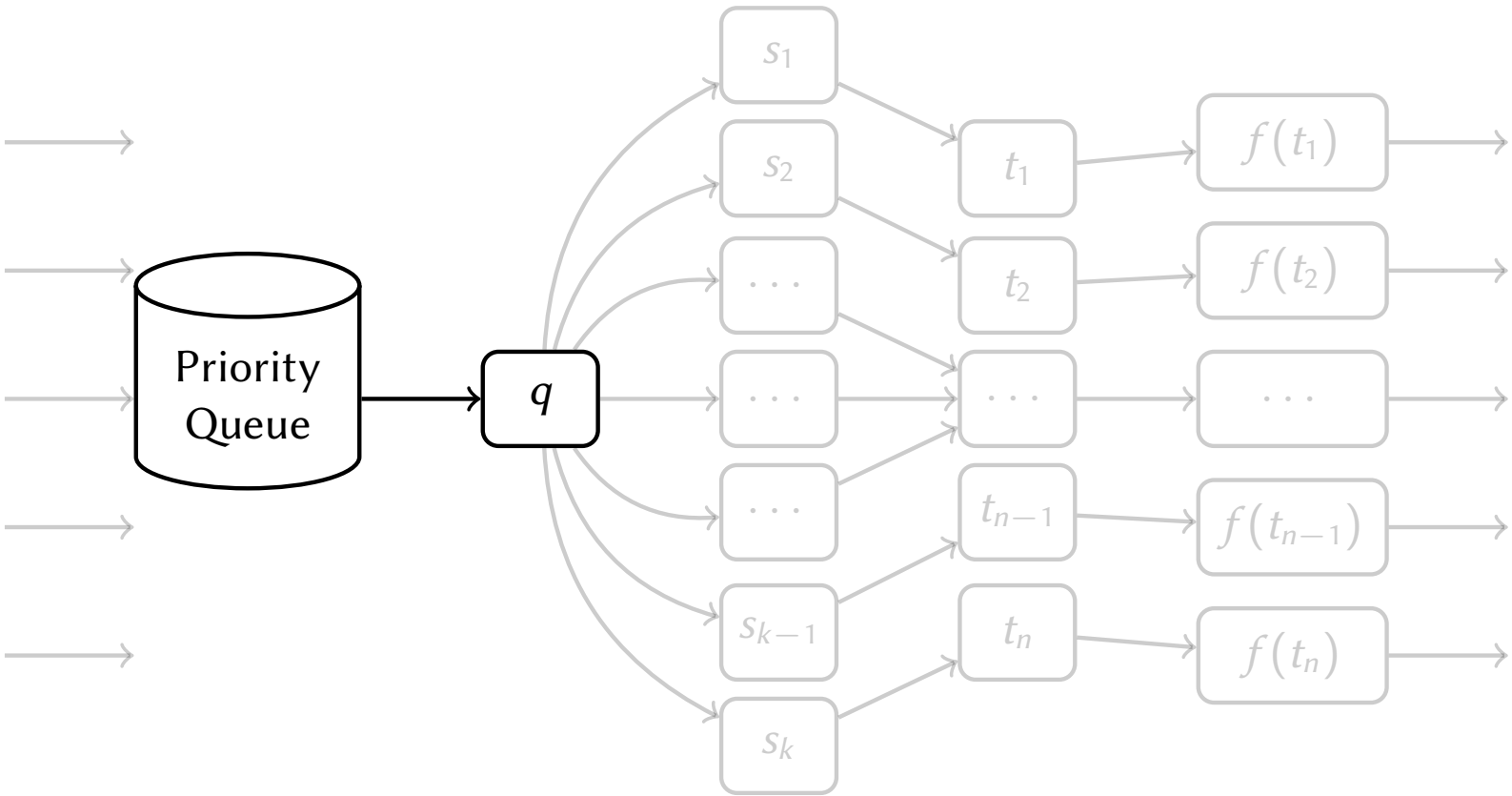
- The first GPU-based A* search framework
- Massively Parallel Algorithm
- "Pure" GPU Algorithm
  - Data structures are stored on the GPU
  - Minimize data transmission overhead
- General A* Search Algorithm
  - Efficient for different problems
  - Guarantee to find the global optimal solution

