Hierarchical Adaptive State Space Caching Based on Level Sampling

Radu Mateescu

Anton Wijs

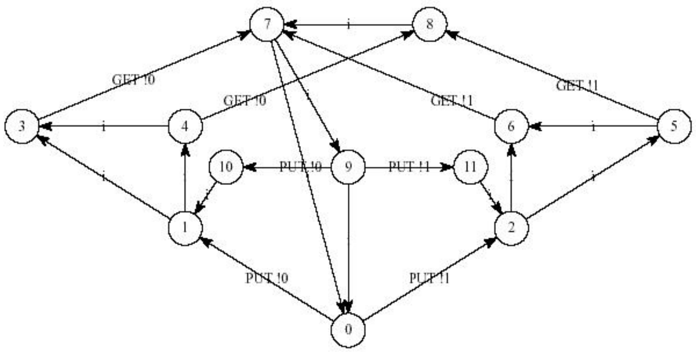
INRIA Rhône-Alpes / VASY

http://www.inrialpes.fr/vasy

Table of Contents

Labelled Transition System (LTS)

- LTS $M=(S,A,T,s_0)$ with
  - S a set of states
  - A a set of transition labels
  - $T: S \times A \times S$ a transition relation
  - s_0 the initial state
- Framework: the CADP toolbox
  - http://www.inrialpes.fr/vasy/cadp



Breadth-First Search

Algorithm 1 BFS

procedure BFS(s_0)
  $Open \leftarrow \{s_0\}$ {Initial state added to horizon}
  $Closed \leftarrow \emptyset$ {History is empty}
  while $Open \neq \emptyset$ **do** {Repeat until there are no more states to explore}
    for all $s \in Open$ **do** {Explore all states in the horizon}
      $Next \leftarrow Next \cup \{s' \mid \exists \ell.(s, \ell, s') \in en(s)\}$
    $Closed \leftarrow Closed \cup Open$ {Add explored states to history}
    $Open \leftarrow Next \setminus Closed$ {Add new states to horizon}

Table of Contents

Partial Storage of Search History

- State Space Explosion Problem
  - Linear growth of # concurrent processes -> exponential growth # states in LTS
- Memory problems due to having to store all states in history (*Closed*) in memory
- One approach is to consider partial storage -> research can be divided in two classes:
  - *Guaranteeing exhaustiveness*
  - Not guaranteeing exhaustiveness

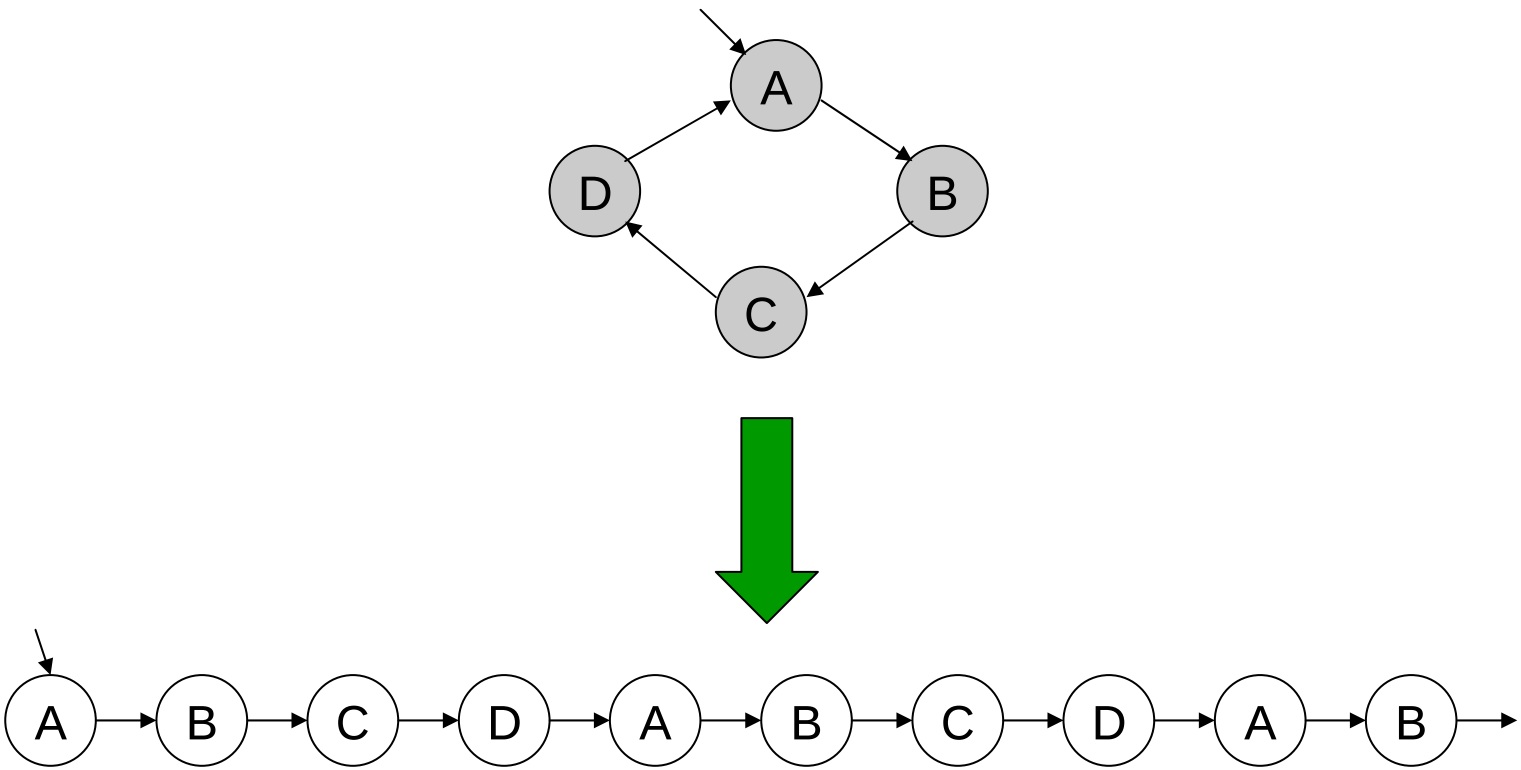
Guarantee exhaustiveness

- Depth-first search (DFS) With Caching
  - Holzmann ‘87
  - Jard & Jéron ’91
  - Godefroid et al. ’95 add POR and static analysis
- Behrmann et al. 2003 use storing strategies involving static analysis
- From AI, e.g. *IDA**, *MA**; use structural info

Our Goals

- Other attempts concern reachability analysis, we want *state space generation*
- -> Besides dropping memory use, we also want LTSs with few redundancy
- Most concern DFS, we also want BFS
- Probabilistic <-> Certainty
  - Exhaustiveness
  - Termination
- We do *not* want static analysis (language independent)

No Duplicate Detection



Partial Duplicate Detection

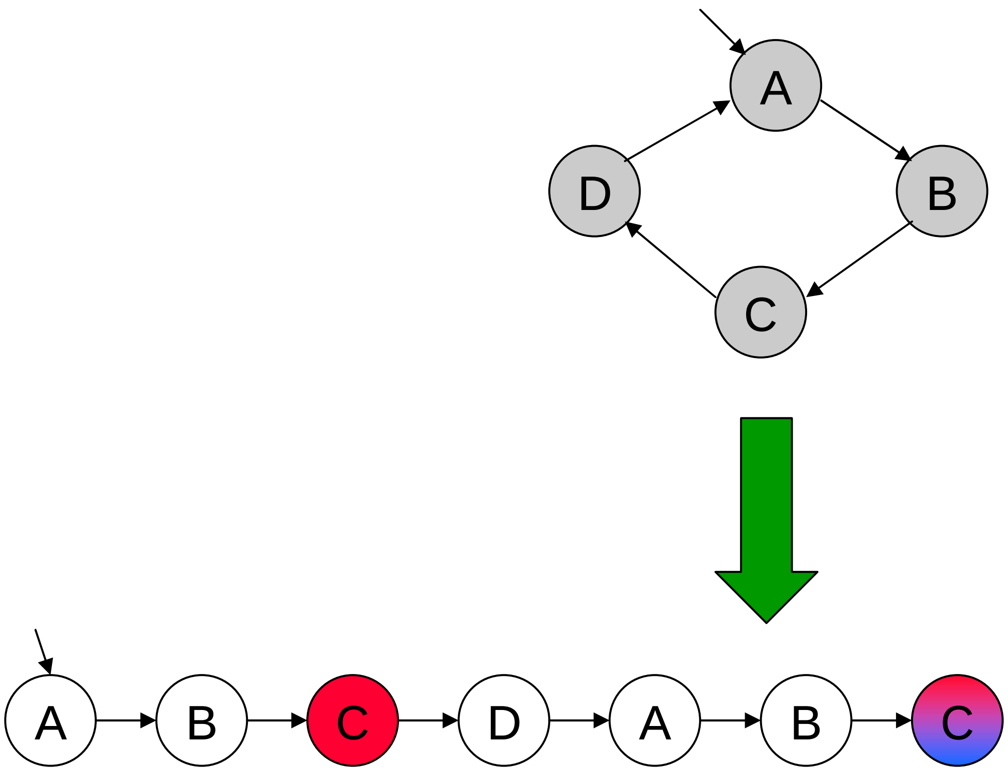


Table of Contents

- Labelled Transition System (LTS)/ Breadth-first Search (BFS)
- Partial Storage of Search History
- **BFS With Snapshots**
- BFS With Snapshot Caches
- Implementation
- Methods
- Conclusions & Future Work

