Time Optimal Reachability Analysis using Swarm Verification

Zhengkui Zhang, Brian Nielsen, Kim G. Larsen

Department of Computer Science, Aalborg University

{zhzhang,bnielsen,kgl}@cs.aau.dk

Contents

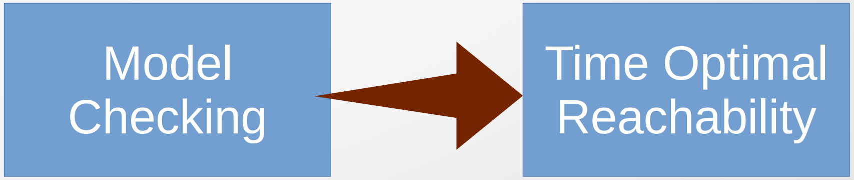
- Motivation
  - Time optimal reachability
  - Swarm verification
- Sequential Time Optimal Reachability
- Swarm Time Optimal Reachability
- Experiments
- Conclusion
- Future Work

Motivation – Time optimal reachability

- Methods for scheduling and planning problems:
  - Numerical: linear / dynamic programing, operation research, etc
  - Model based: time optimal reachability, timed game, etc
- Tools for model based method: Uppaal, Kronos.
- Advantages of model based method:
  - Model real-time behaviors, constraints in natural way;
  - Flexible choice of underling algorithms efficiently implemented in model checker.

Motivation – Time optimal reachability

- Real time model checking: given a timed model $\mathcal{M}$ and a specification ϕ, verify $\mathcal{M} \models \phi$, generate a diagnostic trace ρ if $\mathcal{M} \not\models \phi$.
- Time optimal reachability: given a timed model $\mathcal{M}$ and a reachability specification ϕ, when $\mathcal{M} \models \phi$ generate a diagnostic trace ρ such that its span (accumulated delay) is minimum.
- Time optimal reachability is real time reachability problem with cost, optimality and pruning into account.



Contents

- Motivation
  - Time optimal reachability
  - Swarm verification
- Sequential Time Optimal Reachability
- Swarm Time Optimal Reachability
- Experiments
- Conclusion
- Future Work

Motivation – Swarm verification

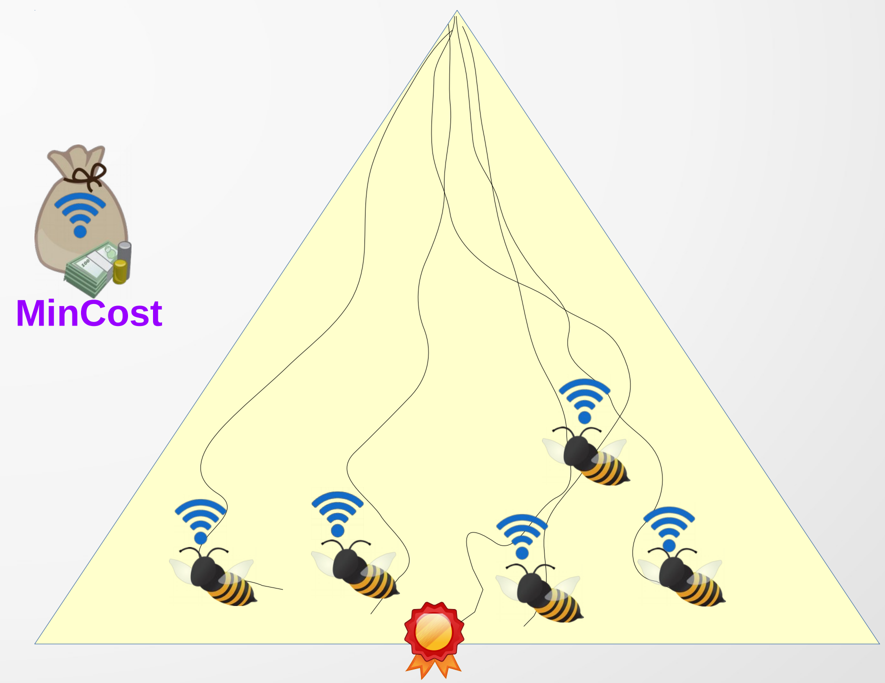
- As the number of components in the model increases, state explosion problem appears.
- Two main paradigms to speedup exploration and alleviate state explosion problem:
  - Parallel- and distributed computing: DiVinE, LTSmin, etc
  - Swarm verification: Swarm-Spin
- Swarm verification: a large number of model checker instances run in parallel on a computer cluster, using different, typically randomized search strategies.

Motivation – Swarm verification

- Swarm verification weak points:
  - No data parallelism, barely cut down termination time;
  - Duplicate work, duplicate copy of (partial) state space.
- Swarm verification strong points:
  - Get high quality results fast before exploring full state space, especially under time- and memory constrains
  - Easy to implement.
- “Even though each instance is tiny, there is strength in numbers.”

Motivation – Swarm verification

- We propose swarm verification using Uppaal to accelerate time optimal reachability.
- Swarm instances can exchange better costs to make more efficient pruning on all instances.
- Four swarm algorithms:
  - P-RDFS,
  - S-RDFS,
  - S-Mix,
  - S-Agent.



Contents

- Motivation
- Sequential Time Optimal Reachability
- Swarm Time Optimal Reachability
- Experiments
- Conclusion
- Future Work

Sequential Time Optimal Reachability

Algorithm 1: Sequential Time Optimal Reachability

$\textsc{Passed} \longleftarrow \emptyset, \textsc{Waiting} \longleftarrow \{(\ell_0, Z_0)\}, \textsc{Cost} \longleftarrow \infty$

Maintain current best span

$\textsc{Order} \in \{\textsc{Dfs}, \textsc{Bfs}, \textsc{Rdfs}\}$

Procedure `Main()`

    while $\textsc{Waiting} \neq \emptyset$ **do**

        `Search()`

    return $\textsc{Cost}$

Procedure `Search()`

    select (ℓ, Z) from $\textsc{Waiting}$ on $\textsc{Order}$

    if $(\ell, Z) \models$ `Goal` **then**

        if `MinCost`$(\ell, Z) < \textsc{Cost}$ **then**

            $\textsc{Cost} \longleftarrow$ `MinCost`(ℓ, Z)

    else if $(\ell, Z) \notin \textsc{Passed}$ *and* `MinCost`$(\ell, Z) < \textsc{Cost}$ **then**