Workflow of Traditional A* Algorithm



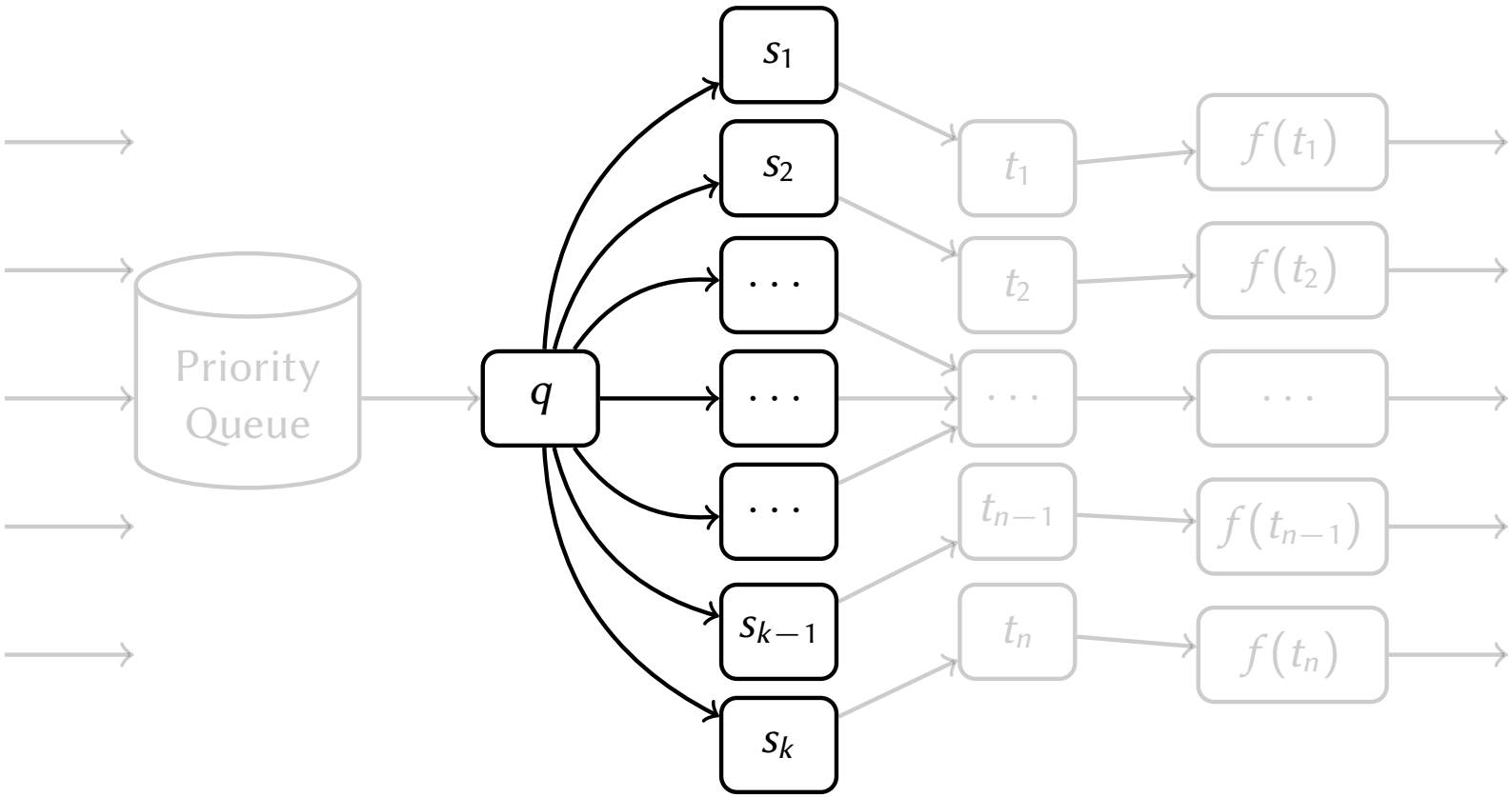
Workflow of Traditional A* Algorithm

Push-Back | Extract | Expand | Deduplicate | Compute | Push-Back



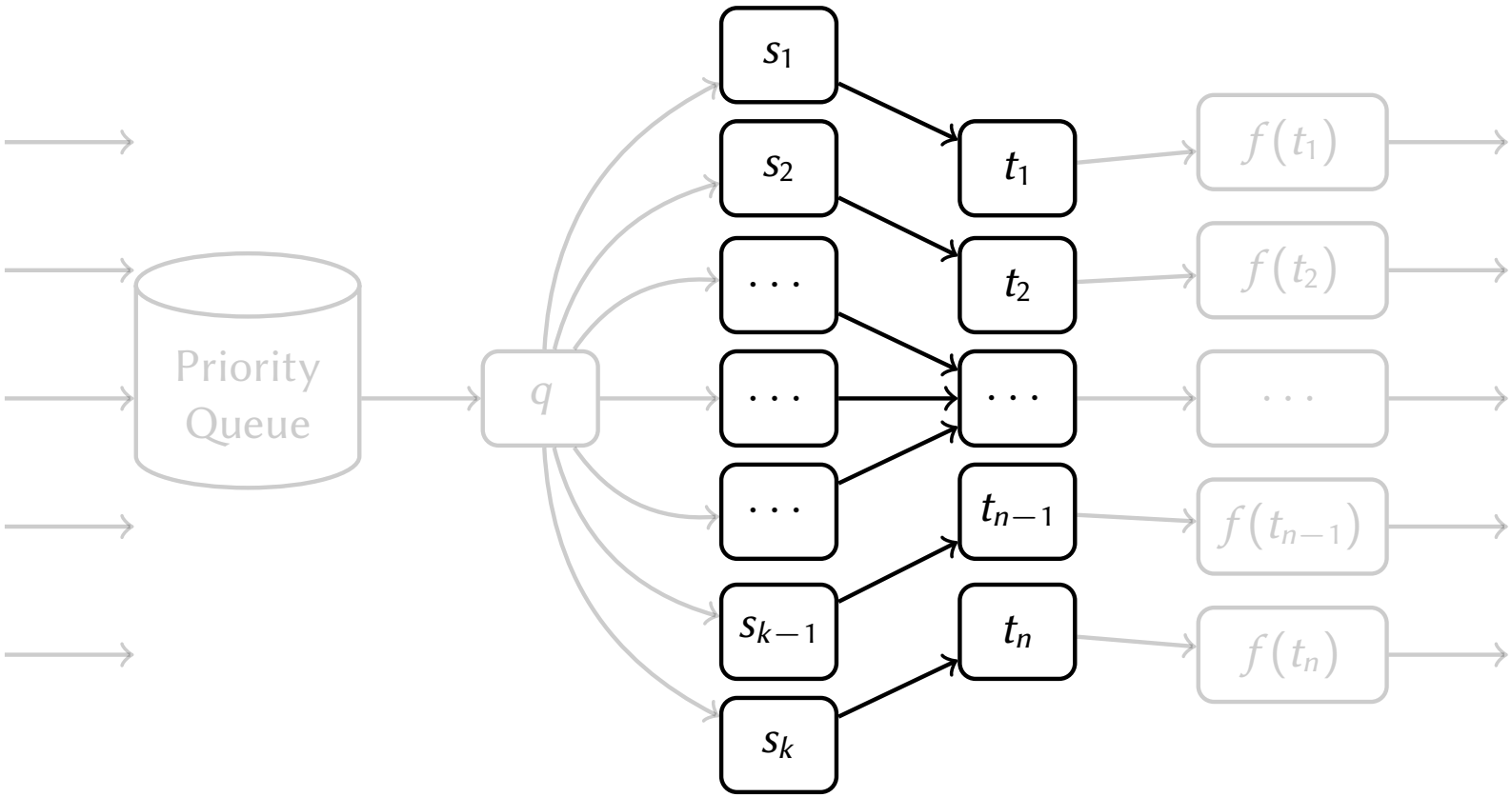
Workflow of Traditional A* Algorithm

Push-Back | Extract | **Expand** | Deduplicate | Compute | Push-Back



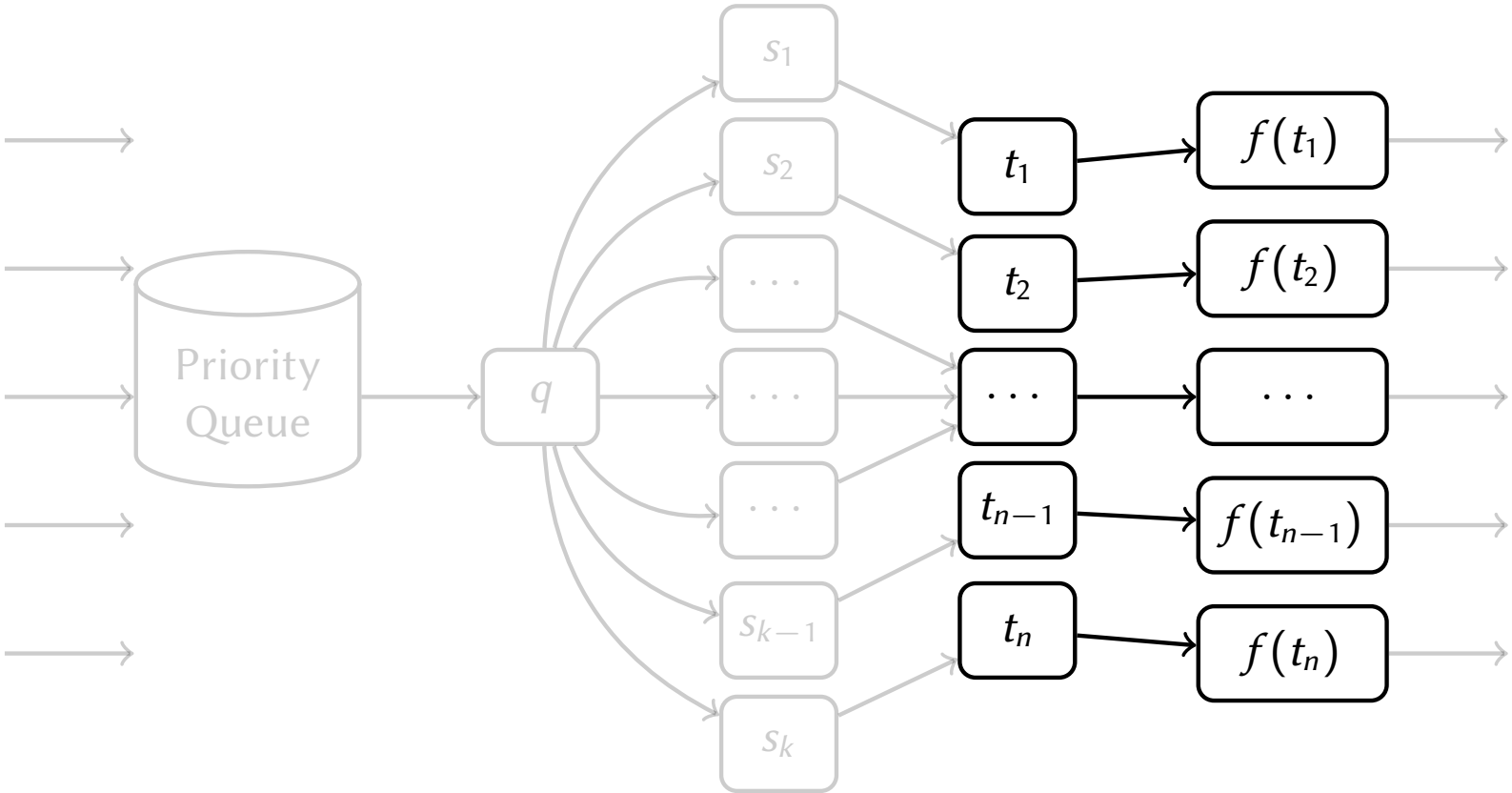