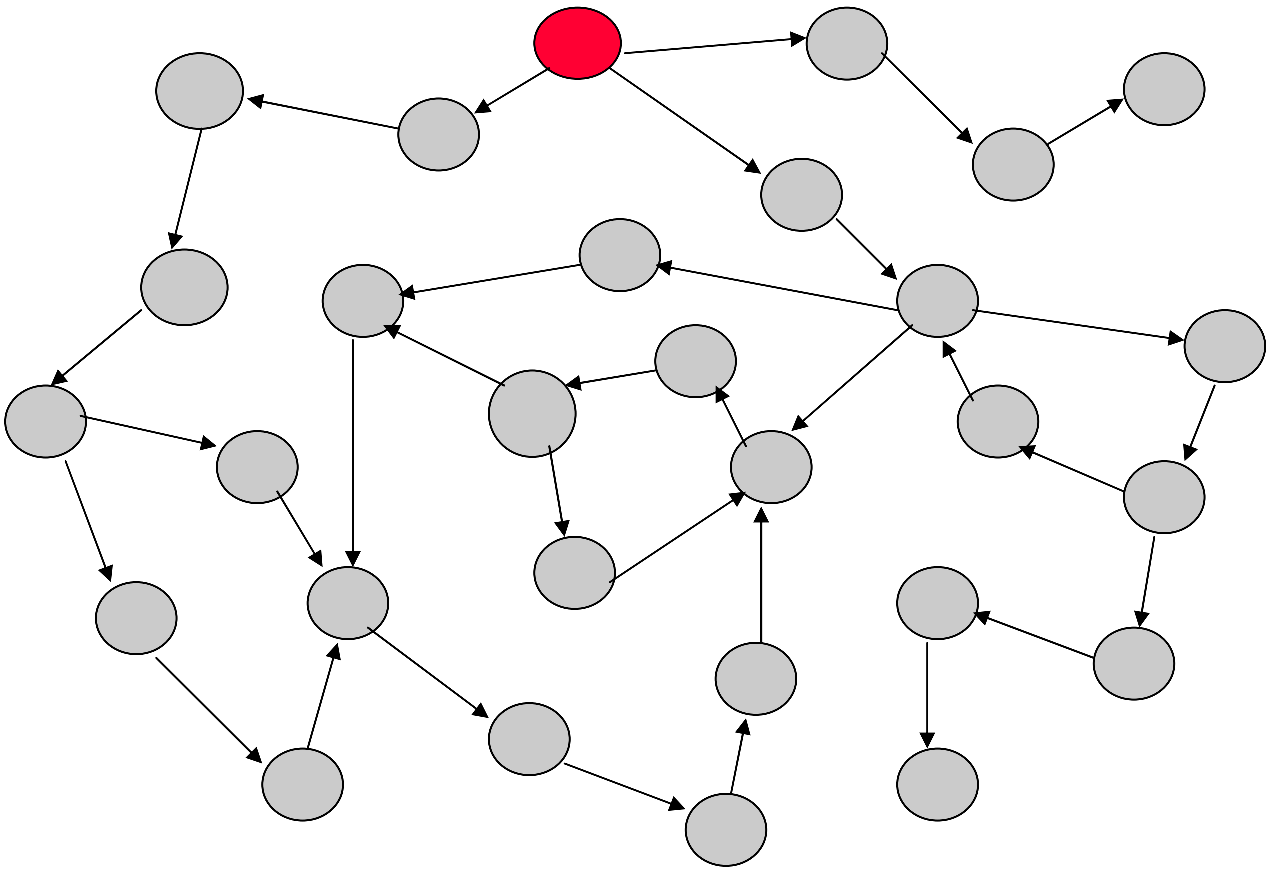
BFS With Snapshots

- State Space Explosion due to growth of *Closed* set
- Restricting growth will lead to partial failure of duplicate detection
- -> Result is LTS with redundancy

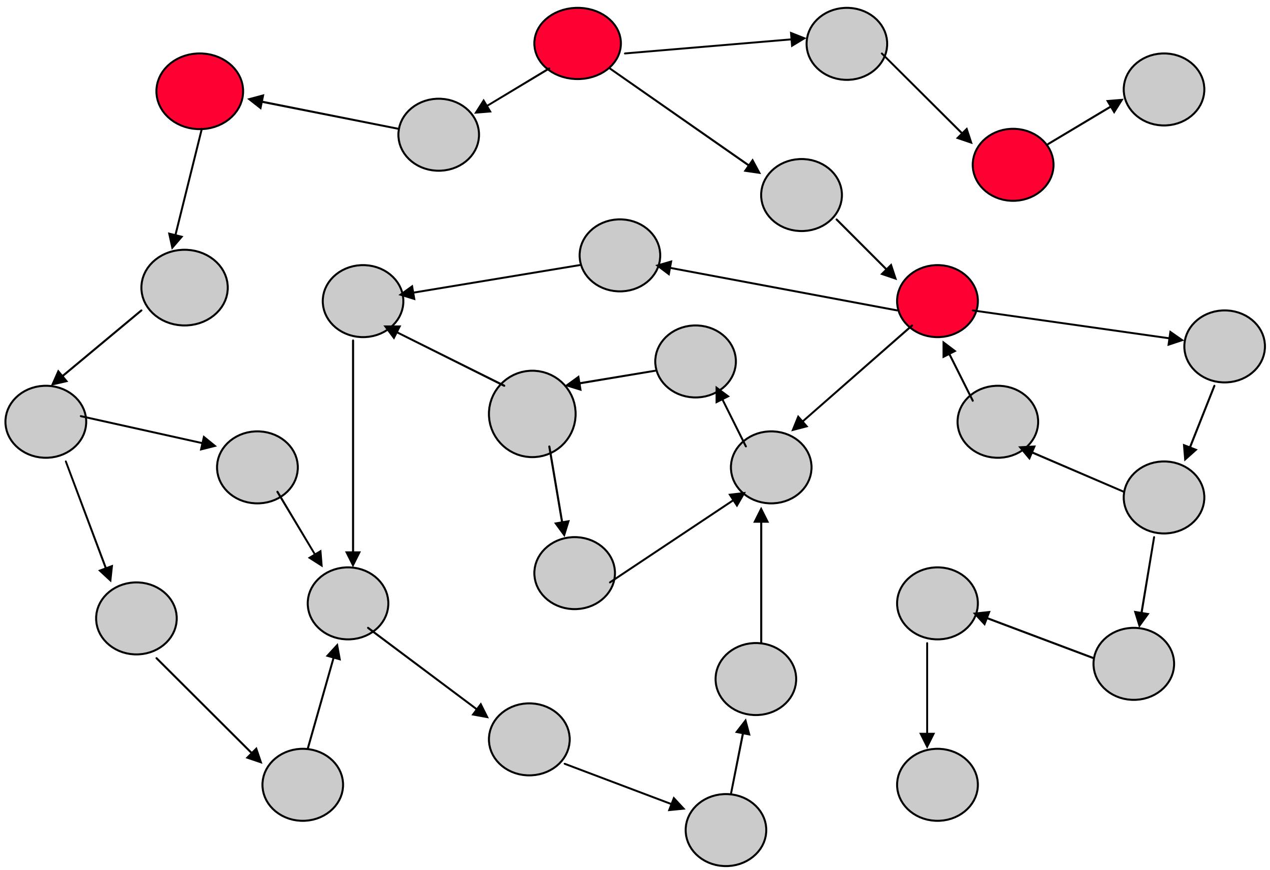
What to Store?

- No a priori knowledge structure LTS
  - Attempts at predicting exist; but theoretically, “anything can happen”
- Termination should be guaranteed
- -> We sample levels as *snapshots*
- Levels should be stored completely, otherwise states may “slip through”

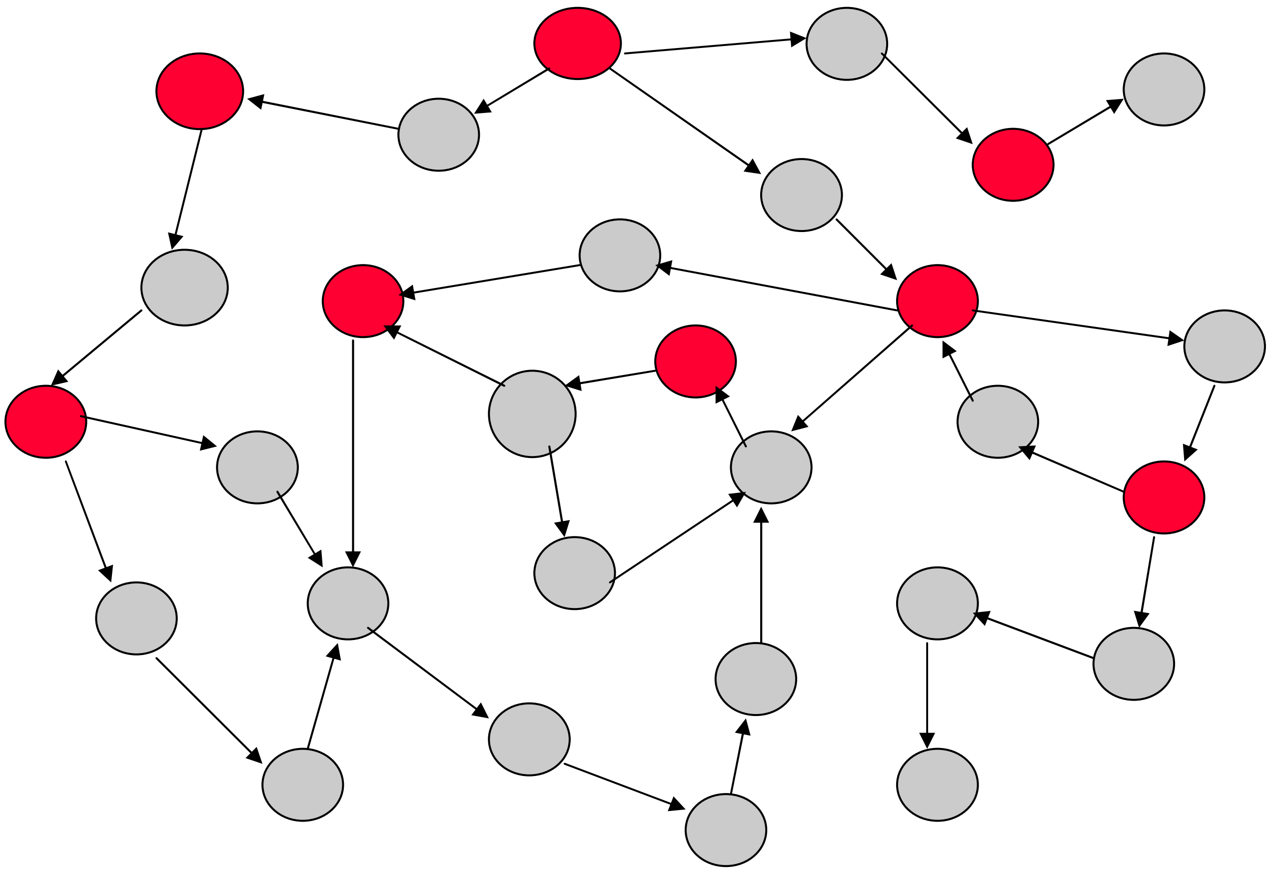
BFS With Snapshots (period=2)