        add (ℓ, Z) to $\textsc{Passed}$

        forall the (ℓ', Z') *such that* $(\ell, Z) \rightsquigarrow (\ell', Z')$ **do**

            add (ℓ', Z') to $\textsc{Waiting}$

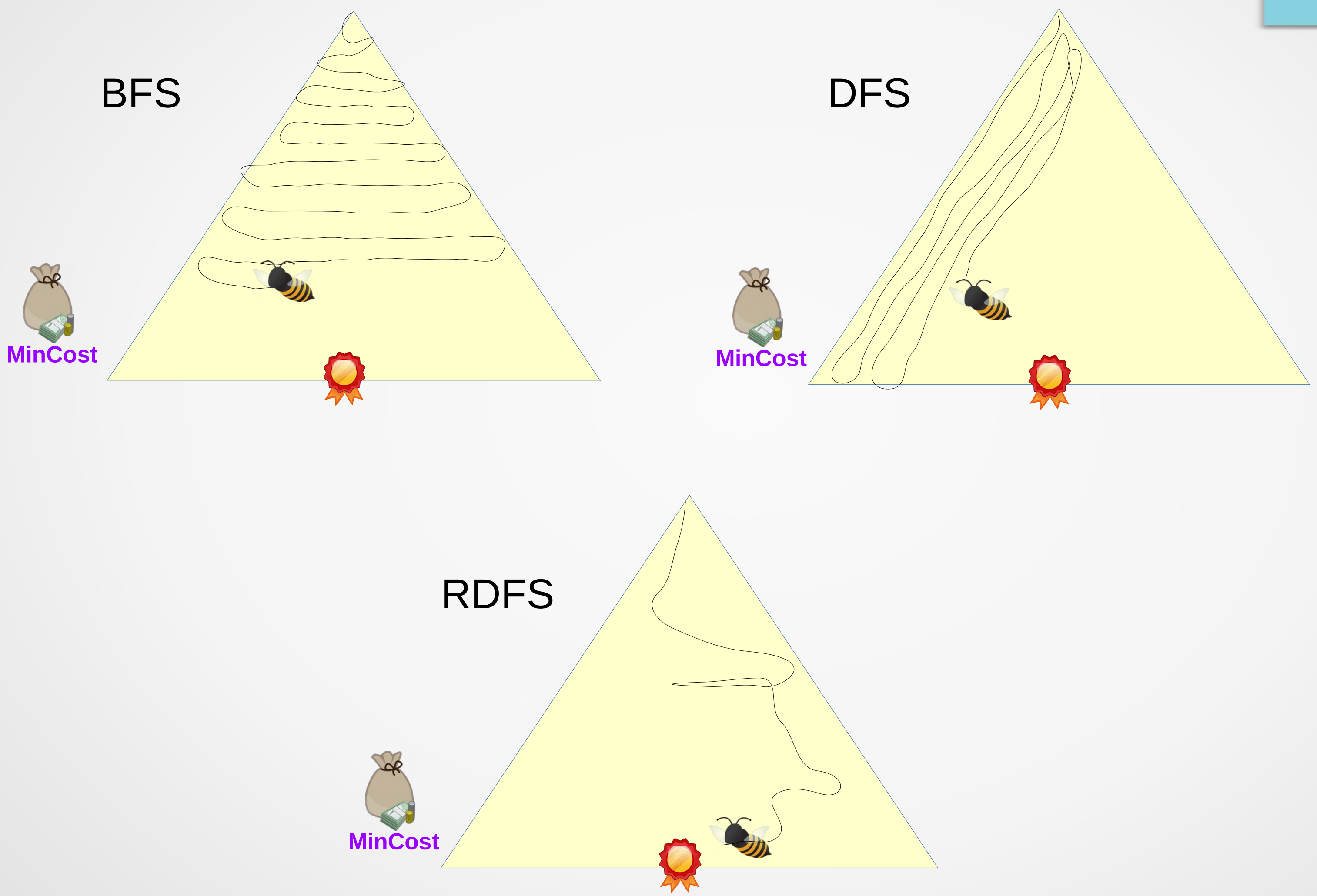
Update cost

Inclusion checking

Branch and bound elimination

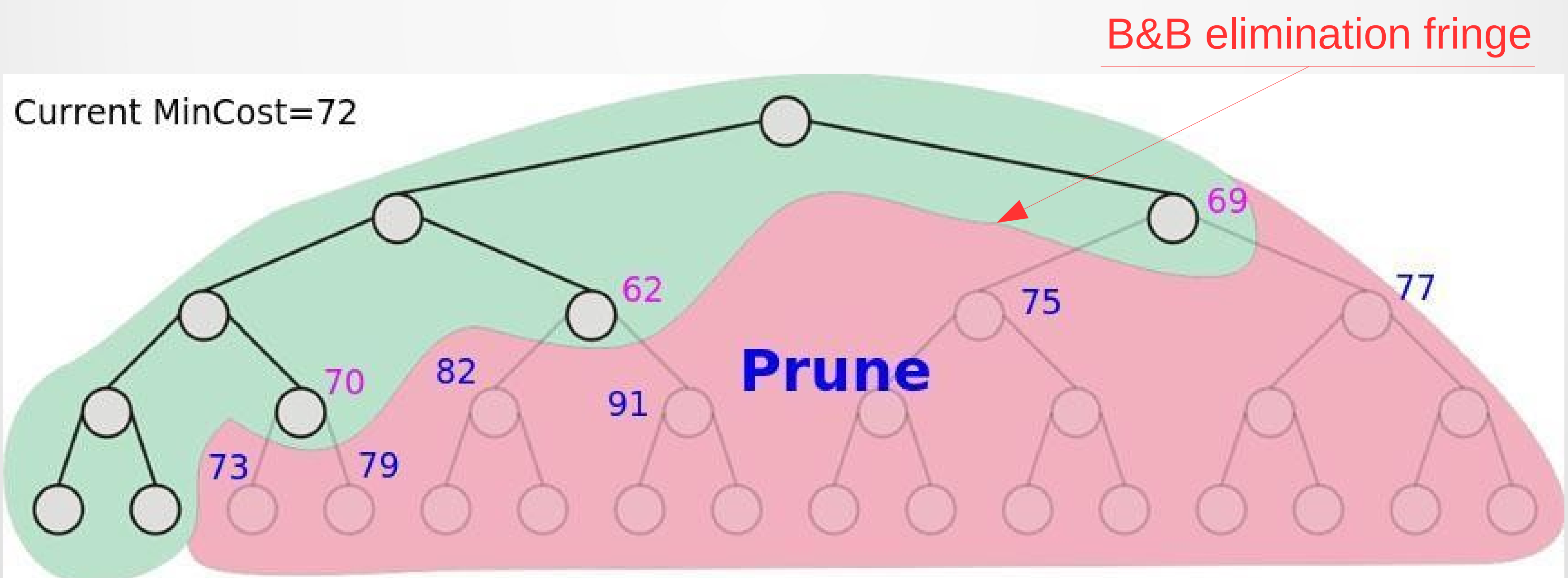
Generate successors

Sequential Time Optimal Reachability



Sequential Time Optimal Reachability

- Branch and Bound (B&B)
  - Pruning technique, avoid enumerate full state space;
  - Skip nodes whose cost > current best cost to the goal;
  - Heuristic function.



Contents

- Motivation
- Sequential Time Optimal Reachability
- Swarm Time Optimal Reachability
- Experiments
- Conclusion
- Future Work

Swarm Time Optimal Reachability

- Benefits of RDFS: random search, give results in inexhaustible state space.
- Four swarm algorithms based on RDFS:
  - Primitive swarm: do not exchange cost
    - P-RDFS: all do RDFS, first-complete-terminate
  - Cooperative swarm: exchange cost
    - S-RDFS: as P-RDFS
    - S-Mix: one (BFS), others (RDFS)
    - S-Agent: root (BFS), agents (RDFS), agents periodically ask states from Root to search from, root terminates agents.

Is sharing cost effective?

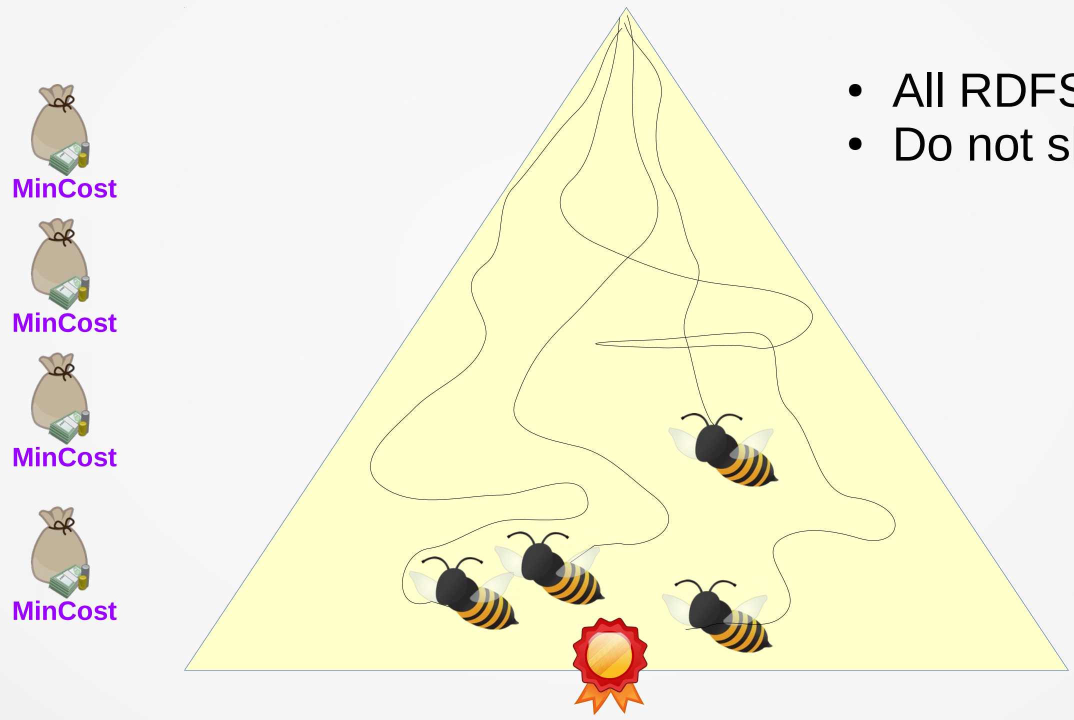
Why BFS?

Swarm Time Optimal Reachability

- BFS is important in S-Mix and S-Agent:
  - In Uppaal, BFS runs much faster than DFS and RDFS.
  - BFS builds large zones while DFS/RDFS causes high level of segmentations.
- In a word, a BFS instance enables swarm verification terminate fast.

