I/O Efficient Model Checking

Pavel Šimeček
Jiří Barnat, Luboš Brim, Michael Weber

May 19, 2008
Introduction

Model checking:
- large-scale,
- enumerative,
- temporal properties (LTL),
- finite state systems.

Possible solution: **external memory** (hard disks) utilization in model checking
Related Work

Pioneering Work

*Ulrich Stern, David L. Dill*: Using Magnetic Disk instead of Main Memory in the Murϕ Verifier (CAV’98)

LTL Model Checking

*Stefan Edelkamp, Shahid Jabbar*: Large-Scale Directed Model Checking LTL (SPIN’06)

*J. Barnat, L. Brim, P. Šimeček*: I/O Efficient Accepting Cycle Detection (CAV’07)

*J. Barnat, L. Brim, P. Šimeček, M. Weber*: Revisiting Resistance Speeds Up I/O-Efficient LTL Model Checking (TACAS’08)
Outline

- I/O efficient BFS (reachability analysis)
- I/O efficient LTL model checking
- MAP algorithm – in detail
- Merge omission – heuristic
- Analytical results
- Experimental results
Need for I/O Efficiency

- Enumerative model checking \( \approx \) graph traversal
- Default OS swapping fails:
  - An ordinary (BFS, DFS, \ldots) graph traversal does not have any locality in access to the graph
    \( \Rightarrow \) a lot of random access operations
Need for I/O Efficiency

- Enumerative model checking $\approx$ graph traversal
- Default OS swapping fails:
  - An ordinary (BFS, DFS, ... ) graph traversal does not have any locality in access to the graph
    $\Rightarrow$ a lot of random access operations
- Need to transfer data by **blocks**
I/O Complexity

- Standard two-level I/O-model with one disk (by Aggarwal and Vitter)
- Model attributes:
  - $B$ . . . block size
  - $M$ . . . main memory size
I/O Complexity

- Standard two-level I/O-model with one disk (by Aggarwal and Vitter)
- Model attributes:
  - $B$ ... block size
  - $M$ ... main memory size
- I/O operation = transfer of a data block between RAM and disk
I/O Complexity

- Standard two-level I/O-model with one disk (by Aggarwal and Vitter)
- Model attributes:
  - $B \ldots$ block size
  - $M \ldots$ main memory size
- I/O operation = transfer of a data block between RAM and disk
- I/O complexity = number of I/Os the algorithm performs
I/O Complexity

- Standard two-level I/O-model with one disk (by Aggarwal and Vitter)
- Model attributes:
  - $B$ ... block size
  - $M$ ... main memory size
- I/O operation = transfer of a data block between RAM and disk
- I/O complexity = number of I/Os the algorithm performs
- Typical operations in algorithms using disks:
  - Random access operation: $\theta(1)$
  - Linear pass through $N$ items: $\text{scan}(N) = \theta\left(\frac{N}{B}\right)$
  - Sort of $N$ items: $\text{sort}(N) = \theta\left(\frac{N}{B} \log \frac{M}{B} \frac{N}{B}\right)$
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo A)

Hard disk

Closed set:

*  *  *

Open set: (queue)

RAM

Candidate set: (part of level n & duplicates)

*  **  *
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo A)

Hard disk

Closed set:

*   *   *

Open set: (queue)

Candidate set: (part of level n & duplicates)

*   **   *   *
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo A)

- **Hard disk**
  - **Closed set:**
    - * * * *
  - **Open set:** (queue)
- **RAM**
  - **Candidate set:** (part of level n & duplicates)
    - * ** * * * ** *
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo A)

- **Closed set:**
- **Open set:** (queue)

**Candidate set:**
(part of level \( n \) & duplicates)

```
*   **   *   *   *   *   **
```
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo A)

Hard disk

Closed set:

* * * *

Open set:
(queue)

Candidate set:
(part of level n & duplicates)

RAM

COPY NEW STATES
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo A)

Hard disk

Closed set:

*   *
   *
   *   *

Open set: (queue)

RAM

Candidate set: (part of level n & duplicates)

*   **   *   *   *   *   **
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo A)

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
</table>

Hard disk

Closed set:

Open set: (queue)
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

Hard disk

Closed set:

Open set: (queue)

RAM

Candidate set: (part of level n & duplicates)
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

Hard disk

Closed set:

*  *

Open set: (queue)

RAM

Candidate set: (part of level & duplicates)

*  *
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

Hard disk

Closed set:

*   *

Open set: (queue)

RAM

Candidate set: (part of low in & duplicates)

*   *

*
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

Hard disk

Closed set:

*  *

Open set:
(queue)

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*     *     *</td>
</tr>
</tbody>
</table>
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

Hard disk

Closed set:

Open set: (queue)

merge

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>＊ ＊ ＊</td>
</tr>
</tbody>
</table>
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

Hard disk

Closed set:

Open set: (queue)

RAM

Candidate (part of level n & duplicates)

COPY NEW STATES
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

Hard disk

Closed set:

* * *

Open set: (queue)

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*     *     *</td>
</tr>
</tbody>
</table>
I/O Efficient BFS + Delayed Duplicate Detection

BFS (demo B)

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
</table>

Closed set:

Open set: (queue)
I/O Efficient BFS + Delayed Duplicate Detection

BFS

Hard disk

Closed set:

Open set: (queue)

RAM

Candidate set: (part of level n & duplicates)