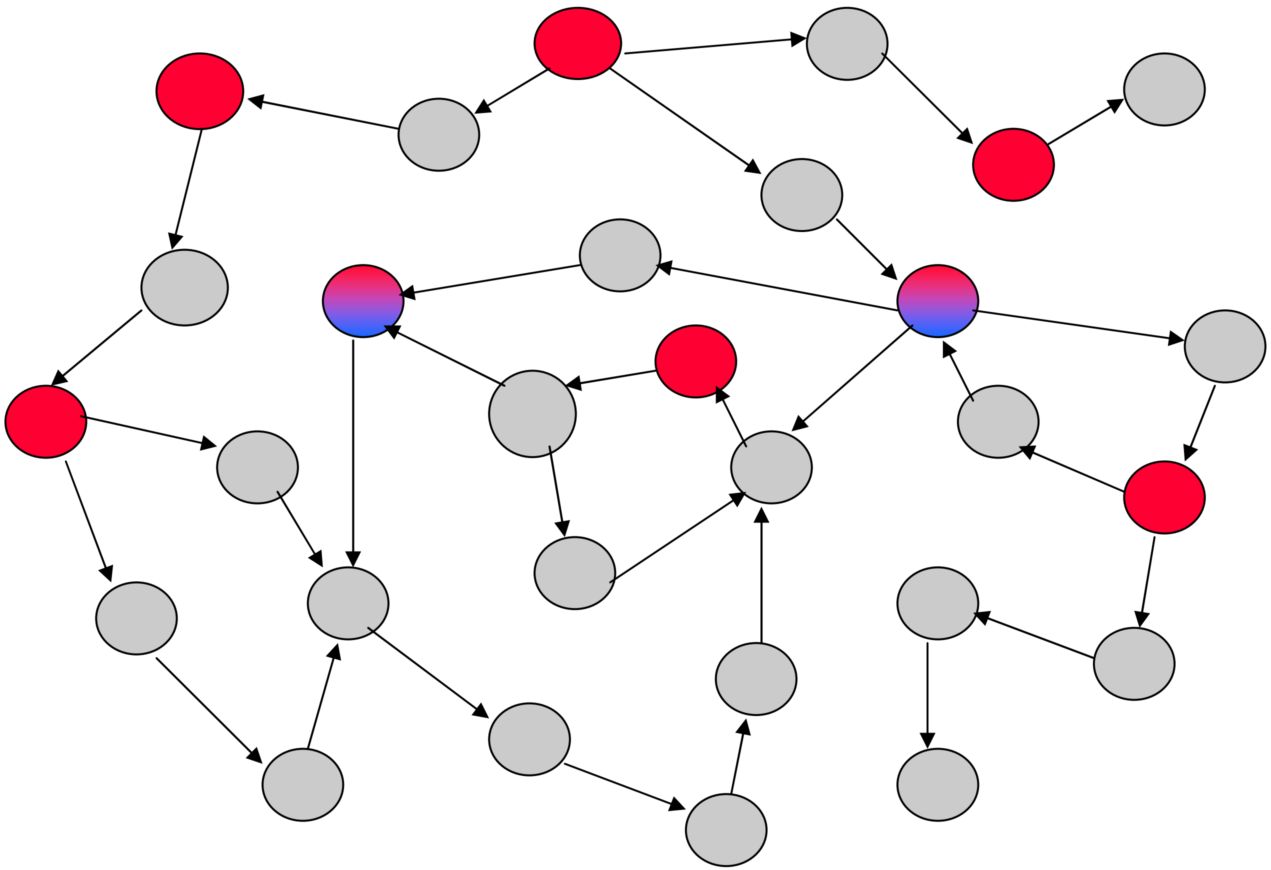
BFS With Snapshots (period=2)



BFS With Snapshots (period=2)



Duplicates Detected!



BFS with Snapshots

Algorithm 2 BFS with Snapshots

Require: Sampling function $f : \mathbb{N} \rightarrow \mathbb{B}$, number of snapshots n

procedure BFSWS(s_0)

$i, j \leftarrow 0$, $Open \leftarrow \{s_0\}$, $S_0, \ldots, S_{n-1} \leftarrow \emptyset$ {Initial state added to horizon}

$S_j \leftarrow Open$ {First snapshot contains initial state}

while $Open \neq \emptyset$ **do** {Repeat until there are no more states to explore}

$i \leftarrow i+1$, $Next \leftarrow \emptyset$ {The next level $(i+1)$ is currently empty}

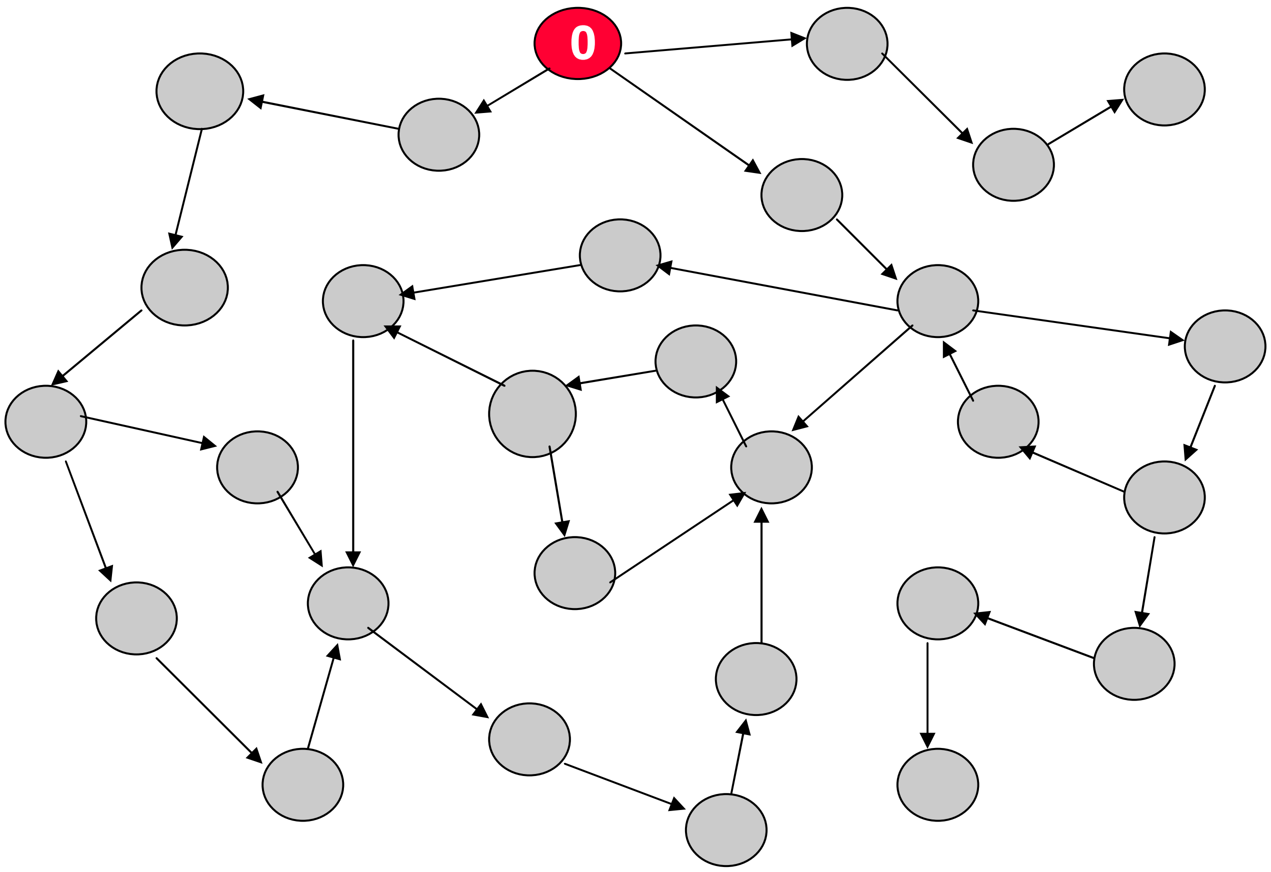
for all $s \in Open$ **do** {Explore all states in the horizon}