Swarm Time Optimal Reachability

- P-RDFS: primitive version of swarm

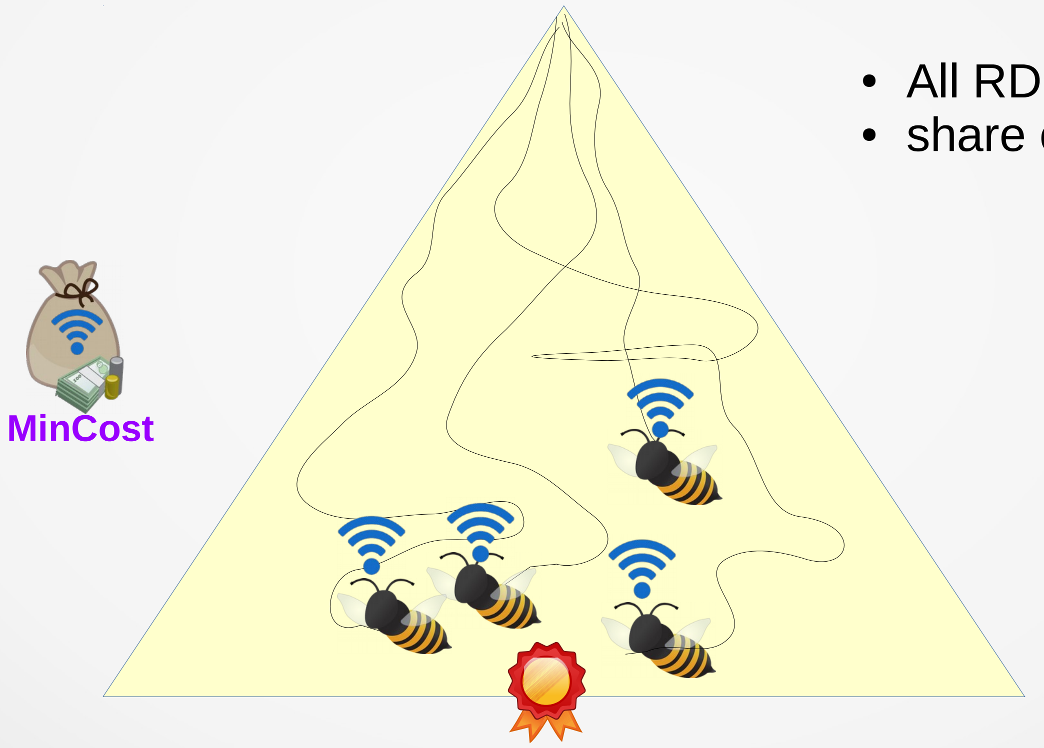


- All RDFS
- Do not share cost

Algorithm in Appendix A

Swarm Time Optimal Reachability

- S-RDFS: Cooperative Swarm RDFS

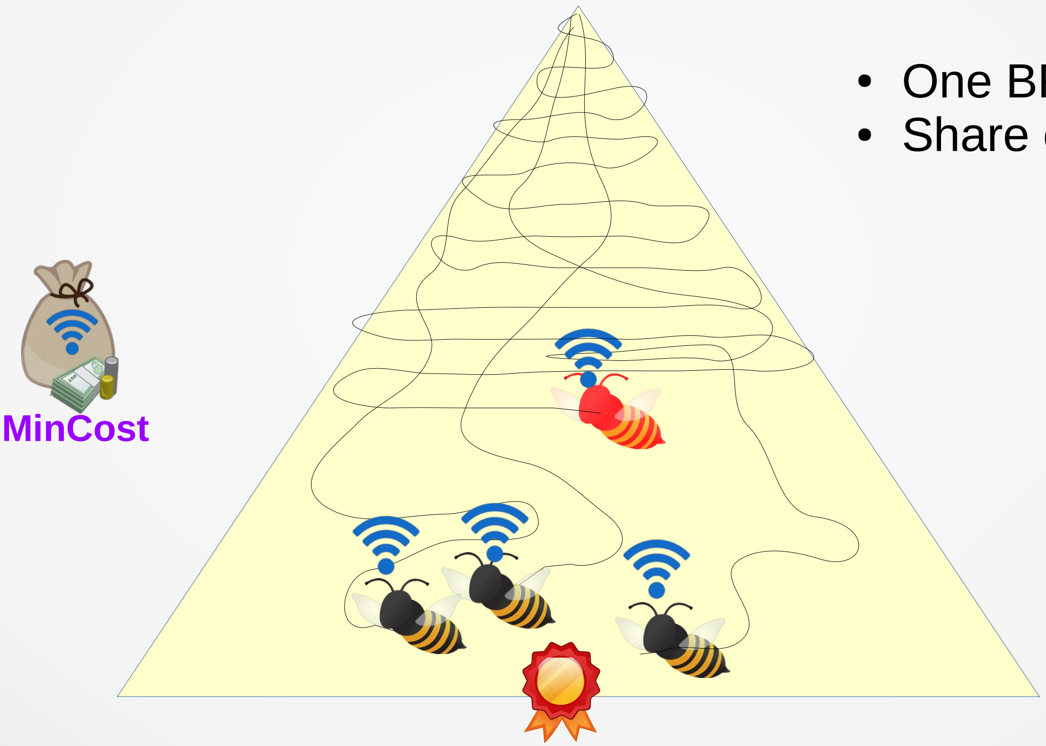


- All RDFS
- share cost

Algorithm in Appendix B

Swarm Time Optimal Reachability

- S-Mix: Cooperative Swarm Mix

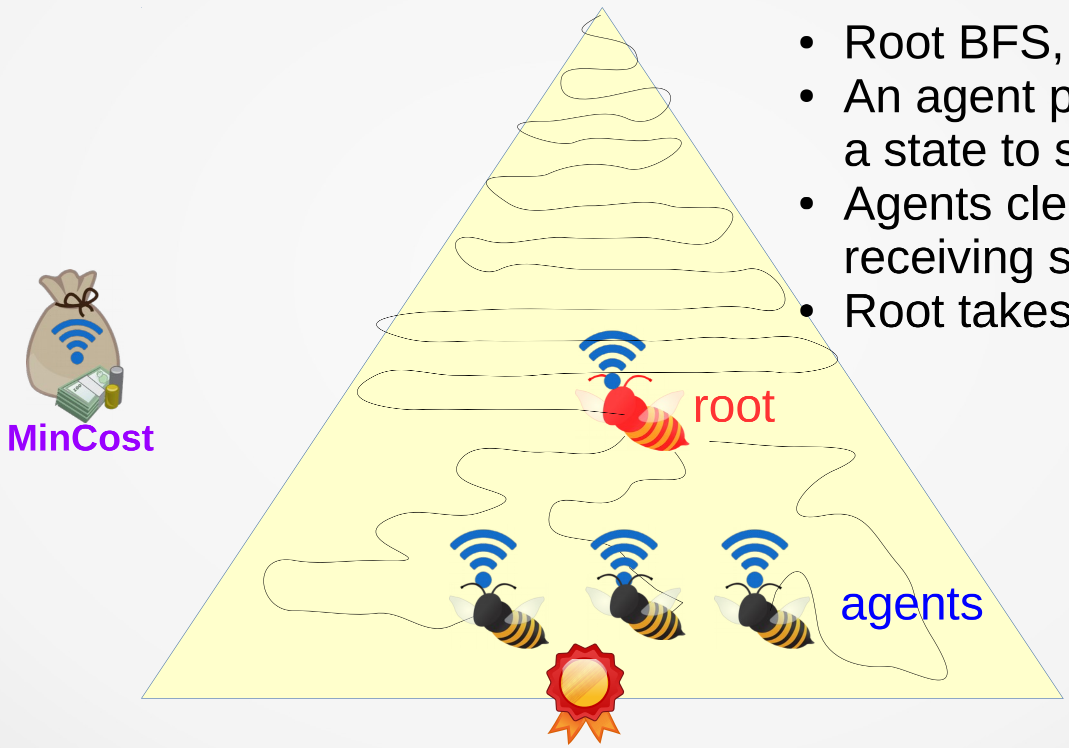


- One BFS, others RDFS.
- Share cost.