Workflow of Traditional A* Algorithm

Push-Back Extract Expand **Deduplicate** Compute Push-Back



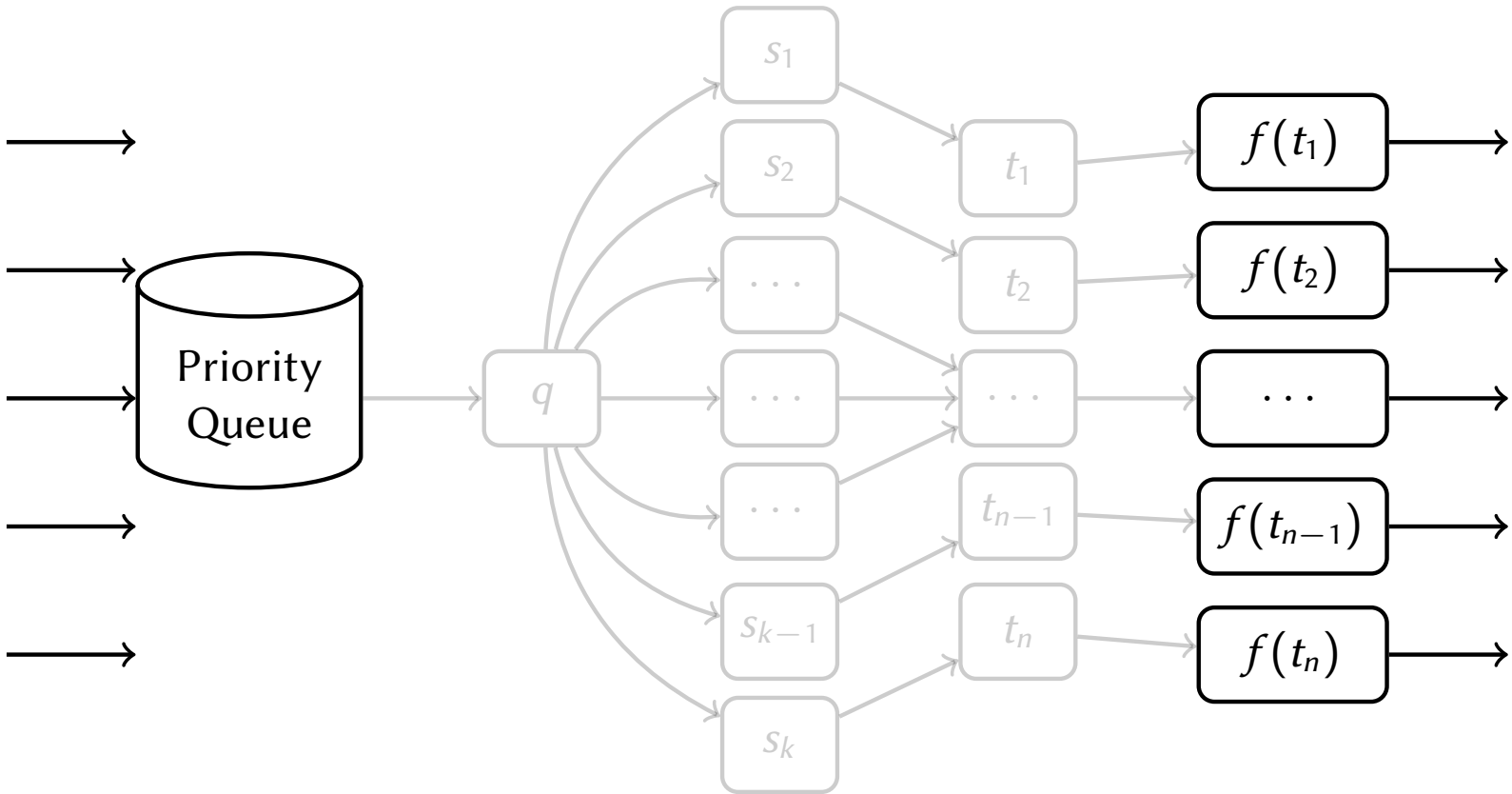
Workflow of Traditional A* Algorithm

Push-Back Extract Expand Deduplicate **Compute** Push-Back



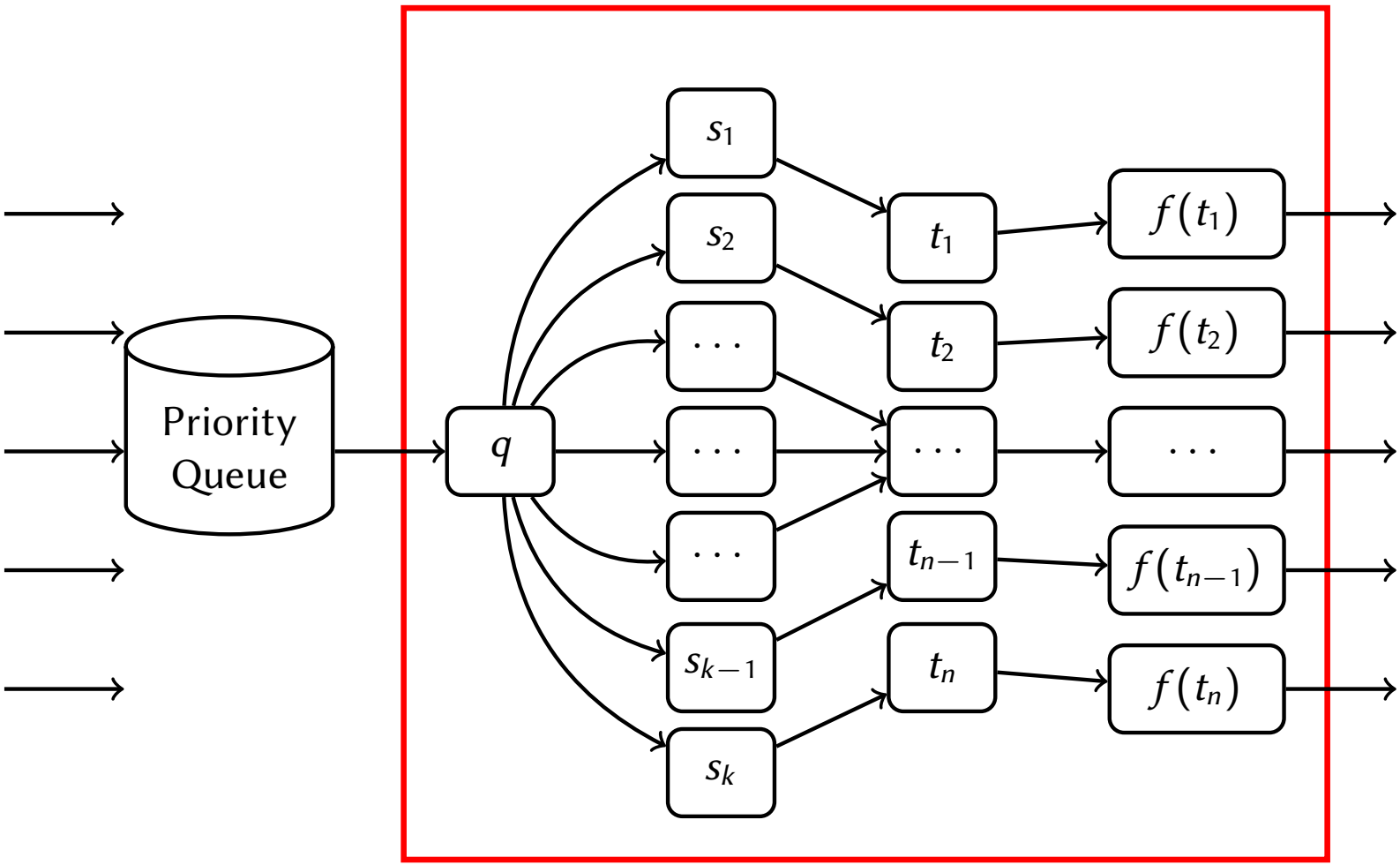
Workflow of Traditional A* Algorithm