$Next \leftarrow Next \cup \{s' \mid \exists \ell.(s, \ell, s') \in en(s)\}$

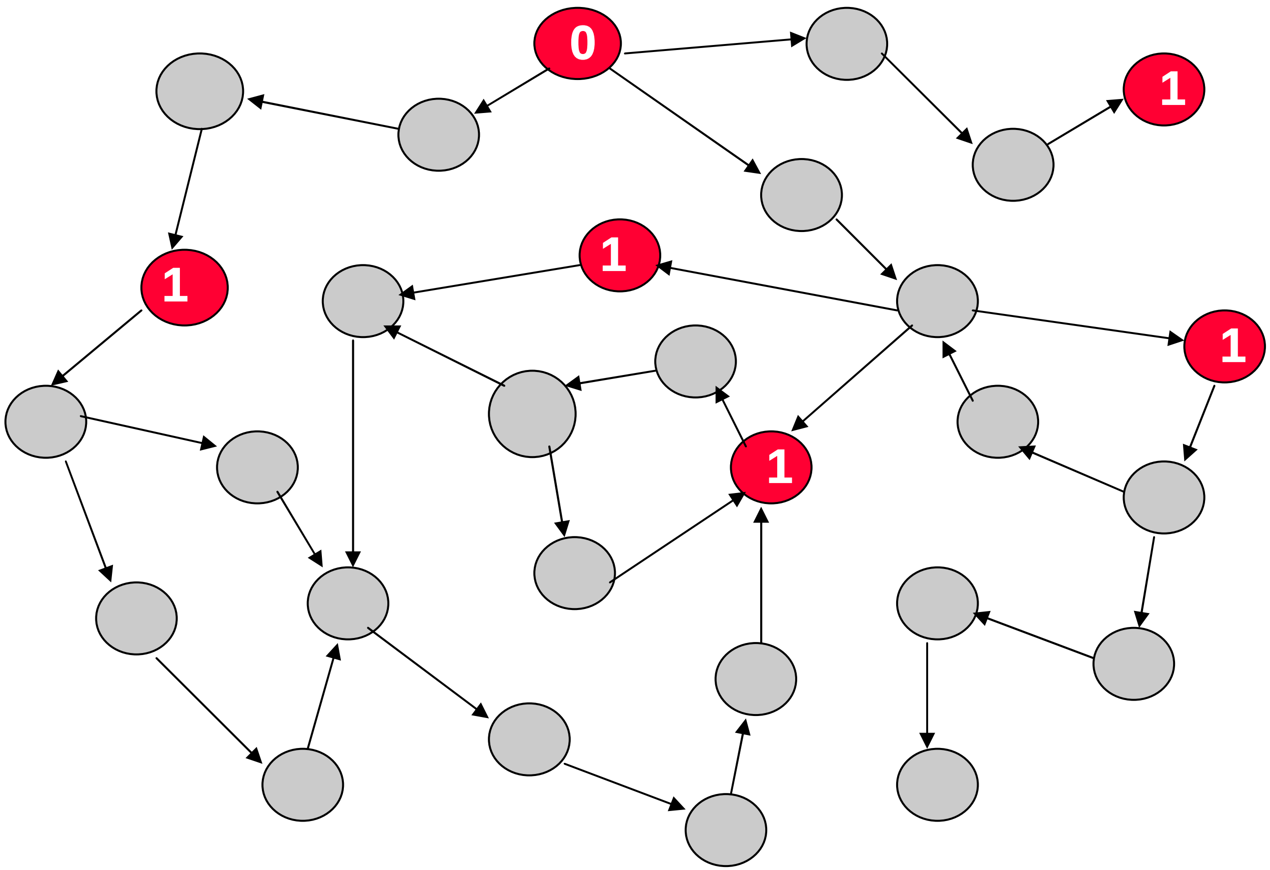
$Open \leftarrow Next \setminus \bigcup_{k=0}^{n-1} S_k$ {Add new states to horizon}

if $f(i)$ **then** $j \leftarrow j+1 \bmod n$, $S_j \leftarrow Next$ {Should this level be sampled?}

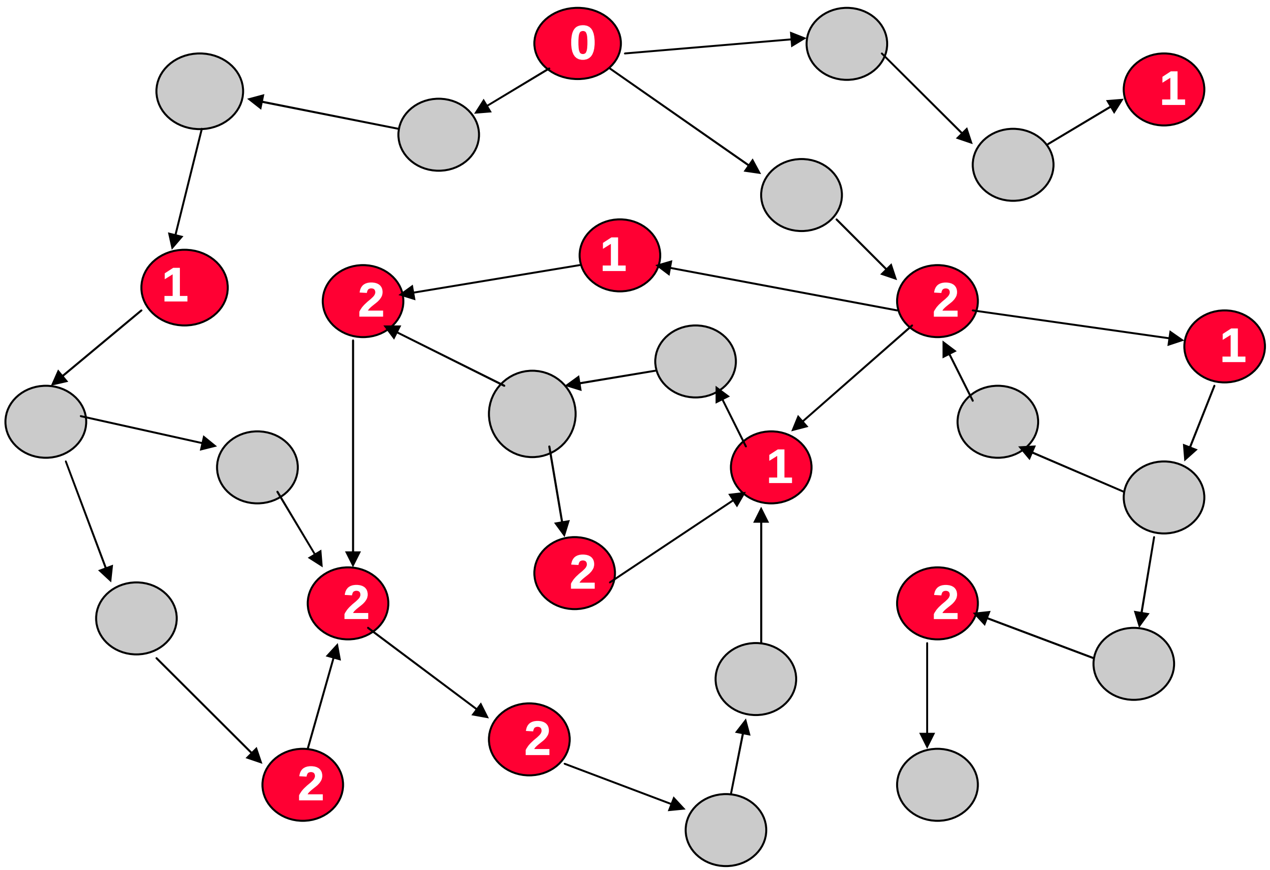
BFS With Snapshots (period=3)



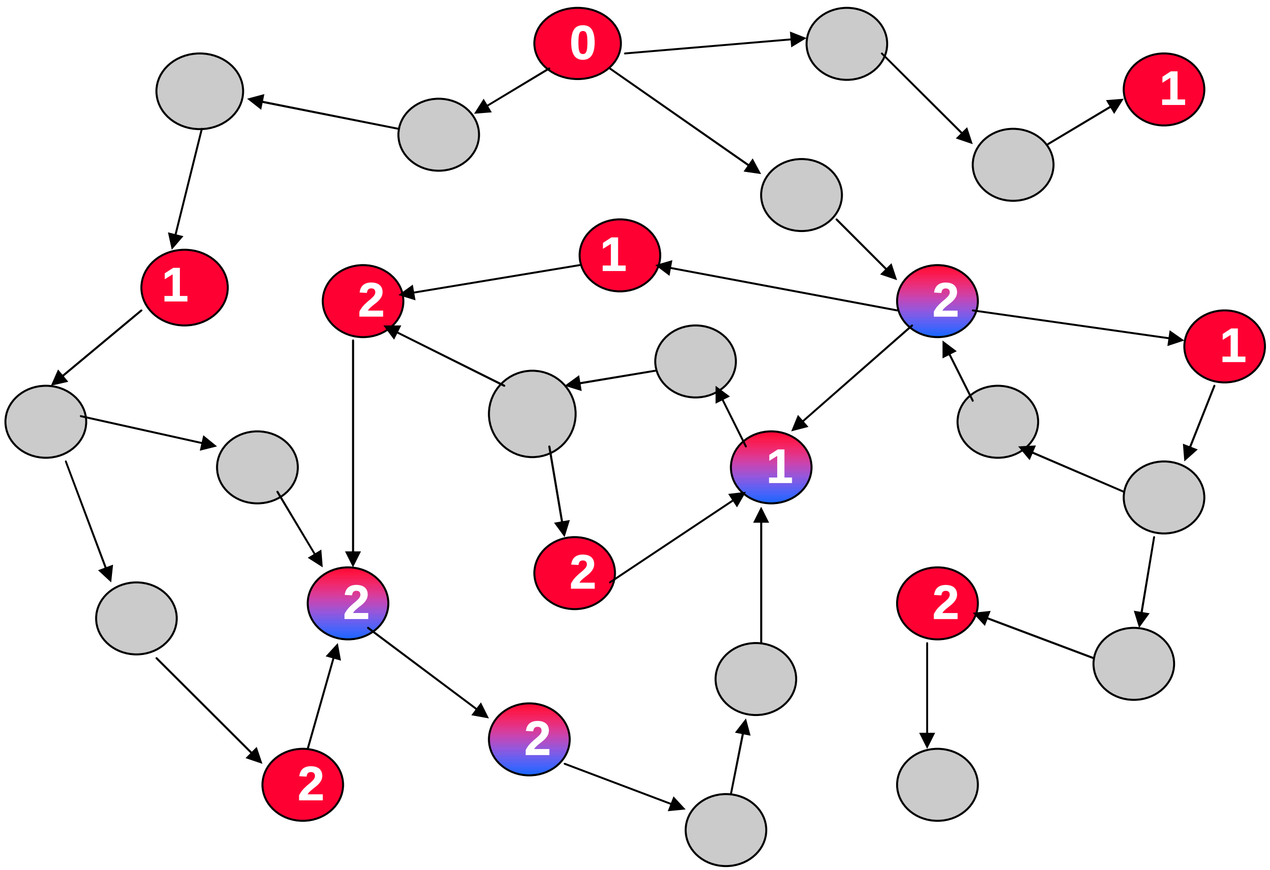
BFS With Snapshots (period=3)



BFS With Snapshots (period=3)

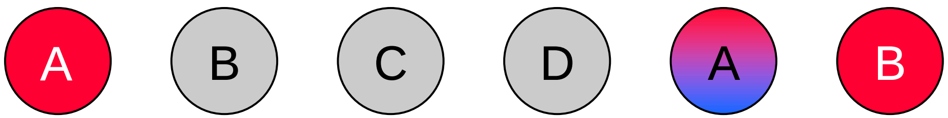


Duplicate Work Intercepted!