Algorithm in Appendix C

Swarm Time Optimal Reachability

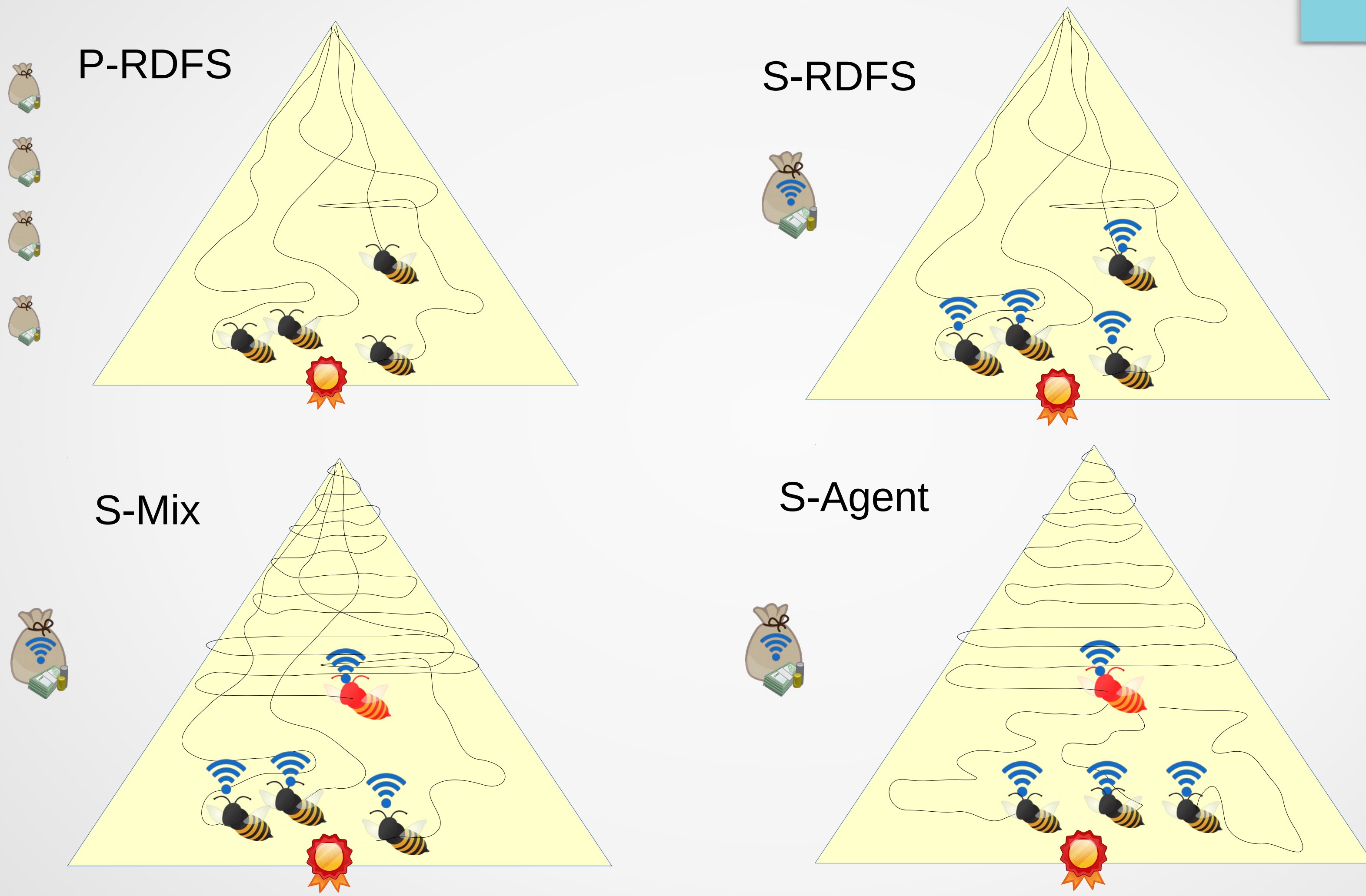
- S-Agent: Cooperative Swarm Agent



- Root BFS, agents RDFS;
- An agent periodically asks root for a state to search from.
- Agents clear passed/waiting on receiving states. Light weight.
- Root takes charge of termination.

Algorithm in Appendix D

Swarm Time Optimal Reachability



Contents

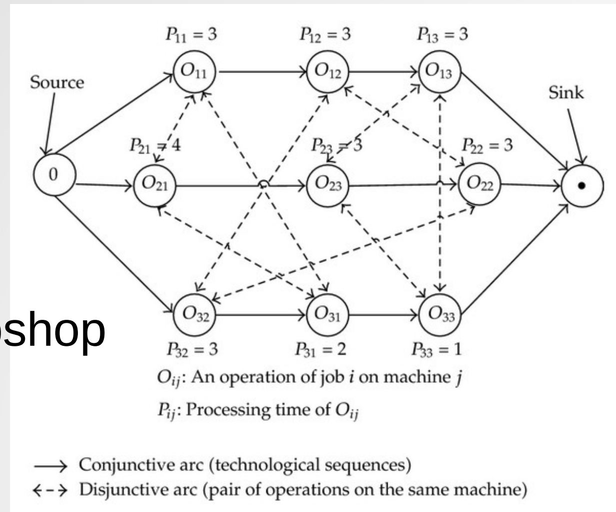
- Motivation
- Sequential Time Optimal Reachability
- Swarm Time Optimal Reachability
- Experiments
- Conclusion
- Future Work

Experiments

- Four timed scheduling models:
  - Job-shop-6 (jb-6): 6 people share a 4-section newspaper. Each person has own reading speed and favorable reading order on sections.
  - Aircraft-landing-15 (alp-15): 15 planes with different target landing time, need to land on 2 runways. Planes also have different landing intervals in between to avoid turbulence
  - Viking-bridge-15 (vik-15): 15 vikings with different walking speeds and sharing a single torch, cross a damaged bridge (Max 2 per) in darkness.
  - Task-graph-88 (task-88): non-preemptive schedule for 88 tasks with different precedence constraints on 4 CPUs with different frequencies (speeds).

Experiments

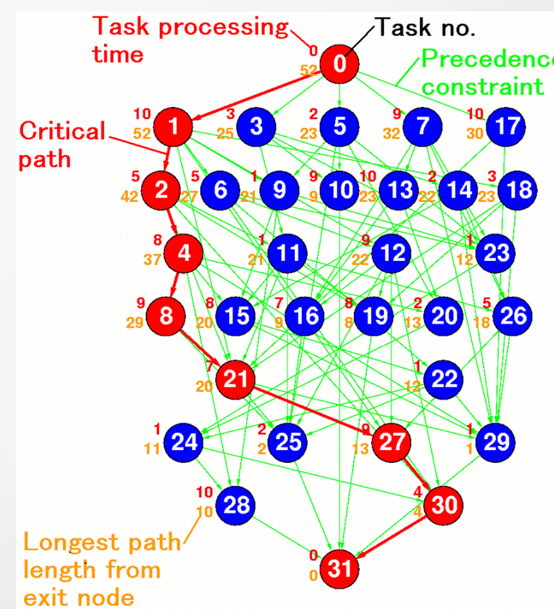
Jobshop



Airplane Landing

Viking Bridge

Task Graph

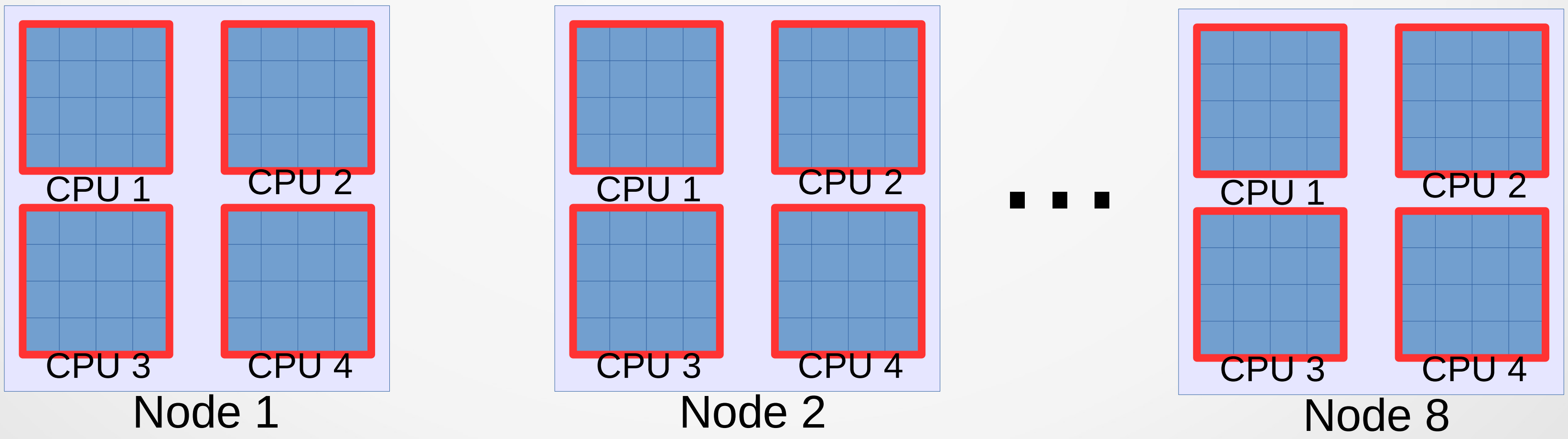


Experiments - Metric

- Metric of performance:
  - Metric 1: Shortest time to find optimal result t_{opt}
  - Metric 2: Time to terminate thus prove optimal result t_{prov}
  - Metric 3: Progressively improved near optimal results as a function of time, converge speed.
  - Metric 4: Memory consumption.