Push-Back | Extract | Expand | Deduplicate | Compute | Push-Back



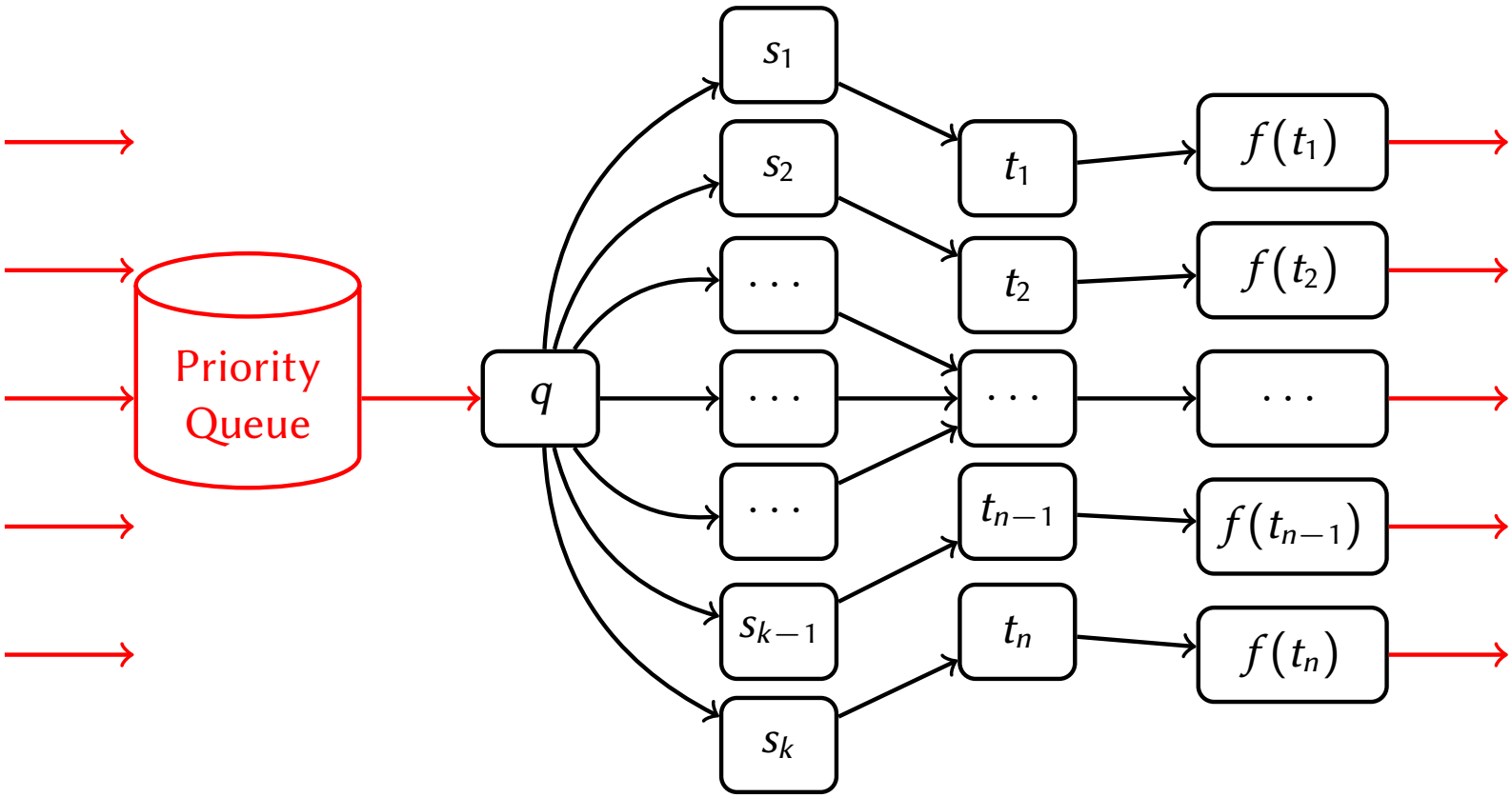
Workflow of Traditional A* Algorithm

Push-Back Extract Expand Deduplicate Compute Push-Back

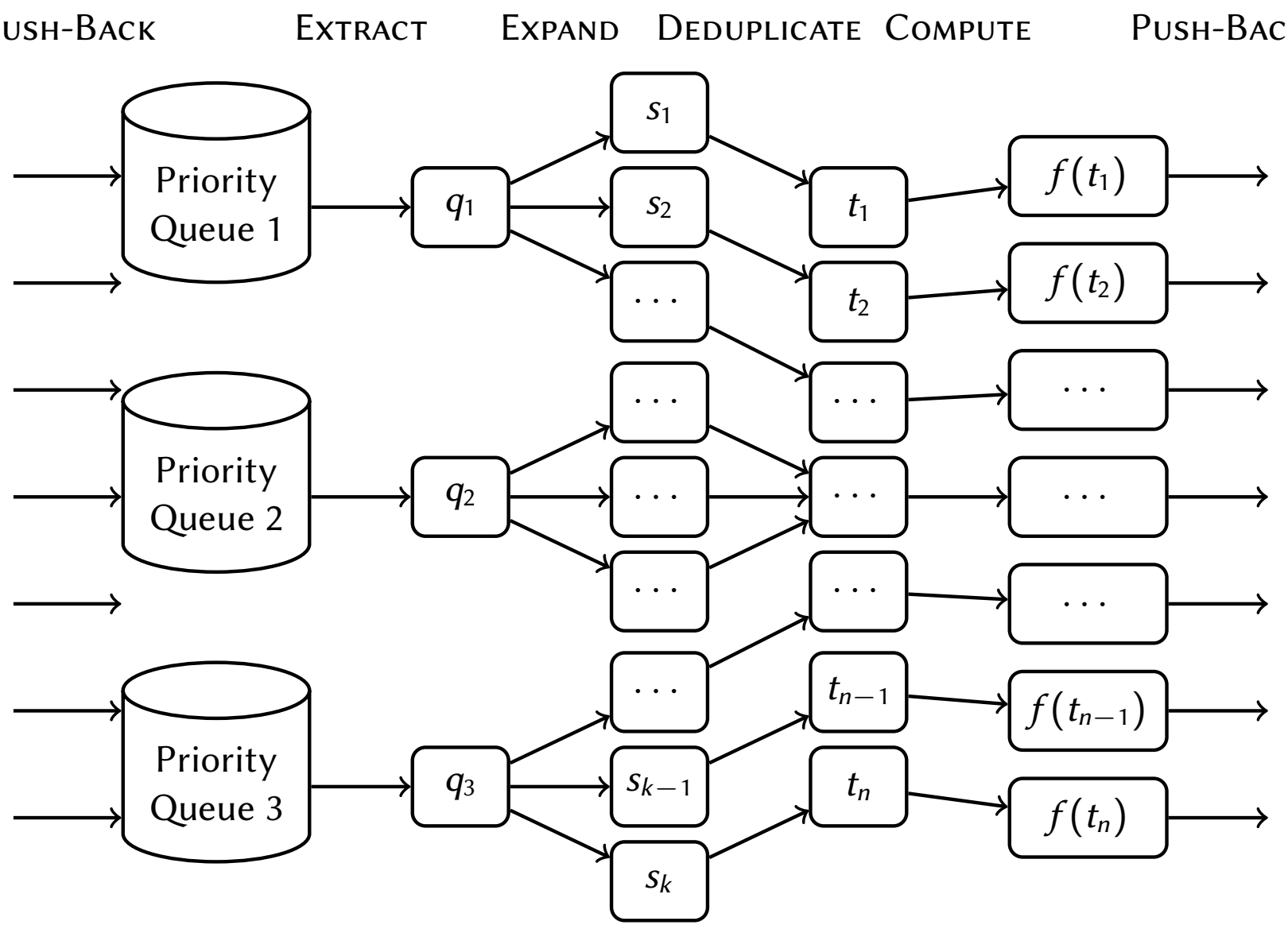


Workflow of Traditional A* Algorithm

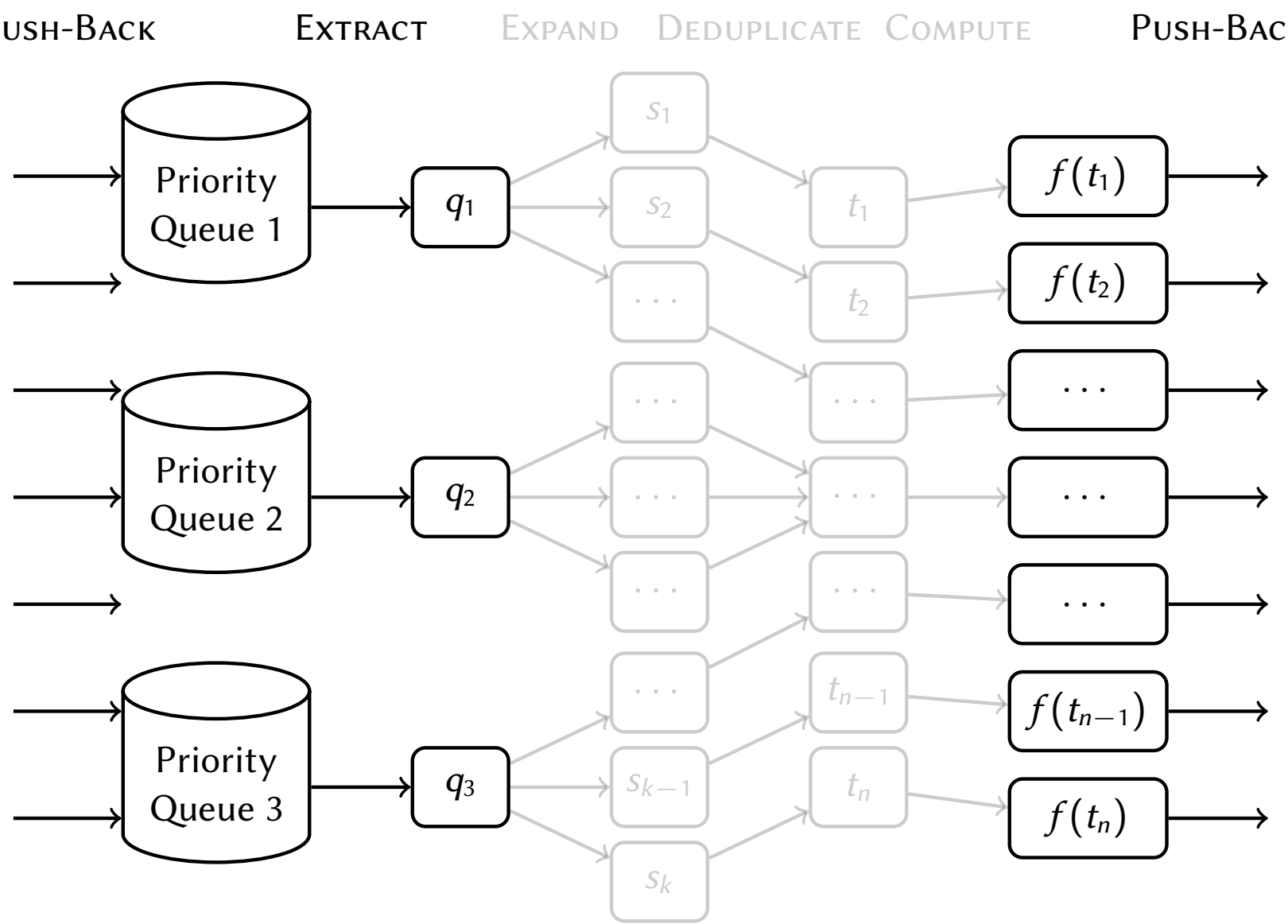
Push-Back Extract Expand Deduplicate Compute Push-Back