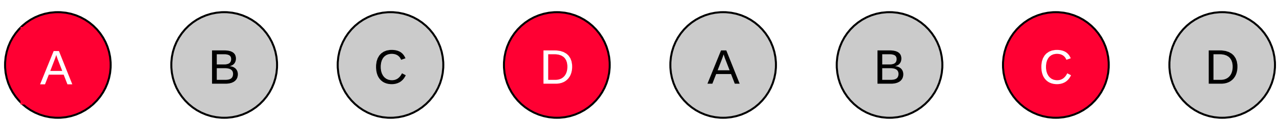


BFS With Snapshots

- With n=1, if sampling period >= size of cycle, then detect



- NO detect if sampling period < size of cycle!



- Let sampling period increase along search, then termination!

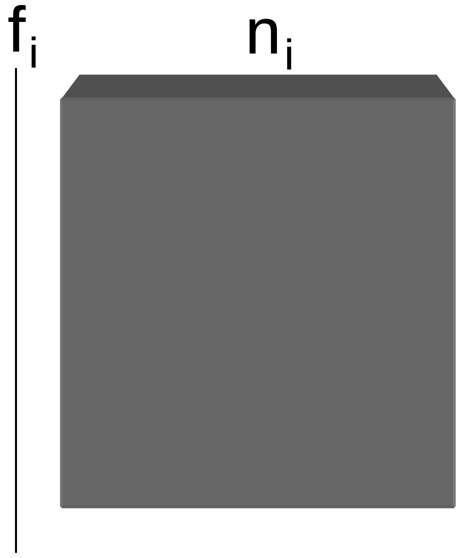
Table of Contents

- Labelled Transition System (LTS)/ Breadth-first Search (BFS)
- Partial Storage of Search History
- BFS With Snapshots
- **BFS With Snapshot Caches**
- Implementation
- Methods
- Conclusions & Future Work

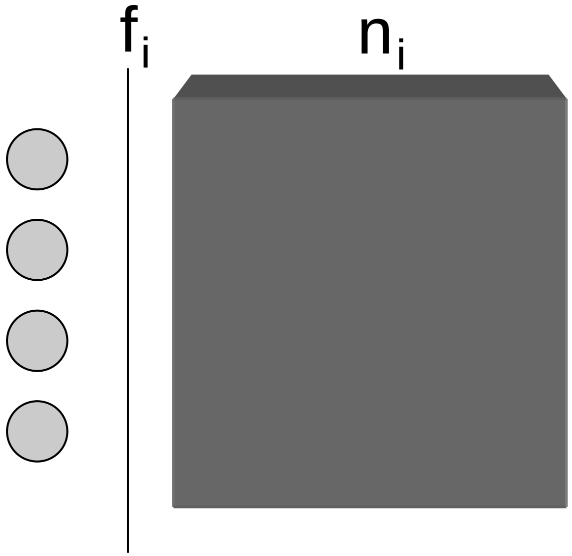
BFS With Snapshot Caches

- Similar to
  - Fast memory cache for CPU
  - Cache of web browsers
- Selectively store history in memory
  - Usually, to speed up process
  - For LTSs, to reduce memory use

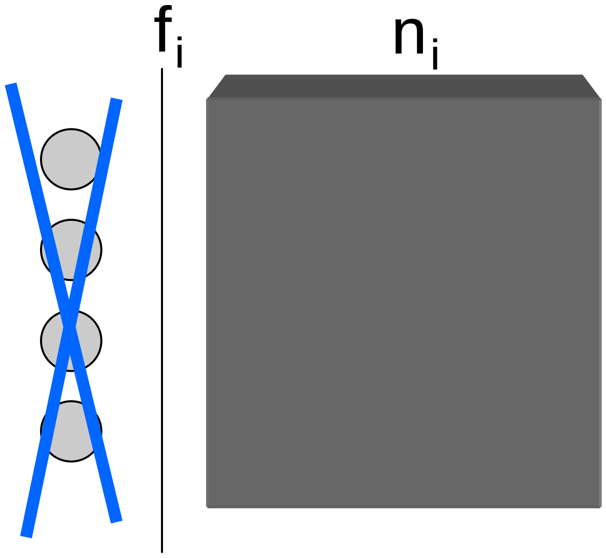
BFS With Snapshot Caches



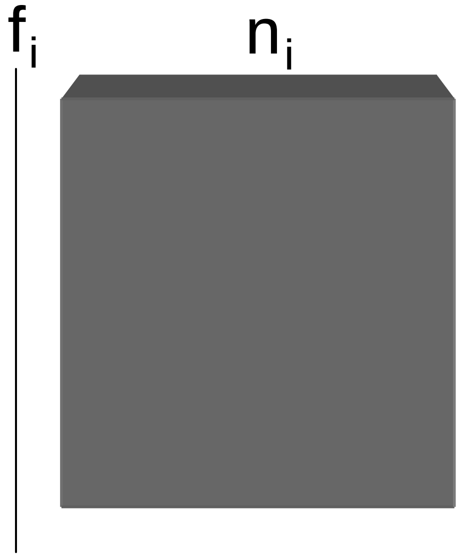
BFS With Snapshot Caches



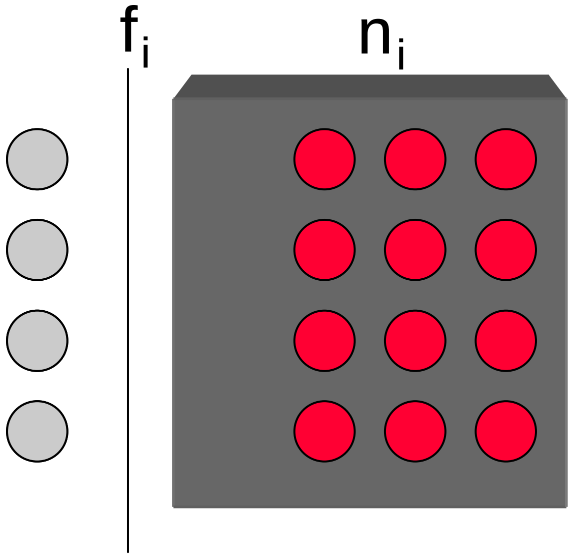
BFS With Snapshot Caches



BFS With Snapshot Caches

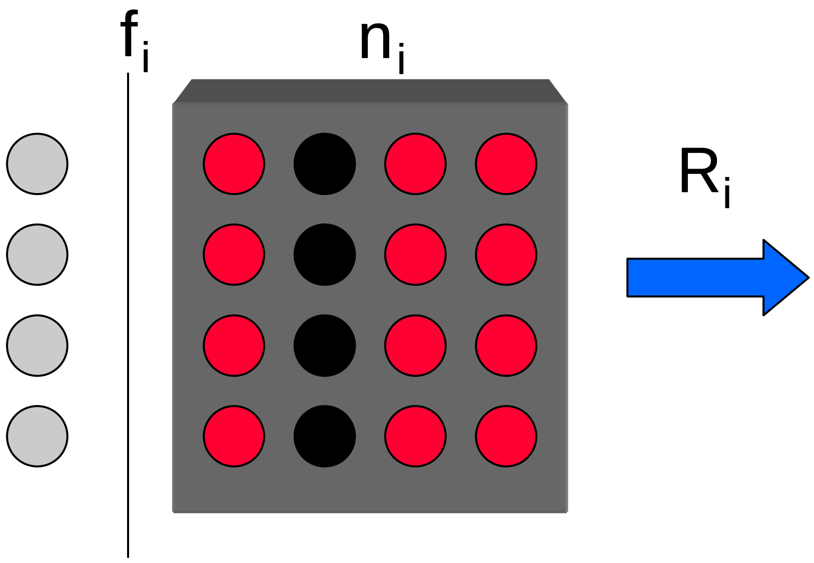


BFS With Snapshot Caches



?

BFS With Snapshot Caches



- -Least recently used
- -Most recently used
- -Least frequently used
- -Most frequently used
- -Snapshot size
- -Level number (FIFO)
- -Random

BFS With Snapshot Caches

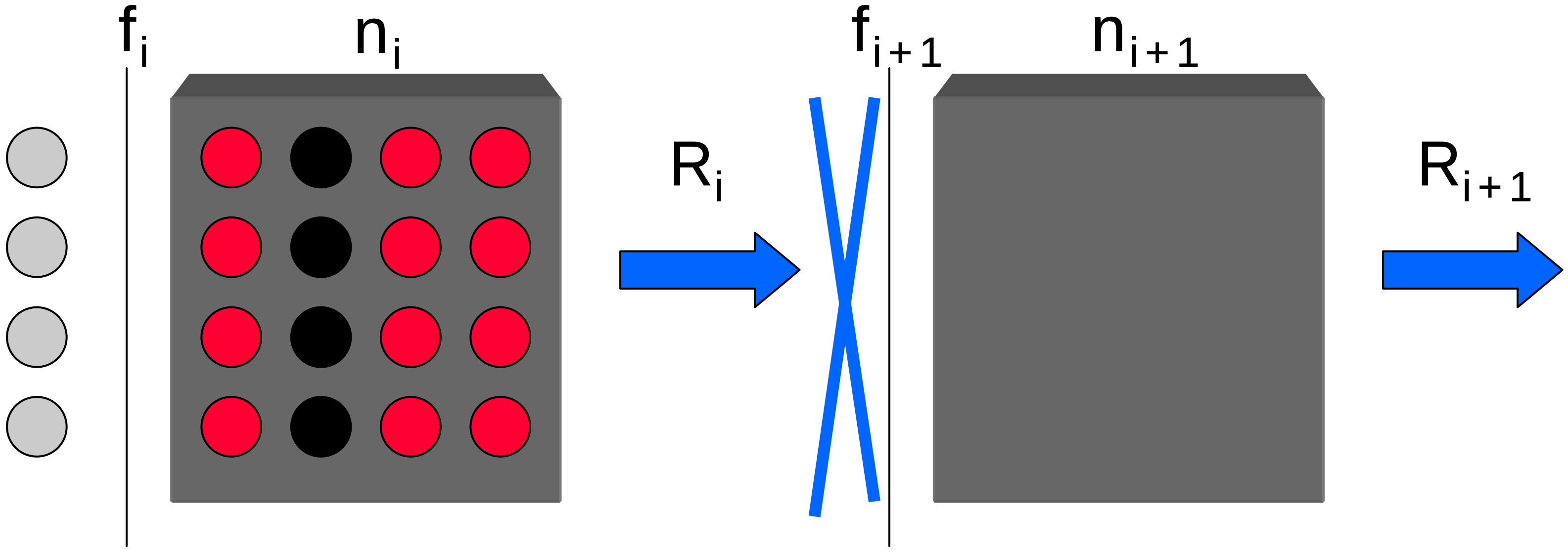


Table of Contents

- Labelled Transition System (LTS)/ Breadth-first Search (BFS)
- Partial Storage of Search History
- BFS With Snapshots
- BFS With Snapshot Caches
- **Implementation**
- Methods
- Conclusions & Future Work