Experiments - Settings

- Experiment settings:
  - Cluster: NUMA architecture, 9 computing nodes, 1 TB per node, 4 CPU @ 2.3GHz per node, 16 cores per CPU, 574 cores total.
  - Algorithms: sequential-BFS/DFS/RDFS, P-RDFS, S-RDFS, S-Mix, S-Agent, 4-hour limit, 15 runs per core setting.



Jobshop-6 t_{opt} and t_{prov} (Metric 1&2)

	Job-Shop-6						
#C	BFS		DFS		RDFS		
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}	
1	108	108	7877	9899	650	2283	

- BFS runs much faster than DFS/RDFS

Jobshop-6 t_{opt} and t_{prov} (Metric 1&2)

	Job-Shop-6					
#C	BFS		DFS		RDFS	
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}
1	108	108	7877	9899	650	2283

#C	P-RDFS	
	t_{opt}	t_{prov}
2	535	1996
4	310	1730
8	176	1720
16	100	1591
32	65	1488
64	32	1452
128	4	1412
256	2	1365
512	<1	1337

Table 1: Runtime (s

P-RDFS:

- Good speed up for $t_{opt,}$ outperforms BFS above 16 cores.
- Limited speed up for t_{prov}.

Jobshop-6 t_{opt} and t_{prov} (Metric 1&2)

	Job-Shop-6					
#C	BFS		DFS		RDFS	
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}
1	108	108	7877	9899	650	2283

#C	P-RDFS		S-RDFS	
	t_{opt}	t_{prov}	t_{opt}	t_{prov}
2	535	1996	459	1821
4	310	1730	280	1683
8	176	1720	116	1653
16	100	1591	87	1506
32	65	1488	52	1494
64	32	1452	24	1385
128	4	1412	4	1367
256	2	1365	<1	1341
512	<1	1337	<1	1263

Table 1: Runtime (sec) of Job-Shop

S-RDFS vs. P-RDFS
- Sharing cost improves t_{opt} 15% and t_{prov} by 5%.

Jobshop-6 t_{opt} and t_{prov} (Metric 1&2)

#C	BFS t_{opt}	BFS t_{prov}	DFS t_{opt}	DFS t_{prov}	RDFS t_{opt}	RDFS t_{prov}
1	108	108	7877	9899	650	2283

#C	P-RDFS t_{opt}	P-RDFS t_{prov}	S-RDFS t_{opt}	S-RDFS t_{prov}	S-Mix t_{opt}	S-Mix t_{prov}
2	535	1996	459	1821	102	102
4	310	1730	280	1683	102	102
8	176	1720	116	1653	101	102
16	100	1591	87	1506	99	101
32	65	1488	52	1494	64	99
64	32	1452	24	1385	24	98
128	4	1412	4	1367	4	98
256	2	1365	<1	1341	<1	98
512	<1	1337	<1	1263	<1	100

Table 1: Runtime (sec) of Job-Shop-6 and Aircraf

S-Mix
- BFS instance allows best t_{prov}. and best t_{opt} between core 2 ~ 8.

Jobshop-6 t_{opt} and t_{prov} (Metric 1&2)

	Job-Shop-6							
#C	BFS		DFS		RDFS			
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}		
1	108	108	7877	9899	650	2283		
#C	P-RDFS		S-RDFS		S-Mix		S-Agent	
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}
2	535	1996	459	1821	102	102	39	99
4	310	1730	280	1683	102	102	25	98
8	176	1720	116	1653	101	102	17	97
16	100	1591	87	1506	99	101	14	97
32	65	1488	52	1494	64	99	13	96
64	32	1452	24	1385	24	98	16	95
128	4	1412	4	1367	4	98	29	95
256	2	1365	<1	1341	<1	98	28	95
512	<1	1337	<1	1263	<1	100	31	96

Table 1: Runtime (sec) of Job-Shop-6 and Aircraft-Landing-15

S-Agent vs. S-Mix
- Even better t_{opt} from core 2 ~ 64
- Equally good t_{prov}.

Aircraft-15 t_{opt} and t_{prov} (Metric 1&2)

	Aircraft-Landing-15							
#C	BFS		DFS		RDFS			
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}		
1	157	159	73	428	191	964		
#C	P-RDFS		S-RDFS		S-Mix		S-Agent	
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}
2	87	845	93	873	91	91	66	90
4	42	737	35	729	46	90	21	91
8	22	712	16	727	17	89	25	88
16	17	676	11	699	11	89	6	86
32	2	638	1	633	2	89	12	86
64	<1	604	<1	582	<1	88	5	85
128	<1	597	<1	620	<1	88	2	84
256	<1	581	<0.1	565	<0.1	88	7	84
512	<1	585	<0.1	573	<0.1	117	5	86

Table 1: Runtime (sec) of Job-Shop-6 and Aircraft-Landing-15

- Similiar pattern as Jobshop
- Sharing costs allow S-Mix and S-Agent 45% faster in t_{prov} than BFS.

Viking-15 t_{opt} and t_{prov} (Metric 1&2)

```
mincost=669930, timestamp=9.5024
mincost=669925, timestamp=10.455
mincost=669910, timestamp=10.467
mincost=669905, timestamp=11.420
mincost=669895, timestamp=12.481
mincost=669890, timestamp=13.408
mincost=669885, timestamp=13.418
mincost=669880, timestamp=14.344
mincost=669870, timestamp=14.350
mincost=669865, timestamp=14.361
mincost=669815, timestamp=15.330
mincost=669795, timestamp=16.324
mincost=669790, timestamp=17.245
mincost=669785, timestamp=18.332
mincost=669780, timestamp=19.289
mincost=669775, timestamp=20.205
mincost=669770, timestamp=21.323
mincost=669765, timestamp=21.336
mincost=669760, timestamp=22.363
mincost=669755, timestamp=23.332
mincost=669750, timestamp=24.316
mincost=669745, timestamp=25.404
```

…

mincost=220

Fine grain cost

#C	BFS		DFS		RDFS	
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}
1	70	70	-	-	-	-

#C	P-RDFS		S-RDFS		S-Mix		S-Agent	
	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}	t_{opt}	t_{prov}
2	-	-	-	-	96	96	75	75
4	-	-	-	-	87	87	72	72
8	-	-	-	-	96	96	72	72
16	-	-	-	-	92	92	83	83
32	-	-	-	-	100	100	94	94
64	-	-	-	-	107	107	77	77
128	-	-	-	-	112	112	74	74
256	-	-	-	-	186	186	71	71
512	-	-	-	-	215	215	73	73

"-": denotes timeout.

Table 2: Runtime (sec) of Viking-Bridge-15

- Sharing costs decelerate S-Mix and S-Agent.

Experiments – t_{opt} and t_{prov} (Metric 1&2)

- Summary for (Metric 1&2)
  - Exchanging cost can generally improve t_{opt}, at low core settings (2 ~ 64);
  - S-Mix and S-Agent combines benefits of BFS and RDFS. They can report results and terminate faster;
  - Costs reported by RDFS instances can help BFS root in pruning, but fine-grained improved costs may backfire.

Jobshop-6 Results versus time (Metric 3)