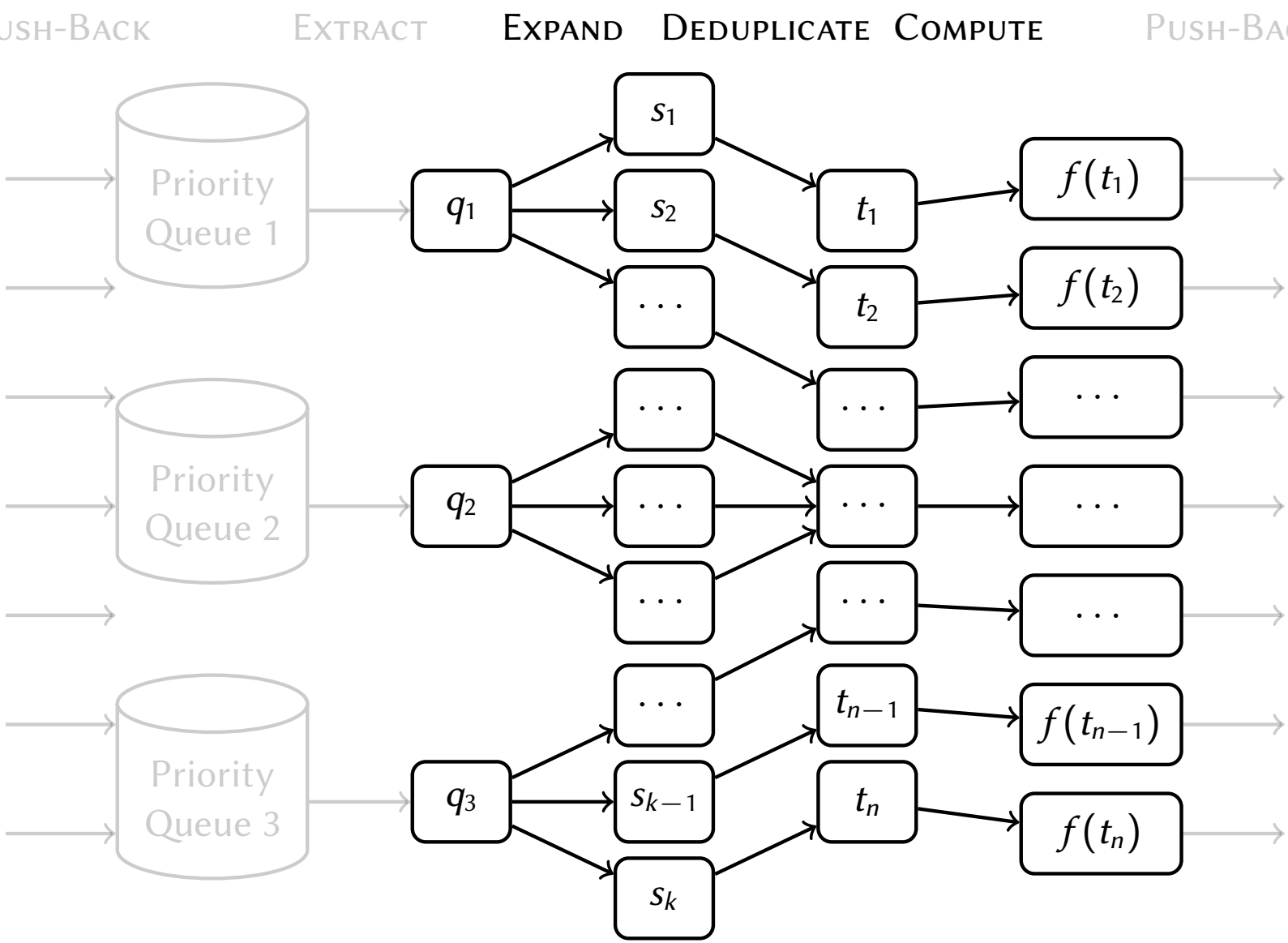
Workflow of GPU-based A* Algorithm



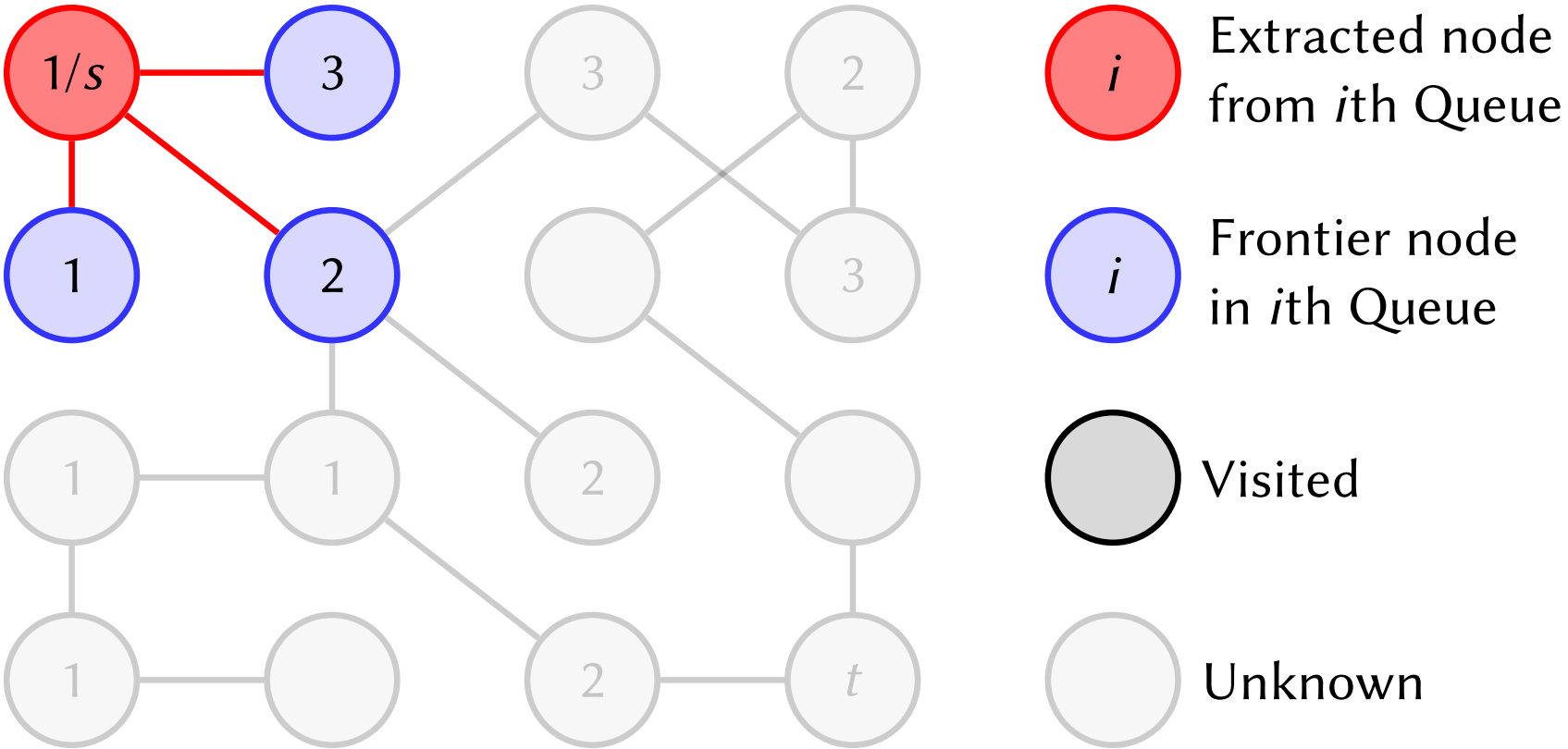
Workflow of GPU-based A* Algorithm



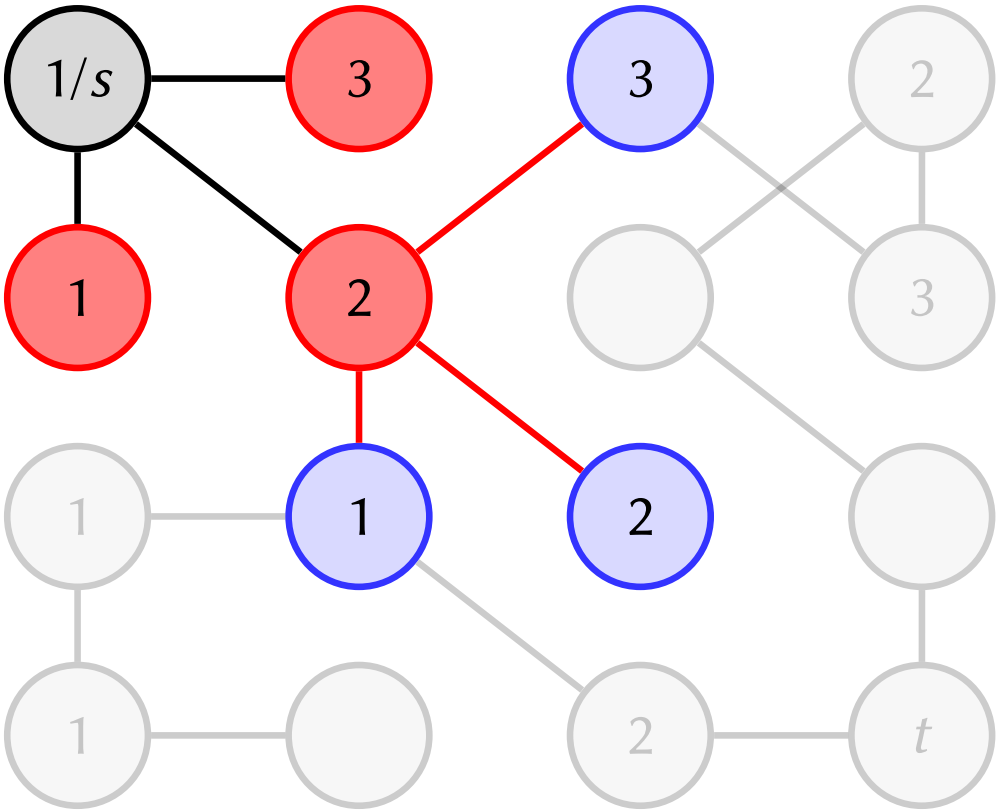
Workflow of GPU-based A* Algorithm

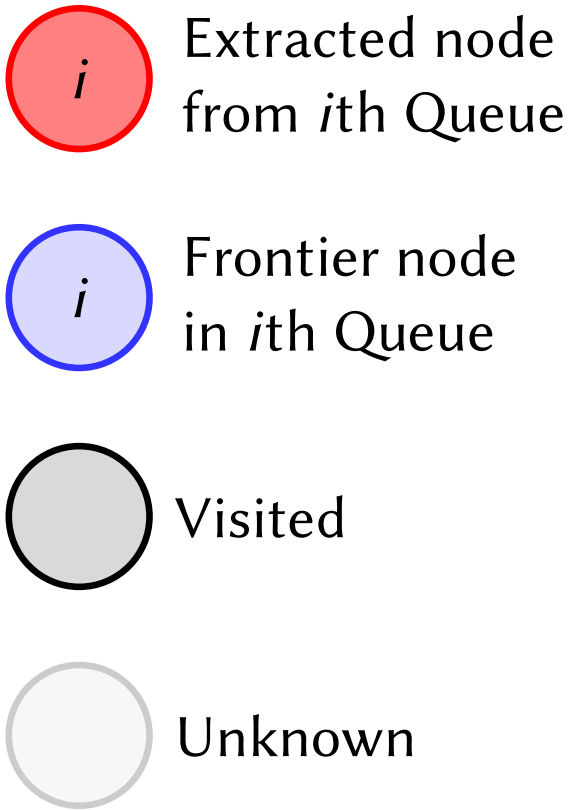


GPU-based A* Search Example Using 3 Priority Queues

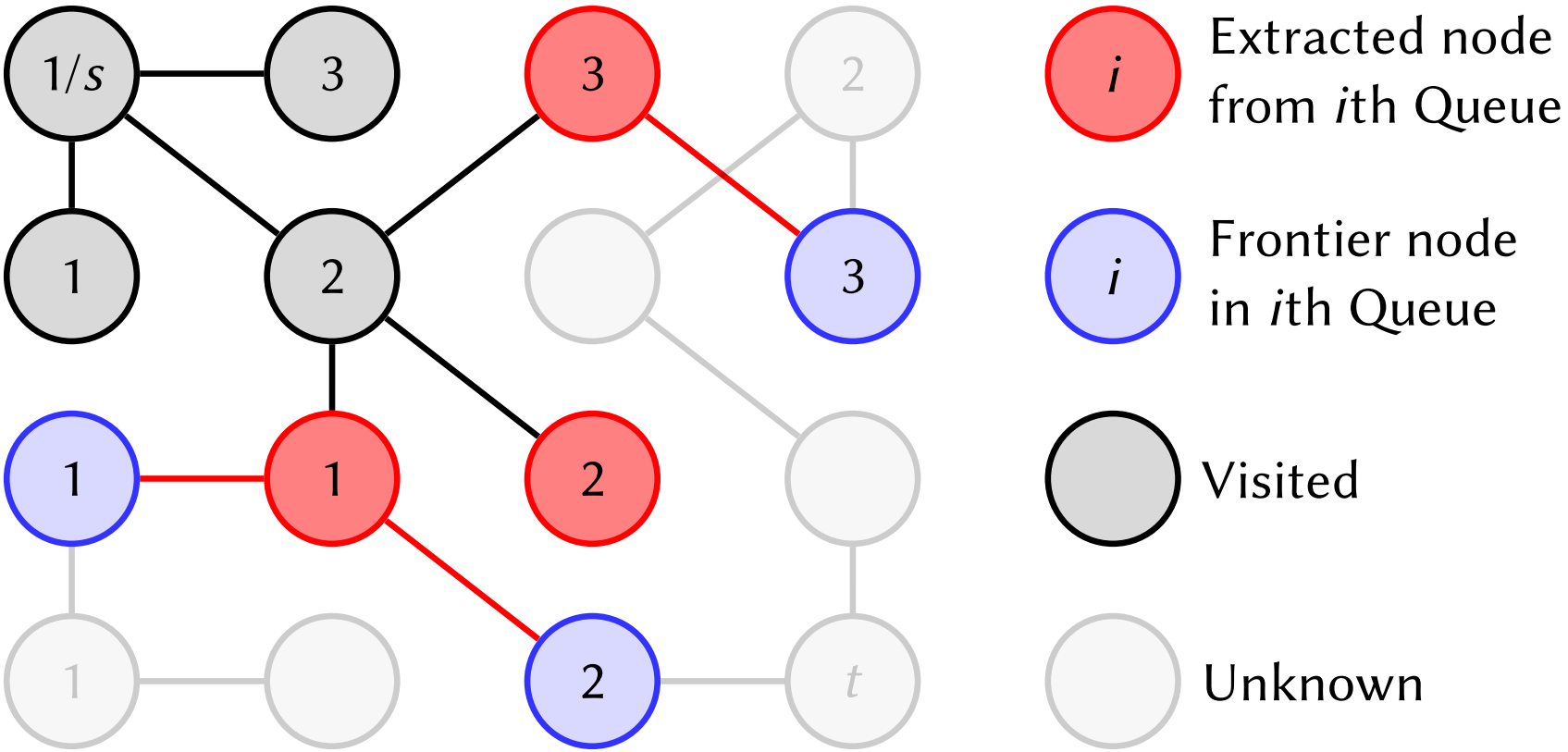


GPU-based A* Search Example Using 3 Priority Queues

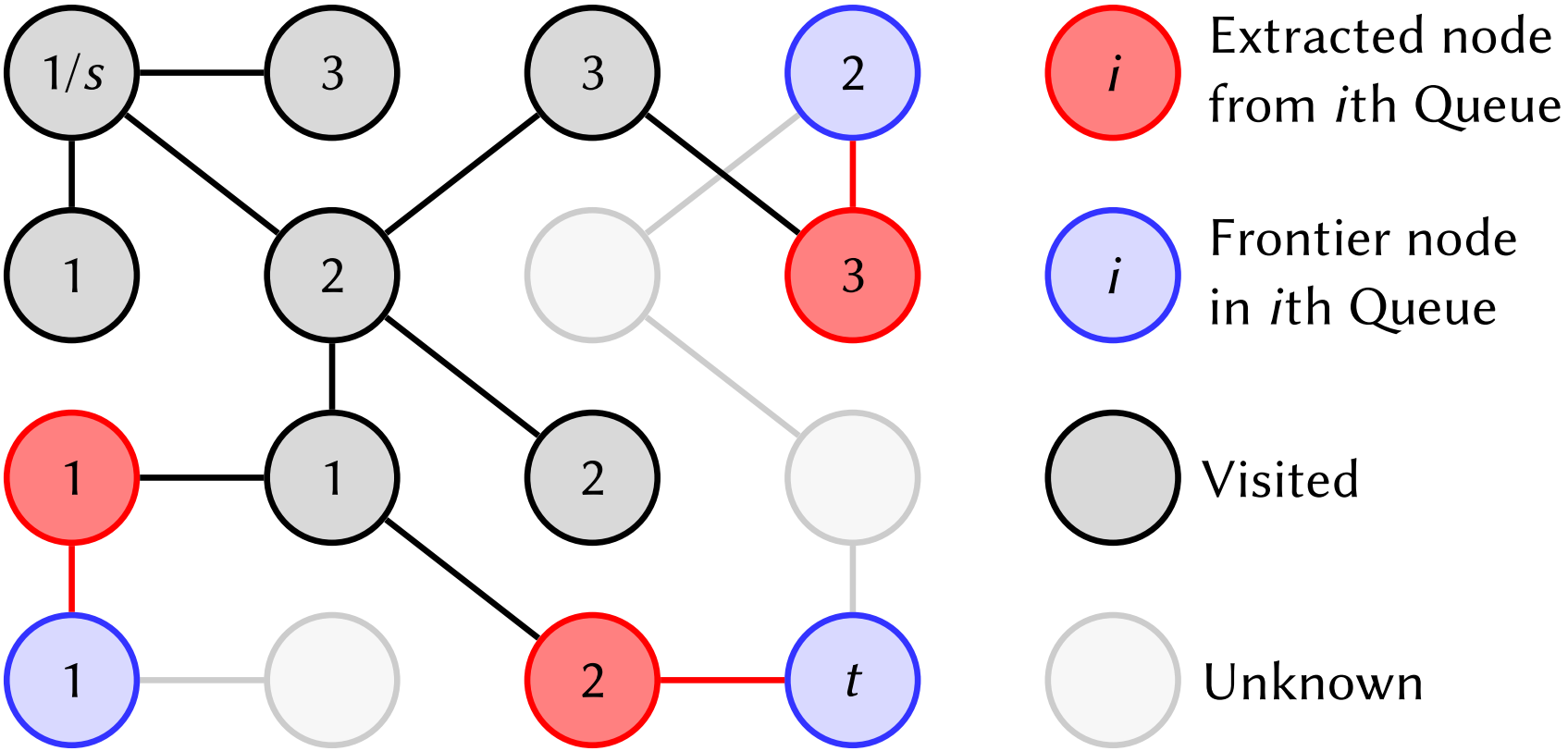




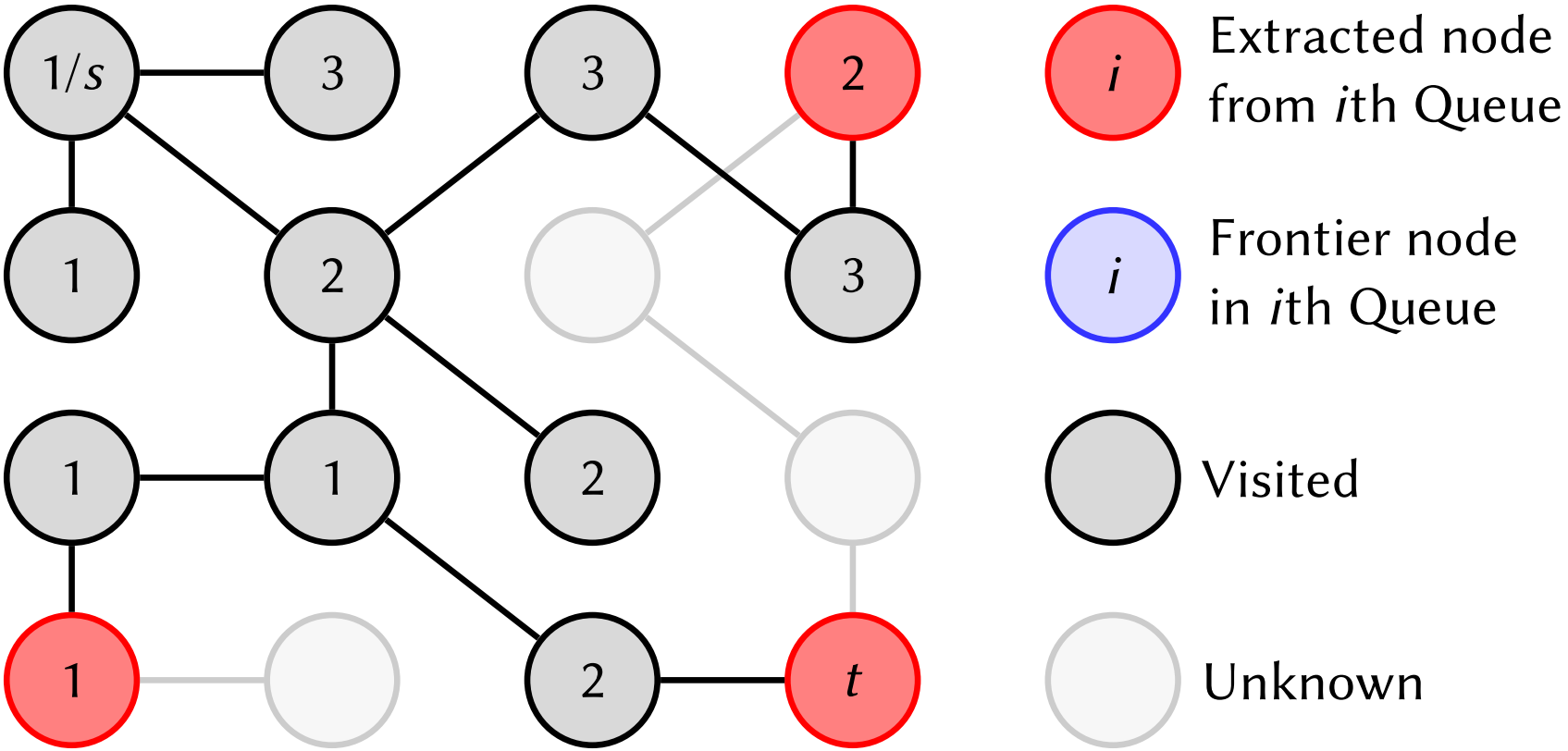
GPU-based A* Search Example Using 3 Priority Queues



GPU-based A* Search Example Using 3 Priority Queues



GPU-based A* Search Example Using 3 Priority Queues



Evaluation

Experiment

- Evaluate running time and memory usage of
  - traditional single-thread CPU-based A* search
  - our GPU-based A* search
- On three problems with different characteristics

Environment Information

CPU: Intel Xeon™ E5-1620 3.6GHz

GPU: NVIDIA Tesla K20C (2496 logic cores)

Evaluation

Experiment

- Evaluate running time and memory usage of