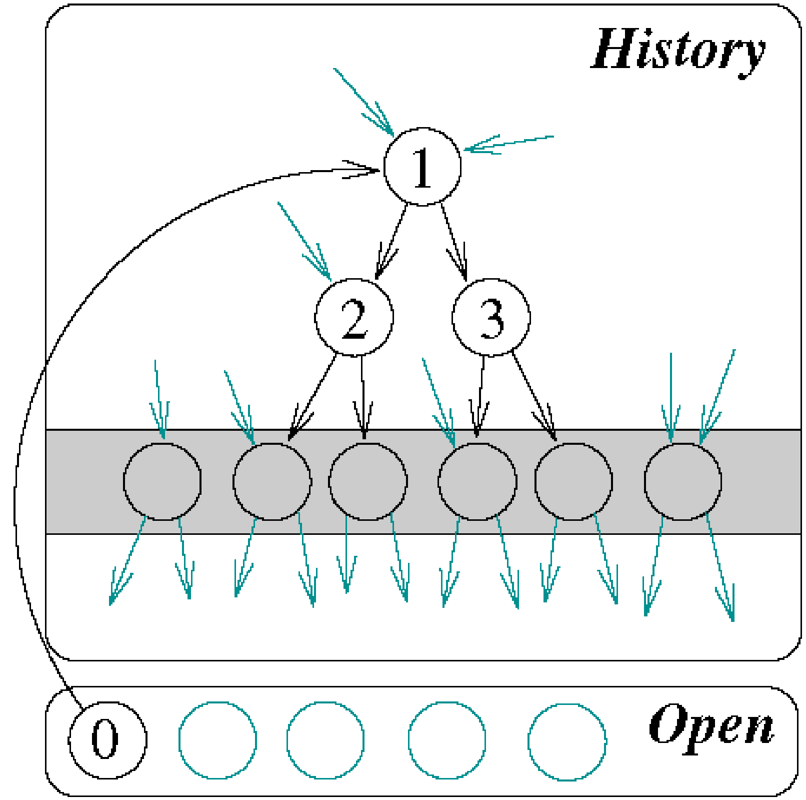
Implementation

- General Machinery in CADP
- Cache library allows creation of hierarchies of caches, each storing finite number of elements, each element paired with meta-information
- Predefined/user-specified replacement strategies
- DFS: states / BFS: levels, uniquely stored (counters)

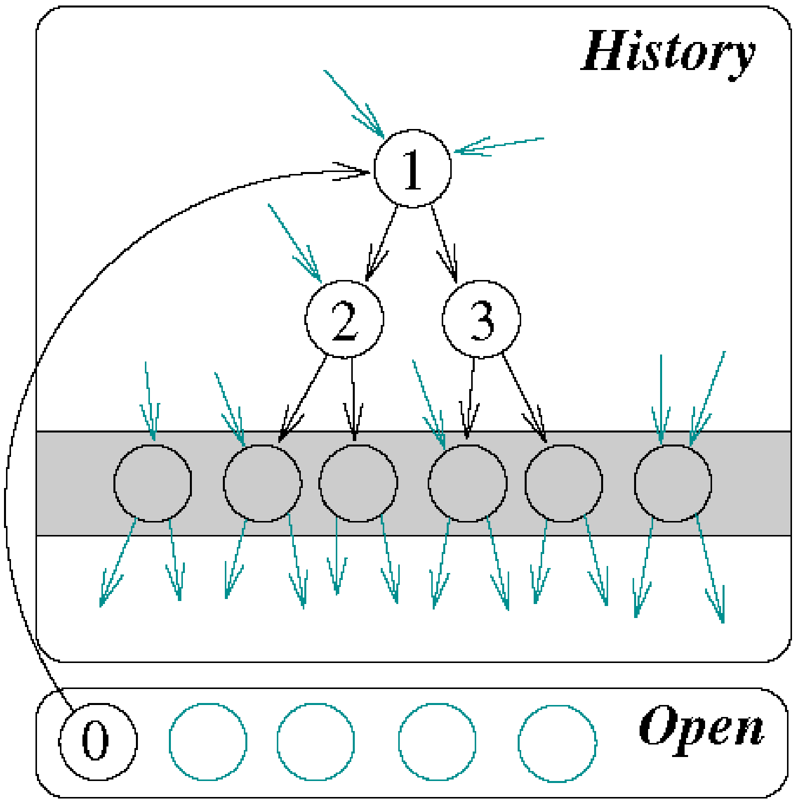
Table of Contents

Frontier Safety Net

- Earlier attempts use *locality l*; maximum jump back in search
- *Frontier Search* (Korf et al. 2005) stores last *l* levels
- Hash table with replacements (Tronci et al. 2001)
- -> Useful when *l* small (protocols)
- We can generalise, by using two caches, first FIFO frontier (period constant), second safety net (increasing period)

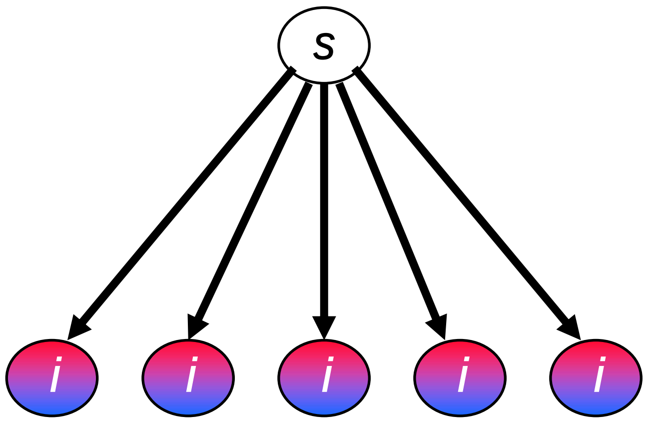


Adaptive: Backtracking



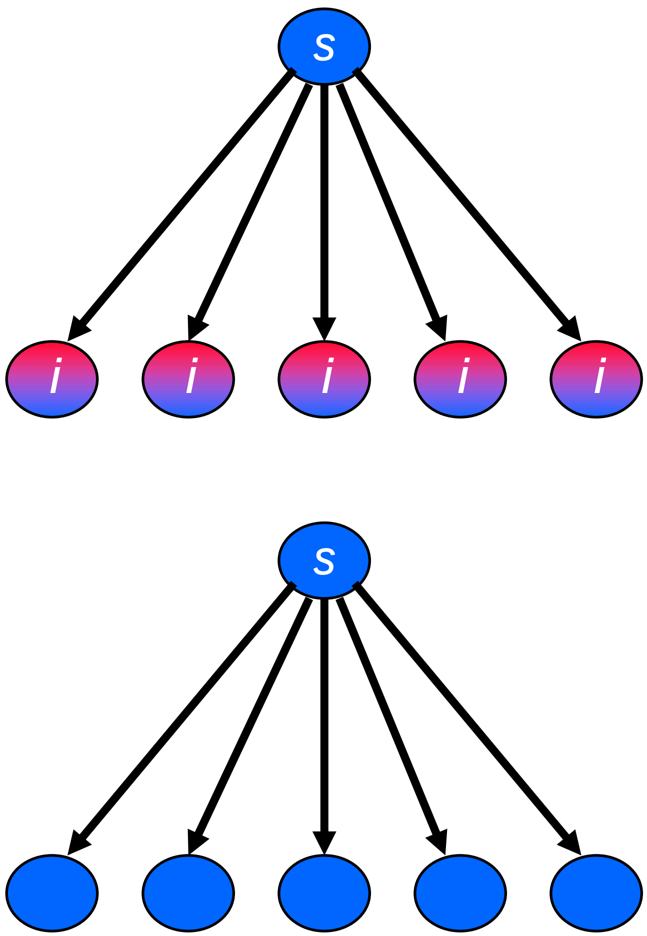
- Detection failure leads to $\Sigma_{i=1..d} b^i$ extra traversals, with b branching factor, d distance to next snapshot