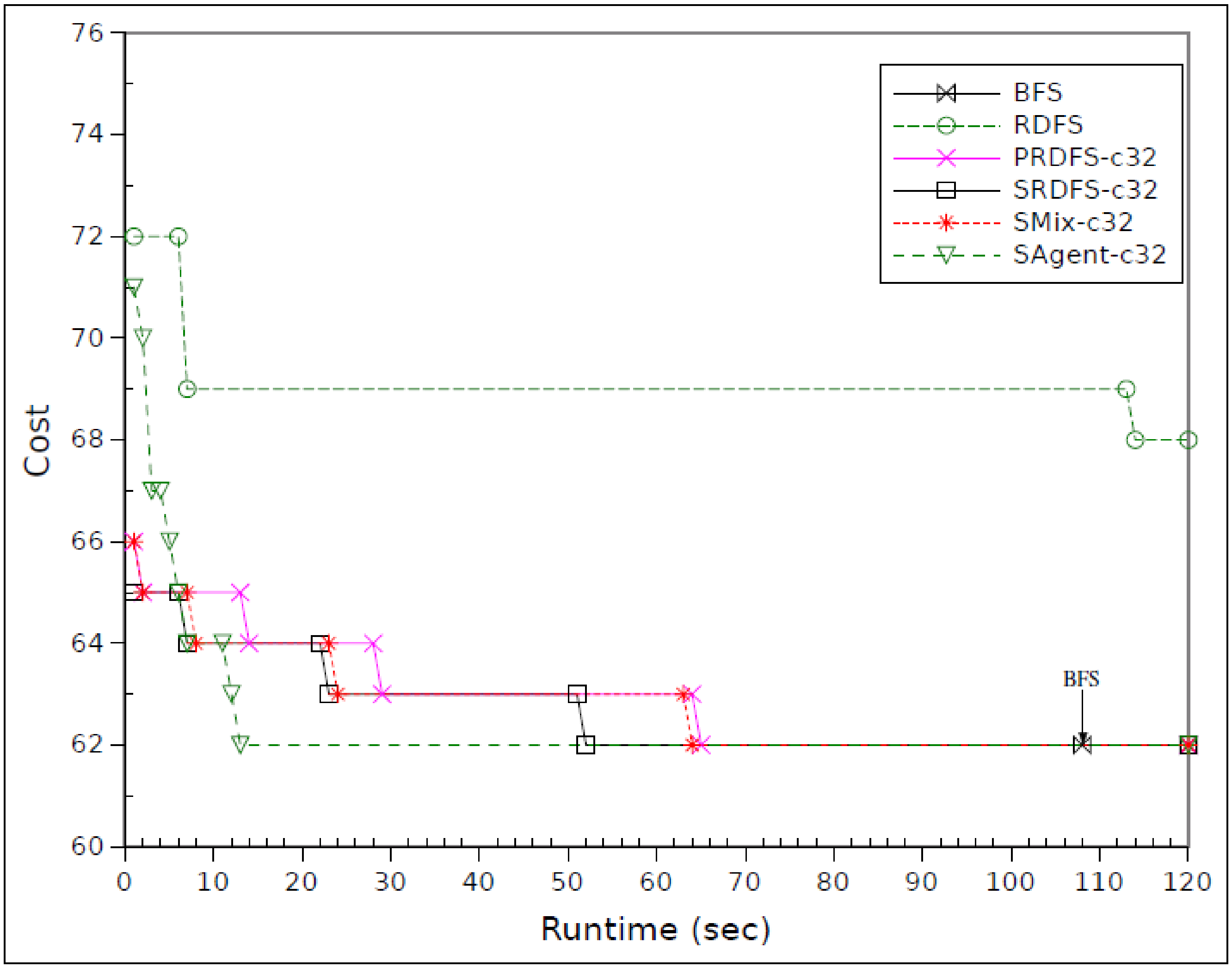


Figure 1: Cost vs. Runtime for Job-Shop-6

Task-88 Results versus time (Metric 3)

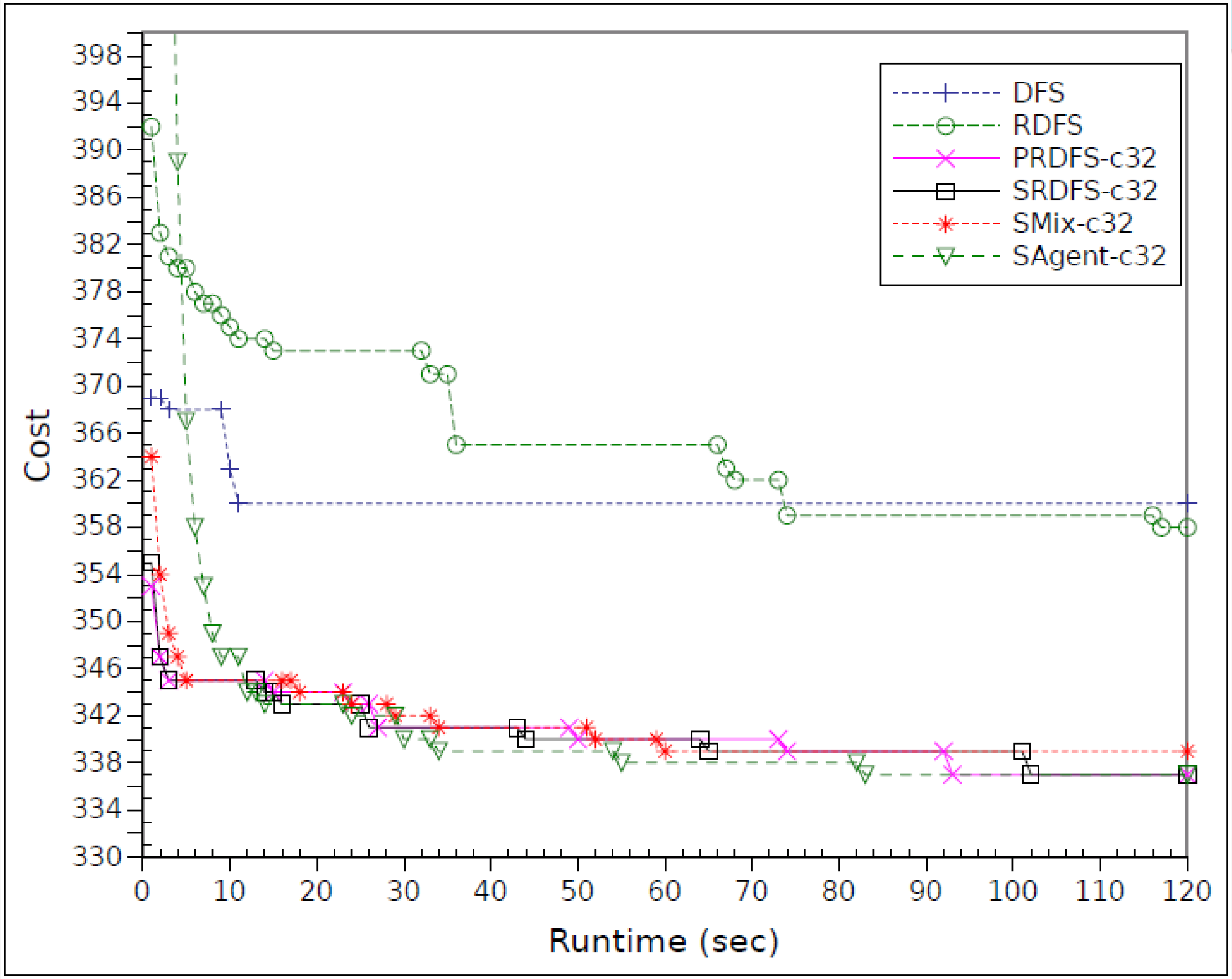


Figure 2: Cost vs. Runtime for Task-Graph-88

Aircraft-15 Results versus time (Metric 3)

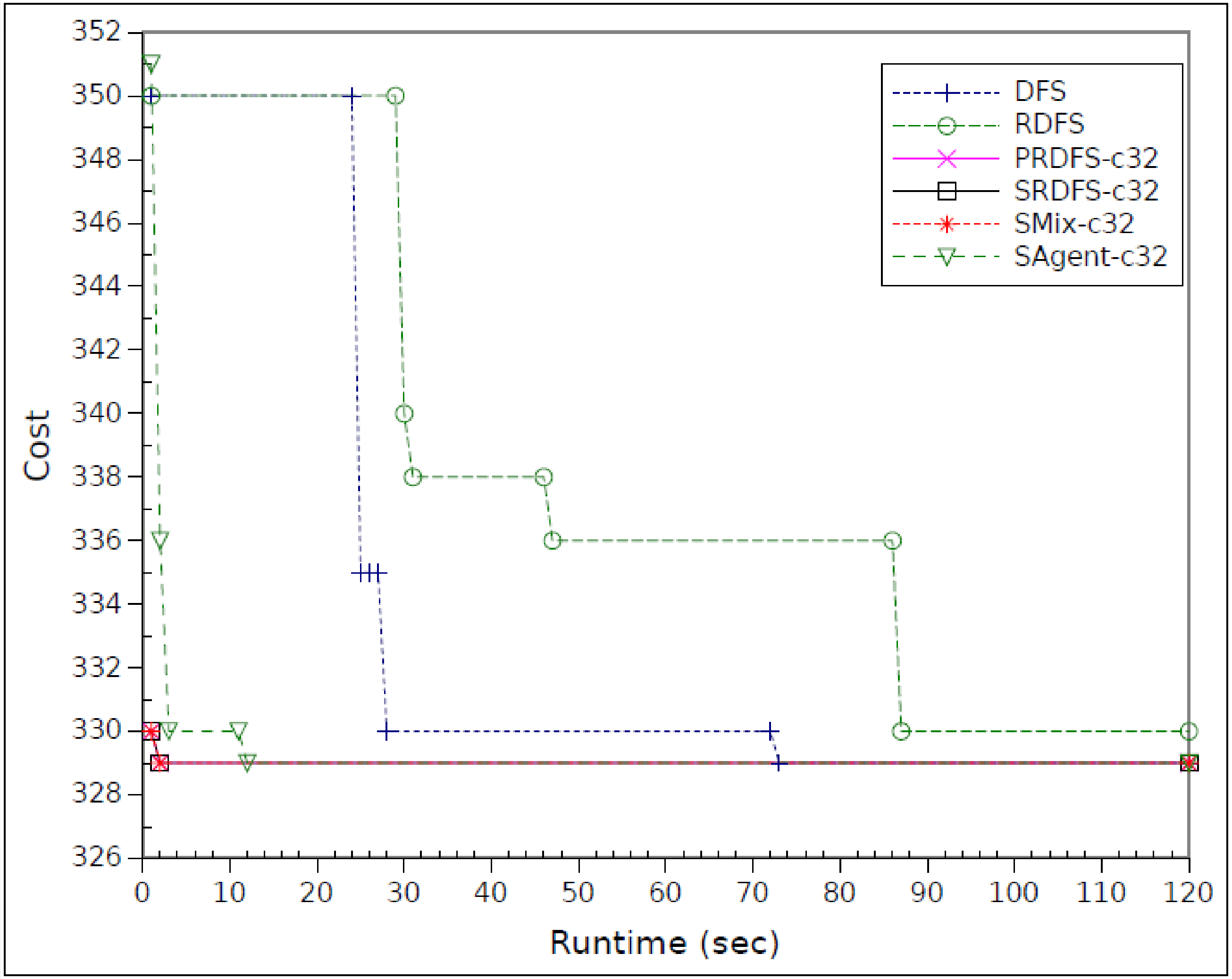


Figure 3: Cost vs. Runtime for Aircraft-Landing-15

Viking-15 Results versus time (Metric 3)