  - traditional single-thread CPU-based A* search
  - our GPU-based A* search
- On three problems with different characteristics

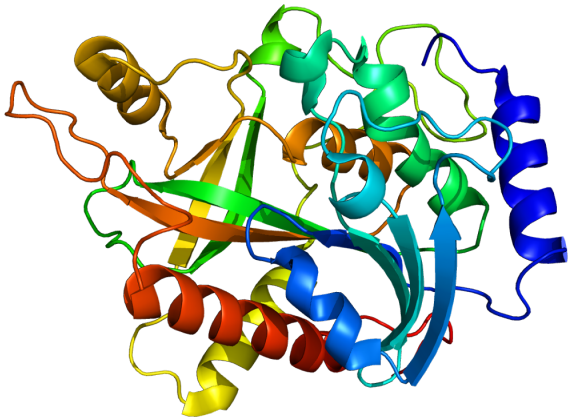
Environment Information

CPU: Intel Xeon™ E5-1620 3.6GHz

GPU: NVIDIA Tesla K20C (2496 logic cores)

Structure-Based Protein Design

Problem Description



Protein structure

Optimization Algorithm

· · ·	Ile	Ser	Ile
Pro	Ala	Ile	Trp
Val	Gly	Ser	Asp
Gly	Gly	Pro	Glu
Ile	Pro	His	· · ·

1D amino acid sequence

Energy Function (Optimization Target)

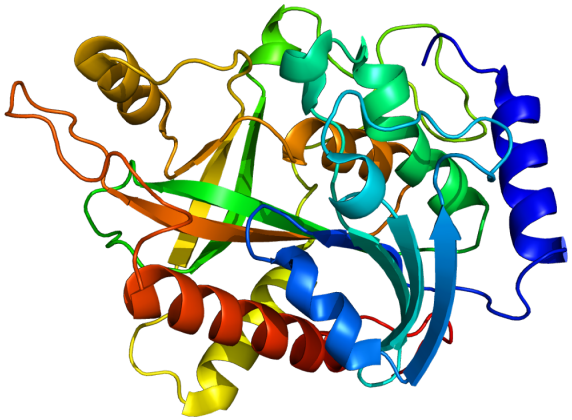
$$E_T = \sum_{i_r} E_1(i_r) + \sum_{i_r} \sum_{j_s, i<j} E_2(i_r, j_s)$$

Features

- Heuristic function is computationally expensive
- Tree search rather than graph search

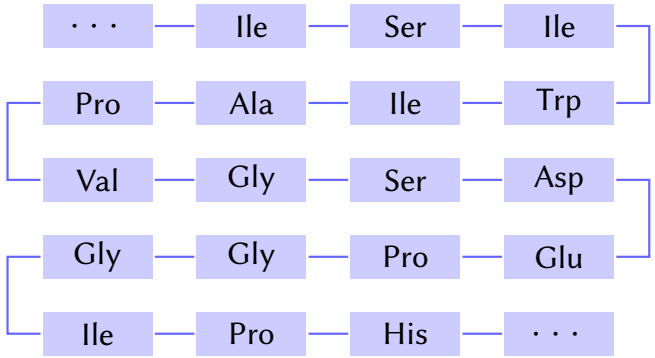
Structure-Based Protein Design

Problem Description



Protein structure

Optimization Algorithm



1D amino acid sequence

Energy Function (Optimization Target)

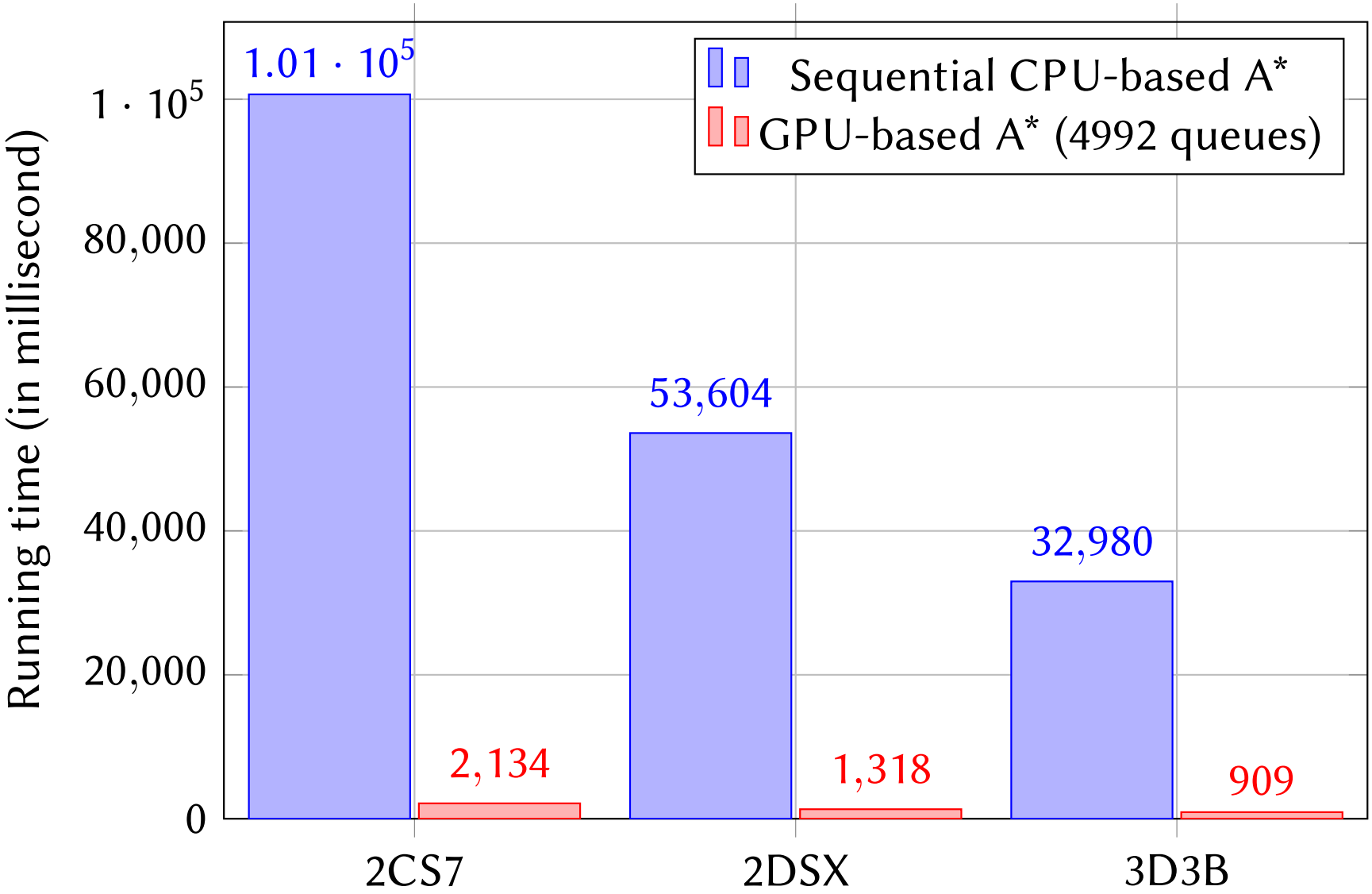
$$E_T = \sum_{i_r} E_1(i_r) + \sum_{i_r} \sum_{j_s, i<j} E_2(i_r, j_s)$$