Adaptive: Backtracking



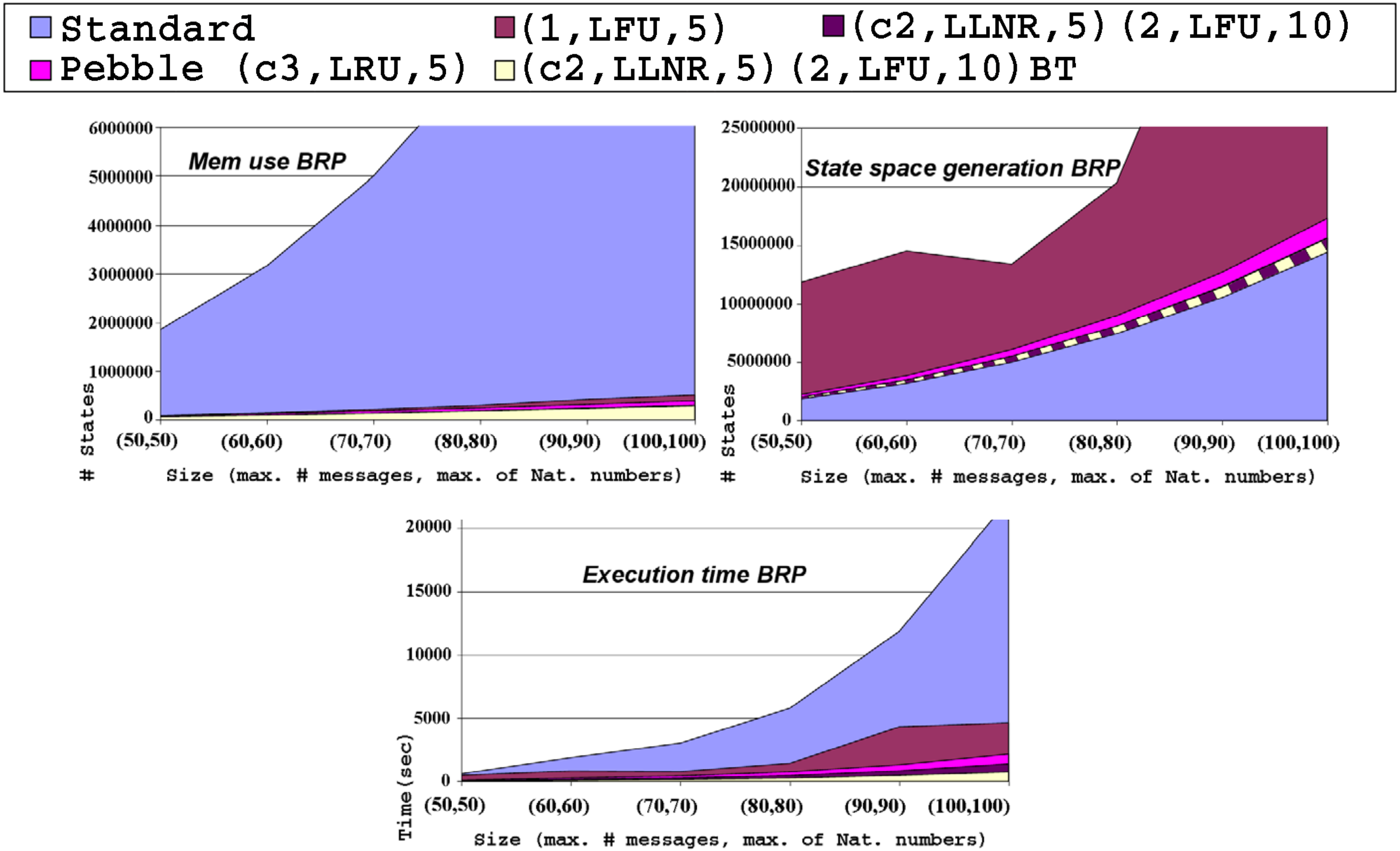
- Observation: if all *n* successors of *s* appear in a single snapshot *i*, perhaps *s* explored before (n>1)

Adaptive: Backtracking

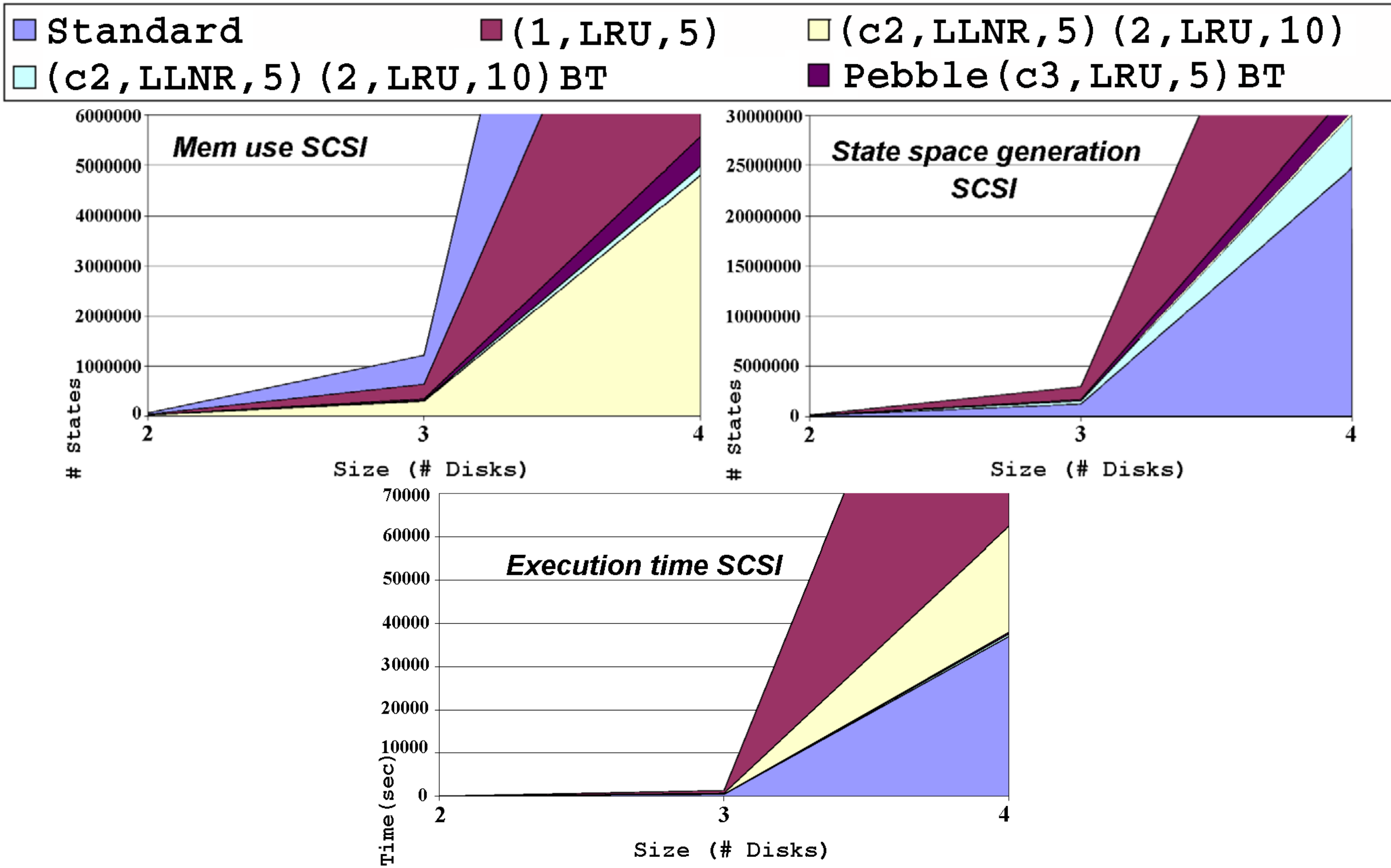


- -> Put *s* in special set of states if all succs
  - Either in 'old' snapshot *i*
    - In stream of caches C1-C2..., *n* in C*i*, *i*>1
  - Or in special set
- This set also used for duplicate detection