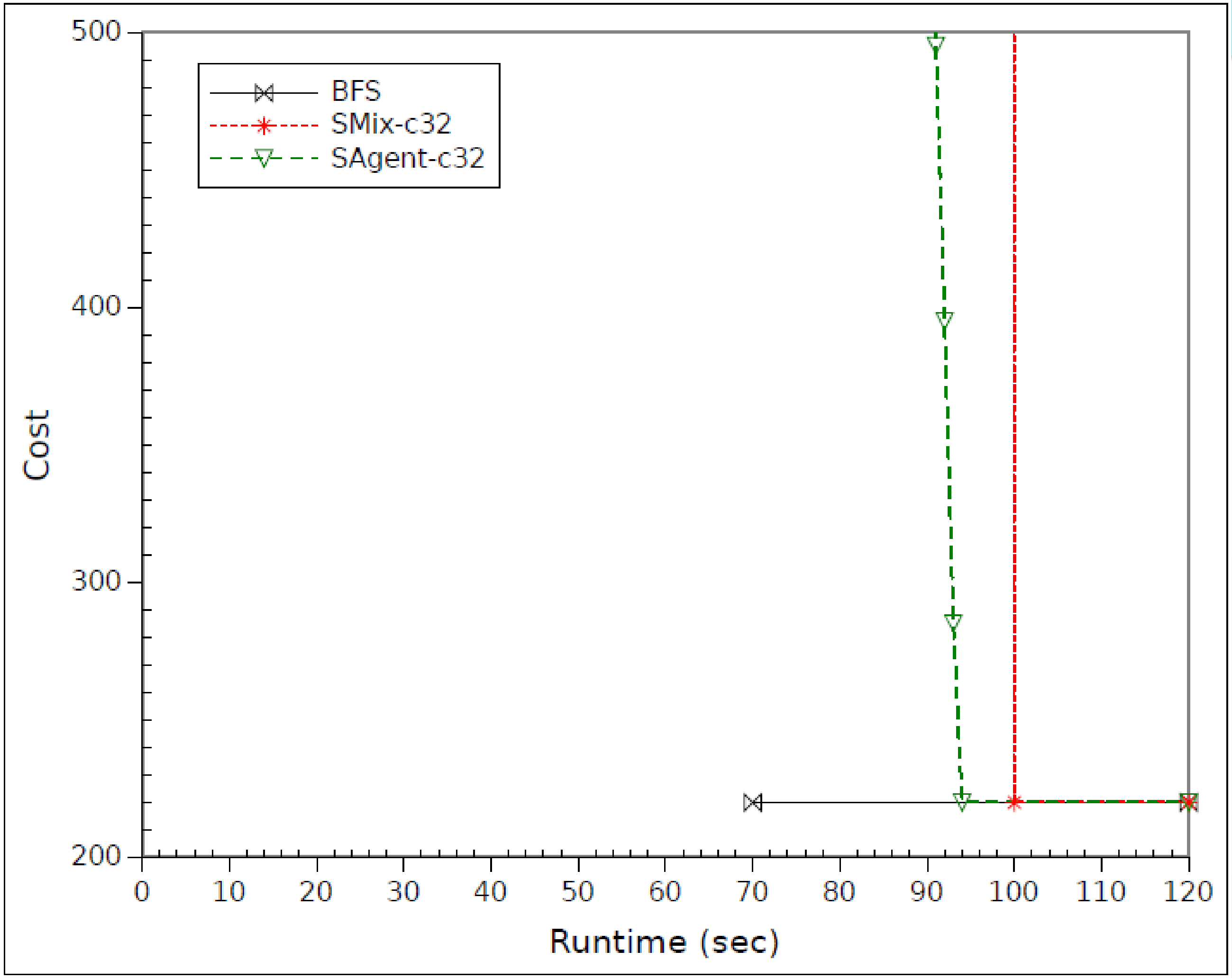


Figure 4: Cost vs. Runtime for Viking-Bridge-15

Experiments – Results versus time (Metric 3)

- Summary (Metric 3)
  - Swarm algorithms are generally faster than sequential algorithms at finding near optimal results, and they work equally well.

Experiments – Memory consumption (Metric 4)

Models	jb-6	alp-15	vik-15	task-88
BFS	152	124	408	*5068*
DFS	1754	127	*13297*	*3060*
RDFS	606	191	*12790*	*3668*
P-RDFS-c32	593	187	*13555*	*3849*
S-RDFS-c32	591	190	*15303*	*3873*
S-Mix-c32	63	51	230	*3914*
S-Agent-c32	21	13	100	*1277*
Italic font denotes termination due to timeout.				

RDFS instances/agents amortize average

Table 3: Resident Memory (MB) per UPPAAL Instance

- Summary (Metric 4)
  - S-Agent has the best average memory footprint.

Contents

- Motivation
- Sequential Time Optimal Reachability
- Swarm Time Optimal Reachability
- Experiments
- Conclusion
- Future Work

Conclusion

- Swarm algorithms generally find optimal (or near optimal) results faster than sequential algorithms.
- S-Mix and S-Agent combine the benefits of BFS and RDFS. They find results and terminate fast.
- S-Agent has smaller memory footprint because agents do not keep state space.
- Sharing cost is useful for speed up at lower core settings.

Future Work

- Swarm intelligence algorithms from AI:
  - Ant colony optimization algorithms (ACO)
  - Particle swarm optimization (PSO)
  - Rapidly-exploring Random Tree (RRT)
- Parallel- and distributed algorithms
- Extend to more general priced time automata (PTA).

Thank You for Your Attention

Appendix A: P-RDFS Algorithm

- P-RDFS

Algorithm 2: Basic Swarm RDFS Time Optimal Reachability (P-RDFS)

All do RDFS

```
PASSED ⟵ ∅, WAITING ⟵ {(ℓ_0, Z_0)}, COST ⟵ ∞
ORDER ⟵ RDFS, TERMINATE ⟵ FALSE
Procedure Main()
    while WAITING ≠ ∅ and ¬TERMINATE do
        Search()
    stop other instances
    return COST
```

First-complete-terminate

Appendix B: S-RDFS Algorithm

- S-RDFS:

Algorithm 3: Cooperative Swarm RDFS Time Optimal Reachability (S-RDFS)

(Local Variables)

$\textsc{Passed} \longleftarrow \emptyset$, $\textsc{Waiting} \longleftarrow \{(\ell_0, Z_0)\}$

$\textsc{Cost} \longleftarrow \infty$, $\textsc{Ecost} \longleftarrow \infty$

Cost received externally

$\textsc{Order} \longleftarrow \textsc{Rdfs}$, $\textsc{terminate} \longleftarrow \textsc{false}$

(Message Types)

Notify better cost

UPDATE

Procedure `Main()`

while $\textsc{Waiting} \neq \emptyset$ *and* $\neg\textsc{terminate}$ **do**

`// lines [2,3] are called atomically`

if $\textsc{Ecost} < \textsc{Cost}$ **then** $\textsc{Cost} \longleftarrow \textsc{Ecost}$

Before local search
Compare costs

if $\textsc{Cost} < \textsc{Ecost}$ **then**

`Update`($\textsc{Cost}$), `Broadcast`(UPDATE, $\textsc{Cost}$)

`Search()`

stop other instances

return $\textsc{Cost}$

Procedure `Update`($\textsc{NewCost}$) `// called atomically`

if $\textsc{NewCost} < \textsc{Ecost}$ **then** $\textsc{Ecost} \longleftarrow \textsc{NewCost}$

(Message Processing Rules)

Message handler

When a node receives UPDATE$\langle\textsc{Ncost}\rangle$, `Update`($\textsc{Ncost}$).