Features

- Heuristic function is computationally expensive
- Tree search rather than graph search

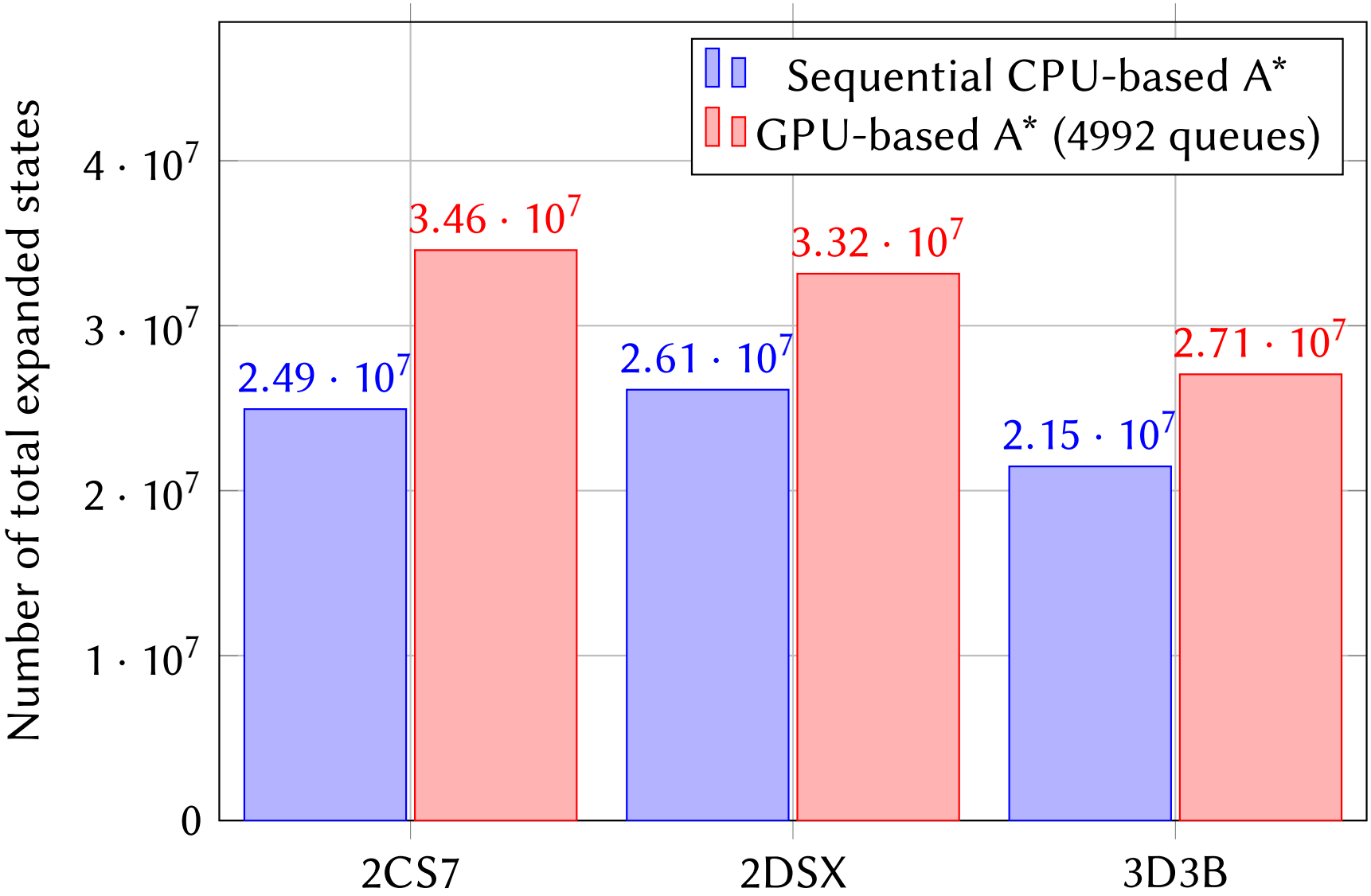
Structure-Based Protein Design

Experiment Result — Running Time

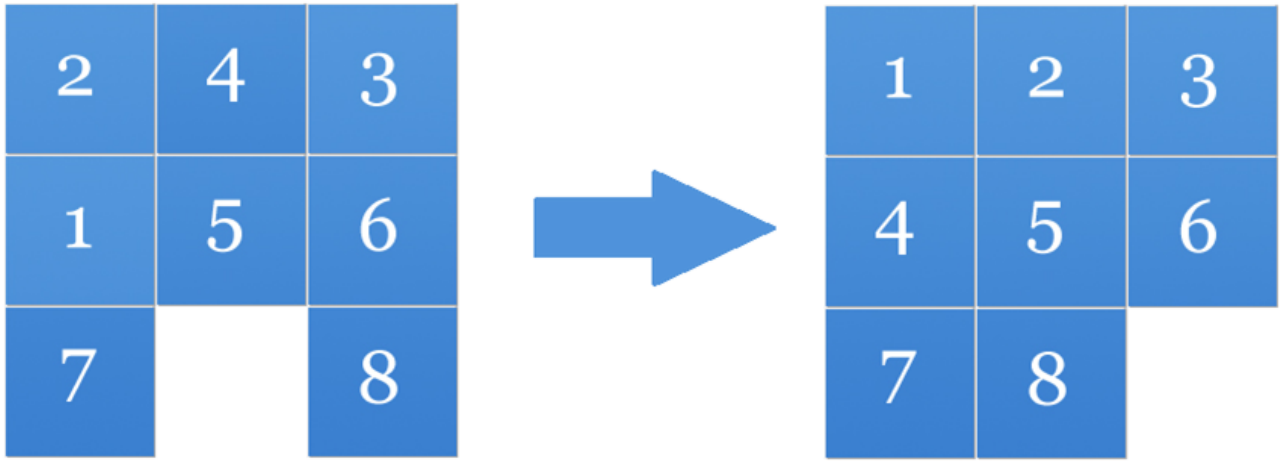


Structure-Based Protein Design

Experiment Result — Number of Total Expanded States



Sliding Puzzle
Problem Description

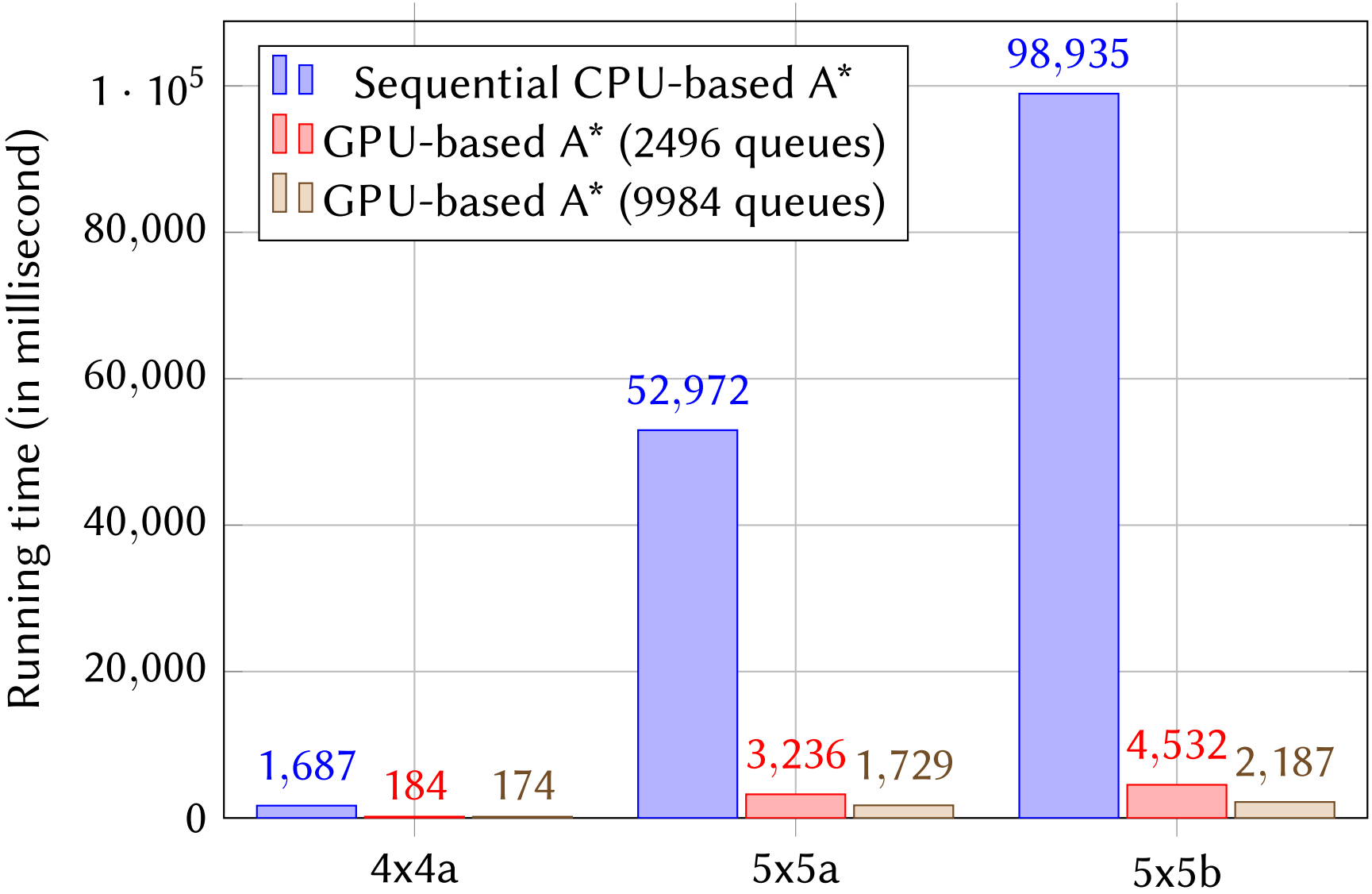


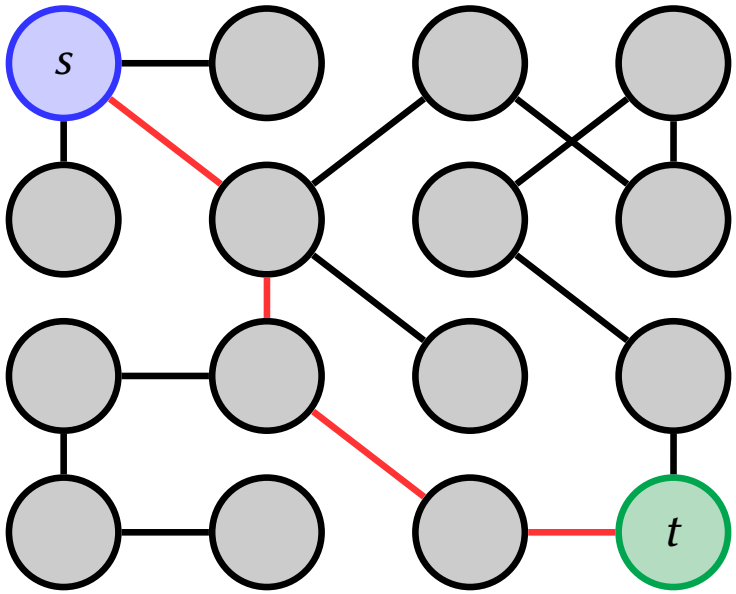
Heuristic Function

- Using *disjoint pattern database* heuristic (Korf and Felner 2002)
- Heuristic function requires memory access

Sliding Puzzle

Experiment Result — Running Time



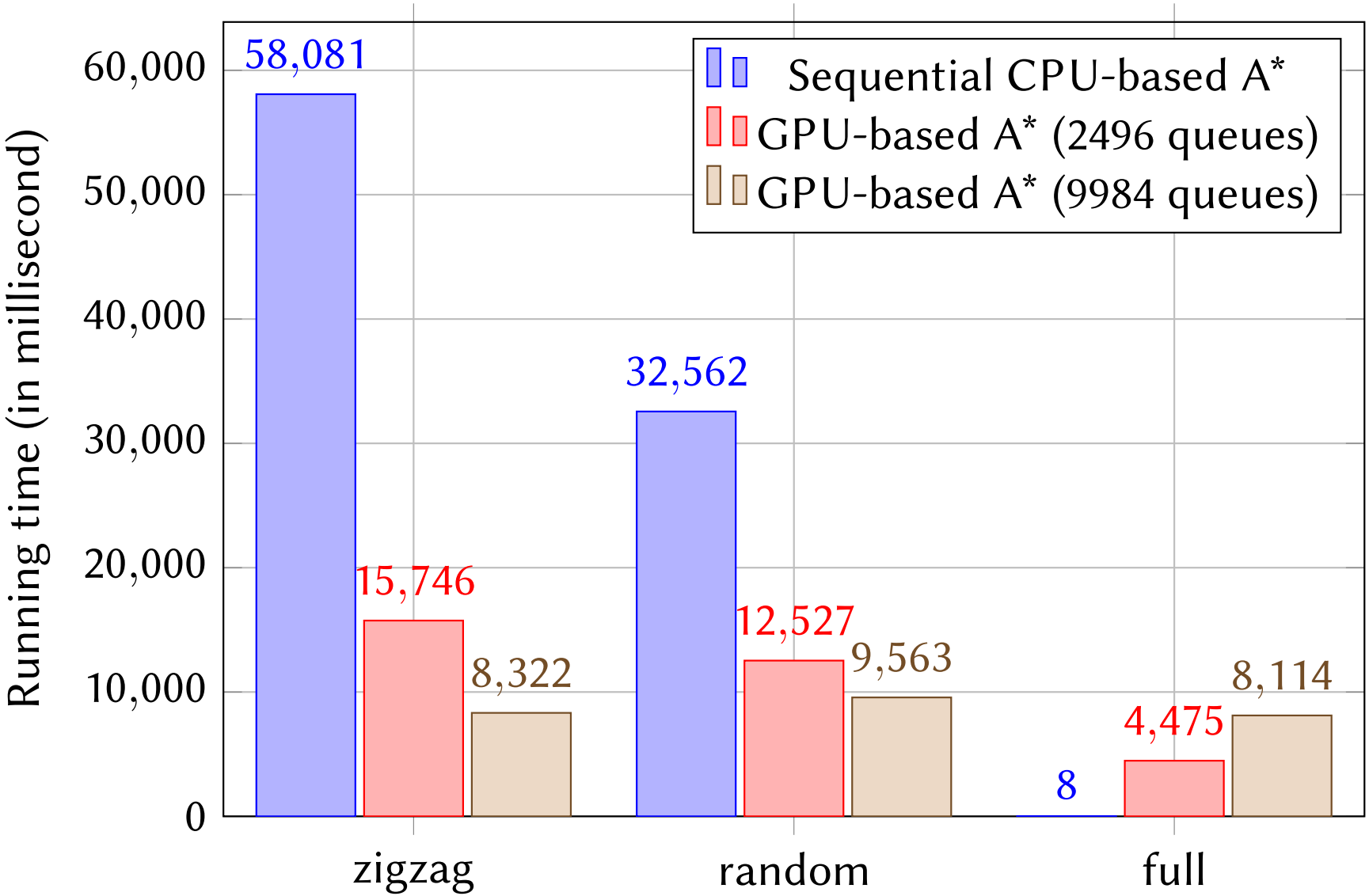


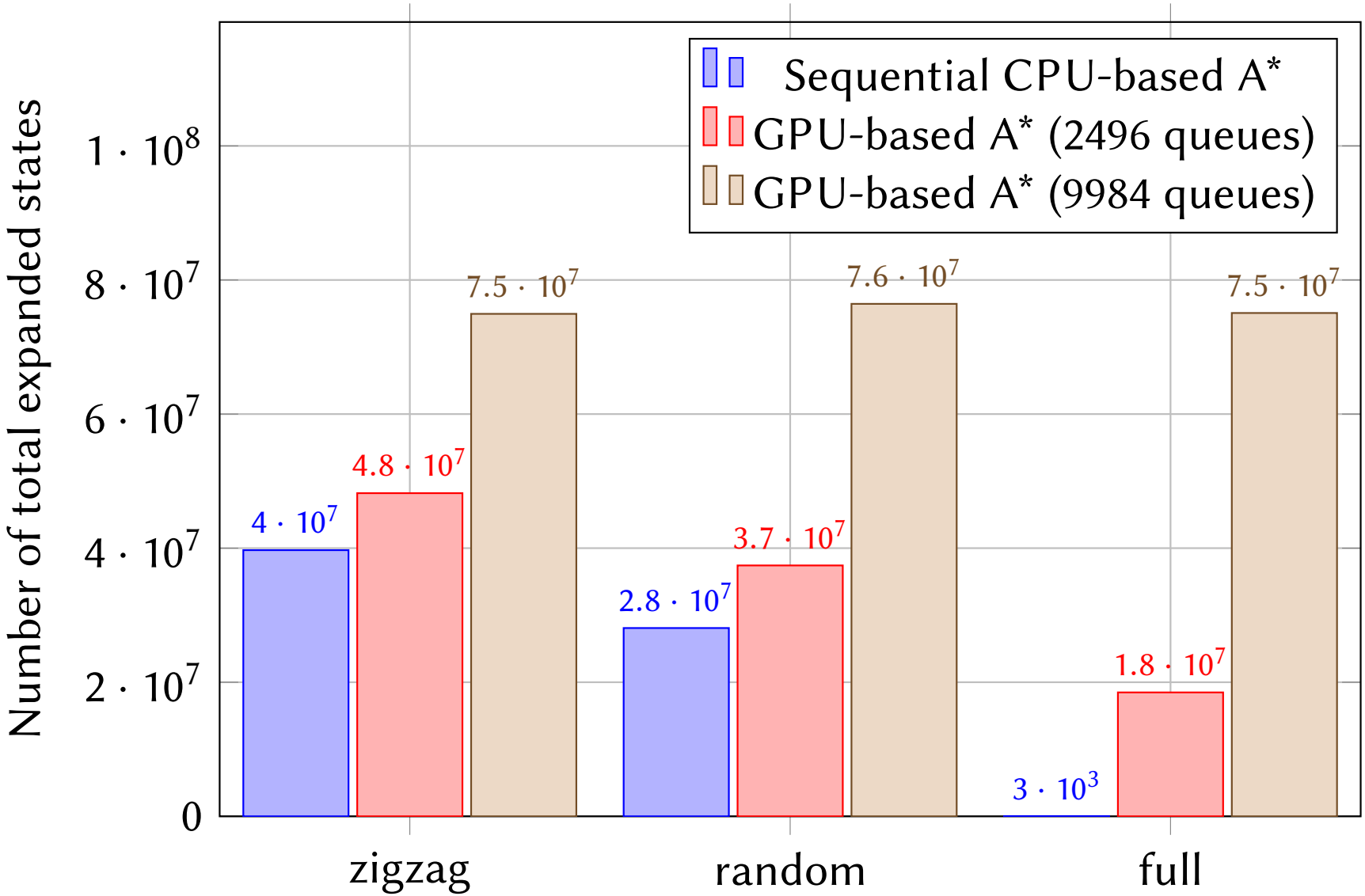
Features

- Grid network size: $10,000 \times 10,000$
- Simple heuristic function: diagonal distance

Shortest Path

Experiment Result — Running Time




Number of total expanded states
1 · 10^8
8 · 10^7
6 · 10^7
4 · 10^7
2 · 10^7
0
Sequential CPU-based A*
GPU-based A* (2496 queues)
GPU-based A* (9984 queues)
4 · 10^7
4.8 · 10^7
7.5 · 10^7
2.8 · 10^7
3.7 · 10^7
7.6 · 10^7
3 · 10^3
1.8 · 10^7
7.5 · 10^7
zigzag
random
full

Conclusion

Our Contribution

- First massively parallel GPU-based A*
- 4x - 45x speedup for most problems

Future Work

- Improvement when the parallelism of problems is limited
- Multi-GPUs A* search
- Extend to other heuristic search

Thank you!

Funding

- National Basic Research Program of China
- National Natural Science Foundation of China