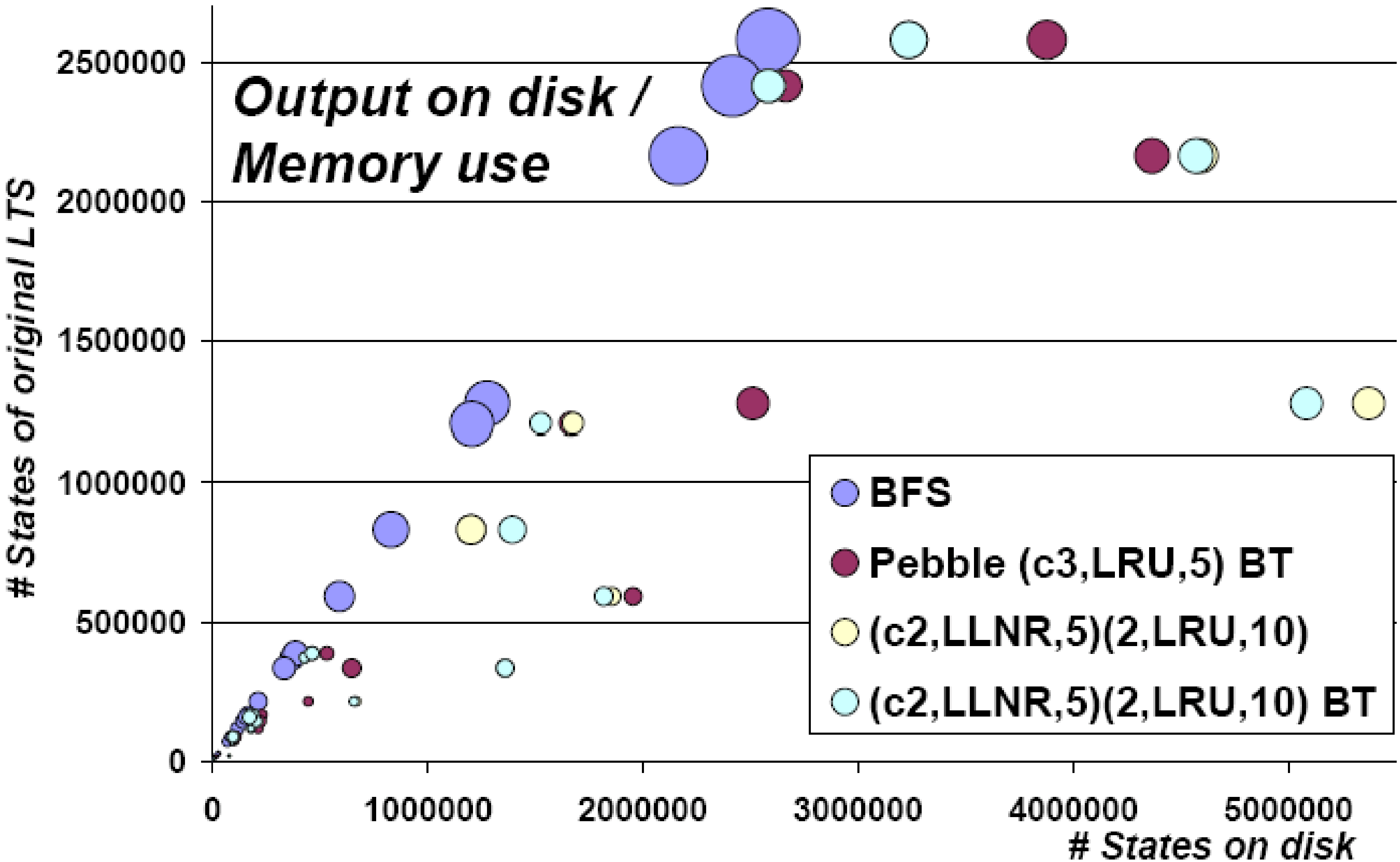
Experimental Results



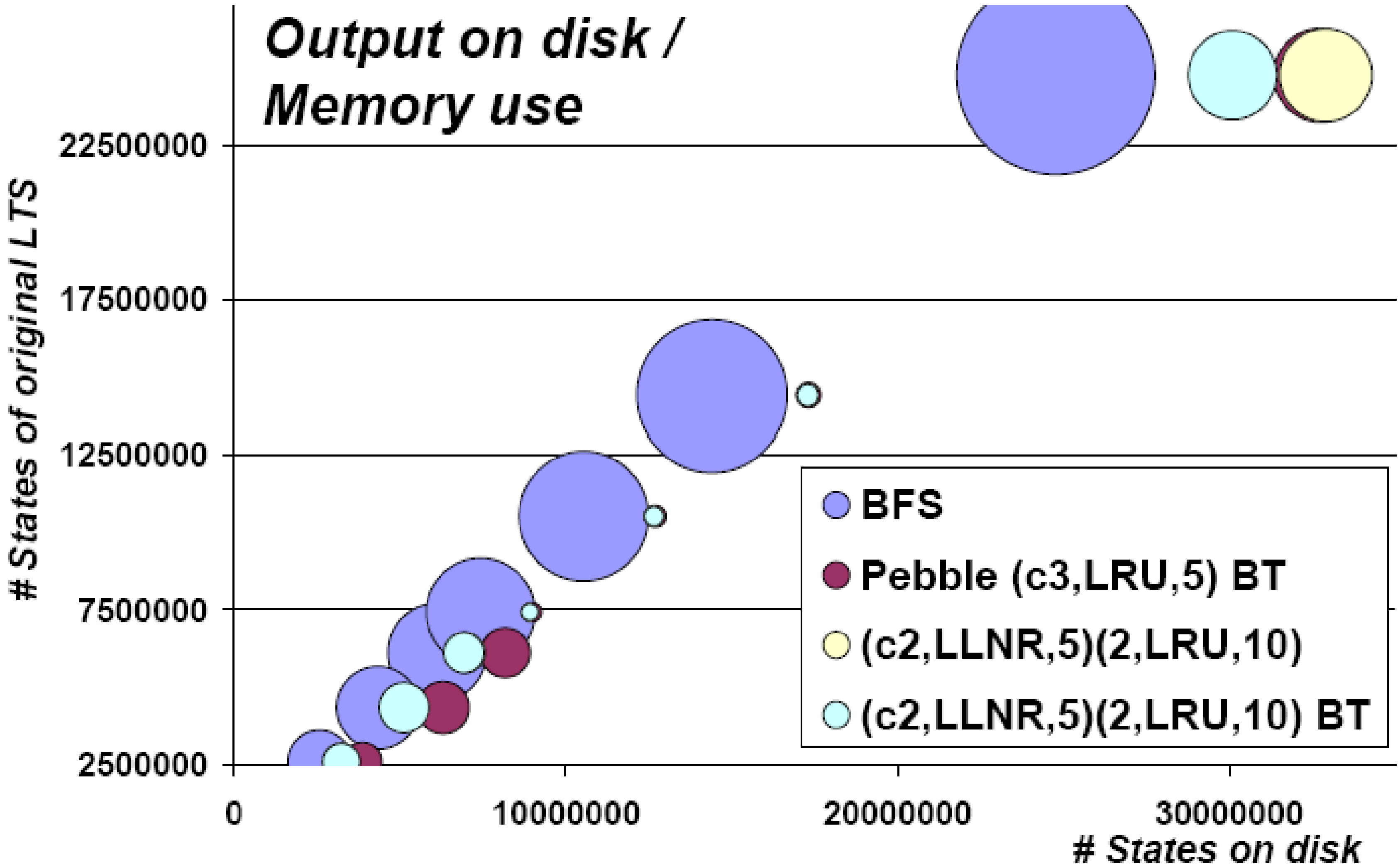
Experimental Results



Experimental Results



Experimental Results



Depth-first Search

- States in Memory
  - Stack used to store current trace
  - Caches store additional individual states
- Best strategies, size LTS < twice original
  - LRU (up to 40% reduction)
  - MFU
  - Random
- Comparable to results of Holzmann ’87
- Combination of two caches <MFU,LRU> and <RND,LRU> with LRU 25% of size, up to 30% reduction
- Overall best performance: <MFU,LRU> and <RND,LRU> with LRU 75% of size

Depth-first Search

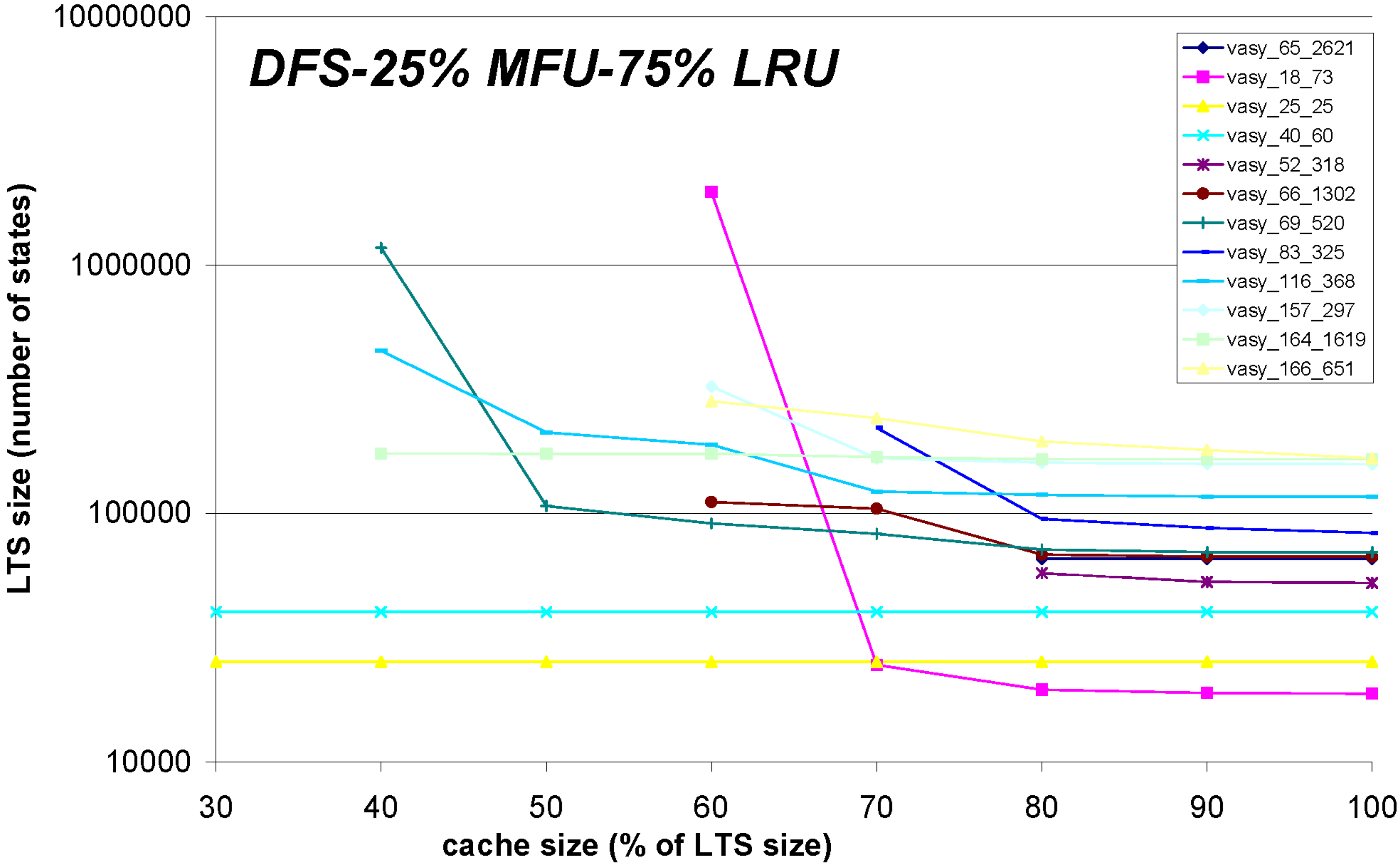


Table of Contents

- Labelled Transition System (LTS)/ Breadth-first Search (BFS)
- Partial Storage of Search History
- BFS With Snapshots
- BFS With Snapshot Caches
- Implementation
- Methods
- **Conclusions & Future Work**

Conclusions

- Delivered general machinery for hierarchical, adaptive state space caching in CADP
- The techniques are
  - Alternatives to existing methods
  - Concerned with generation
  - Guaranteeing exhaustiveness and termination
  - Useful for general LTSs
- Backtracking is an adaptive mechanism which either has a positive effect or hardly an effect -> can be used by default
- Effectiveness DFS with caching differs more from one case to the other
  - 'fixed size' cache hard to determine a priori
  - BFS with caching simply takes as much memory as needed

Future Work

- More experiments
  - Find other adaptive sampling mechanisms
  - Develop more robust setups
- Use for
  - Distributed state space generation
  - On-the-fly state space reduction

No guarantee exhaustiveness

- When hashes (positions in *Closed* list) of states collide:
  - Assume new state already visited
    - Holzmann '87, '88
    - Courcoubetis et al. '92
  - Always replace old state -> loss of termination
    - Tronci et al. 2001
- Probabilistic BFS
  - Tronci et al. 2001
- Stern & Dill '96 reduce prob. of failure

Pebble Search

- As throwing a pebble in a pool, creating waves which die out
- Creates new cache when old ones are full, due to sampling functions taking longer to be filled
- -> Rippling effect
- Use constant sampling functions
- Generalisation of Frontier Search