Appendix C: S-Mix Algorithm

- S-Mix

Algorithm 4: Cooperative Swarm Mix Time Optimal Reachability (S-Mix)

(Local Variables)

$$\textsc{Order} \longleftarrow \begin{cases} \text{BFS} & \text{if } p = 0, \\ \text{RDFS} & \text{otherwise.} \end{cases}$$

One (BFS), others (RDFS)

The rest is the same as Algorithm 3

Appendix D: S-Agent Algorithm

- S-Agent

Algorithm 5: Cooperative Swarm Agent Time Optimal Reachability (S-Agent)

(Local Variables)

$\textsc{Passed} \longleftarrow \emptyset, \textsc{Cost} \longleftarrow \infty, \textsc{Ecost} \longleftarrow \infty$

$$\textsc{Waiting} \longleftarrow \begin{cases} \{(\ell_0, Z_0)\} & \text{if } p = root, \\ \emptyset & \text{otherwise.} \end{cases}$$

$$\textsc{Order} \longleftarrow \begin{cases} \text{BFS} & \text{if } p = root, \\ \text{RDFS} & \text{otherwise.} \end{cases}$$

Root (BFS), agents (RDFS)
Root searches from initial state
Agents have empty waiting list

(Message Types)

UPDATE, REQUEST, ISSUE

Two more messages

Root

```
Procedure MainRoot()
    while WAITING ≠ ∅ do
        same as lines [2-4] in Algorithm 3
    stop all agents
    return COST
```

Root stops agents

Appendix D: S-Agent Algorithm

- S-Agent Cont.

Agent

Procedure MainAgent()

4 $\textsc{Iteration} \longleftarrow 0, \textsc{Limit} \in \mathbb{N}, \textsc{terminate} \longleftarrow \textsc{false}$
5 **while** $\neg\textsc{terminate}$ **do**
6   **if** $\textsc{Waiting} = \emptyset$ **then**
      Send(REQUEST, Root), Recv(ISSUE)
    // lines [7,8] are called atomically
7   **if** $\textsc{Ecost} < \textsc{Cost}$ **then** $\textsc{Cost} \longleftarrow \textsc{Ecost}$
8   **if** $\textsc{Cost} < \textsc{Ecost}$ **then**
      Update(Cost), Broadcast(UPDATE, Cost)
9   Search(), $\textsc{Iteration} \longleftarrow \textsc{Iteration} + 1$
10 **if** $\textsc{Iteration} \geq \textsc{Limit}$ **then**
11   Send(REQUEST, Root), Recv(ISSUE),
      $\textsc{Iteration} \longleftarrow 0$

Send request to root, then blocking receive a state from root

Send request in a period

Agent is light weight

(Message Processing Rules)

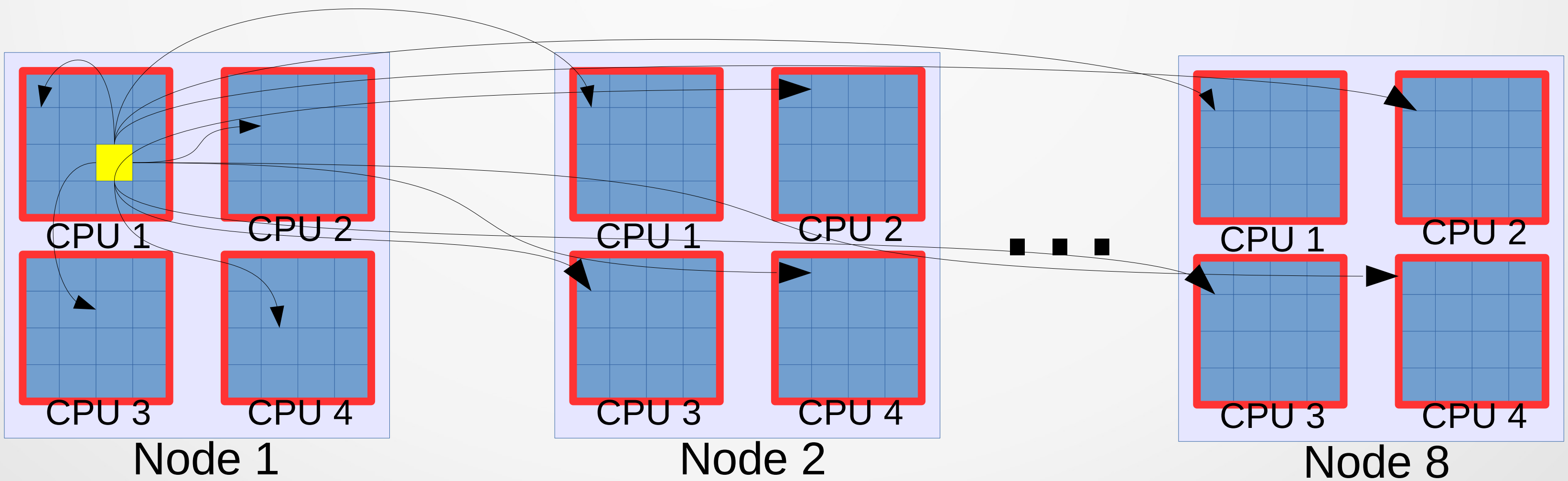
12 When a node receives UPDATE⟨Ncost⟩: Update(Ncost).
13 When root receives REQUEST from agent i: select (ℓ', Z') from Waiting, Send(ISSUE, $(\ell', Z'), i$).
14 When an agent receives ISSUE⟨(ℓ^*, Z^*)⟩ from root: $\textsc{Waiting} \longleftarrow \emptyset$, $\textsc{Passed} \longleftarrow \emptyset$, $\textsc{Waiting} \longleftarrow \{(\ell^*, Z^*)\}$.

Root hands Request
Agents handle Issue

Agents clear old state space when received a new state

Appendix E: Scalability problem

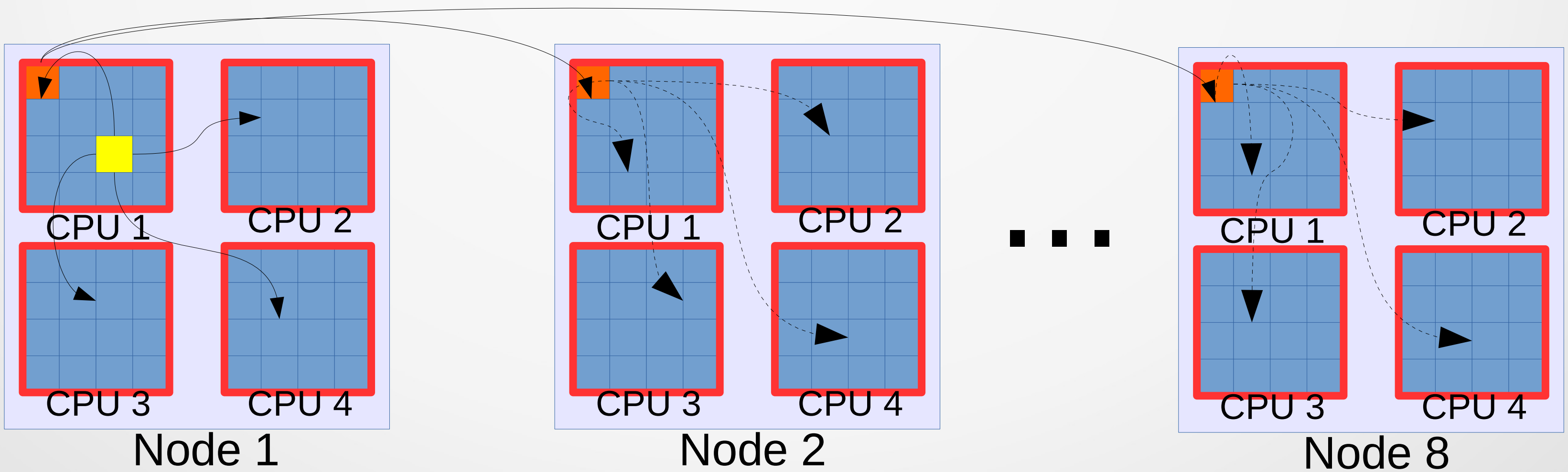
- Scalability problem:
  - Message flood: 512 instances broadcast costs at the same time will overload the cluster severely.
  - Cluster: NUMA architecture, 9 computing nodes, 1 TB per node, 4 CPU @ 2.3GHz per node, 16 cores per CPU, 574 cores total.



7×64 (=448) inter-node messages overload cluster heavily!!!

Appendix E: Scalability problem

- Optimization for scalability
  - Overcome by using cluster architecture, virtual topology;
  - Virtual node: one master core, other normal core;
  - Normal core broadcasts within same virtual node;
  - Only master core send/relay inter-node message.



Now only 7 inter-node messages. Scalability is retained!