$O((h_{BFS} + |E|/M) \cdot scan(|V|))$
Our Setting

- Implicitly given graph (*implicit graph*): initial state + successor function
- Advantage of implicit graphs: no disk operations performed when successors of a state needed
- (Disadvantage explicit graphs: at least $|V|$ I/O operations to explore the entire graph)
Our task: Check for existence of an accepting cycle
I/O Efficient Accepting Cycle Detection

- Naive solution – DFS-based algorithms (NDFS, DDFS)
I/O Efficient Accepting Cycle Detection

- Naive solution – DFS-based algorithms (NDFS, DDFS)
  - No existing efficient external DFS-based algorithm on implicit graphs
I/O Efficient Accepting Cycle Detection

- Naive solution – DFS-based algorithms (NDFS, DDFS)
  - No existing efficient external DFS-based algorithm on implicit graphs
- Existing solutions:
  - EJ: I/O efficient 'liveness as safety' (Edelkamp, Jabbar)
    - Reducing accepting cycle detection to reachability
    - Potentially quadratic state space growth: \( V \times F \)
  - OWCTY: I/O efficient OWCTY (Barnat, Brim, Šimeček)
    - Quite efficient in practice – especially for verification of valid properties
    - NOT on-the-fly
  - Most recent solution: MAP algorithm
I/O Efficient Accepting Cycle Detection

- Naive solution – DFS-based algorithms (NDFS, DDFS)
  - No existing efficient external DFS-based algorithm on implicit graphs
- Existing solutions:
  - **EJ**: I/O efficient ‘liveness as safety’ (Edelkamp, Jabbar)
    - Reducing accepting cycle detection to reachability
    - Potentially quadratic state space growth:
      $$V \xrightarrow{\text{reduction}} V \times F$$
    - On-the-fly
  - **OWCTY**: I/O eff. OWCTY (Barnat, Brim, ˇSimeˇcek)
    - Quite efficient in practice – especially for verification of valid properties
    - Not on-the-fly
- Most recent solution:
  - **MAP** algorithm
I/O Efficient Accepting Cycle Detection

- Naive solution – DFS-based algorithms (NDFS, DDFS)
  - No existing efficient external DFS-based algorithm on implicit graphs
- Existing solutions:
  - **EJ**: I/O efficient ‘liveness as safety’ (Edelkamp, Jabbar)
    - Reducing accepting cycle detection to reachability
    - Potentially quadratic state space growth:
      \[ V \xrightarrow{\text{reduction}} V \times F \]
    - On-the-fly
  - **OWCTY**: I/O eff. OWCTY (Barnat, Brim, Šimeček)
    - Quite efficient in practice – especially for verification of valid properties
    - NOT on-the-fly
I/O Efficient Accepting Cycle Detection

- Naive solution – DFS-based algorithms (NDFS, DDFS)
  - No existing efficient external DFS-based algorithm on implicit graphs

Existing solutions:
- **EJ:** I/O efficient ’liveness as safety’ (Edelkamp, Jabbar)
  - Reducing accepting cycle detection to reachability
  - Potentially quadratic state space growth: $V \xrightarrow{\text{reduction}} V \times F$
  - On-the-fly
- **OWCTY:** I/O eff. OWCTY (Barnat, Brim, Šimeček)
  - Quite efficient in practice – especially for verification of valid properties
  - NOT on-the-fly

Most recent solution: **MAP** algorithm
MAP – Basics

- MAP = Maximal Accepting Predecessors
- Internal memory algorithm introduced by L. Brim, I. Černá, P. Moravec and J. Šimša, FMCAD’04
MAP – Basics

- MAP = Maximal Accepting Predecessors
- Internal memory algorithm introduced by L. Brim, I. Černá, P. Moravec and J. Šimša, FMCAD’04
- Cycle detection based on labeling of vertices with their maximal accepting predecessors
MAP – Basics

- MAP = Maximal Accepting Predecessors
- Internal memory algorithm introduced by L. Brim, I. Černá, P. Moravec and J. Šimša, FMCAD’04
- Cycle detection based on labeling of vertices with their maximal accepting predecessors
- Maximum is taken with respect to arbitrary given linear ordering on accepting vertices
MAP – Basics

- MAP = Maximal Accepting Predecessors
- Internal memory algorithm introduced by L. Brim, I. Černá, P. Moravec and J. Šimša, FMCAD’04
- Cycle detection based on labeling of vertices with their maximal accepting predecessors
- Maximum is taken with respect to arbitrary given linear ordering on accepting vertices
- Advantages:
MAP – Basics

- MAP = Maximal Accepting Predecessors
- Internal memory algorithm introduced by L. Brim, I. Černá, P. Moravec and J. Šimša, FMCAD’04
- Cycle detection based on labeling of vertices with their maximal accepting predecessors
- Maximum is taken with respect to arbitrary given linear ordering on accepting vertices
- Advantages:
  - BFS-based
MAP = Maximal Accepting Predecessors

Internal memory algorithm introduced by L. Brim, I. Černá, P. Moravec and J. Šimša, FMCAD’04

Cycle detection based on labeling of vertices with their maximal accepting predecessors

Maximum is taken with respect to arbitrary given linear ordering on accepting vertices

Advantages:
- BFS-based
- On-the-fly
MAP Demostration
MAP Demonstration

> > (> ○ )
MAP Demonstration

1st iteration

> > (> ○ )
MAP Demonstration

1st iteration

> □ □ □ ( □ □ □ )
MAP Demostration

1st iteration
MAP Demostration

1st iteration
MAP Demostration

1st iteration

\( > > \)
MAP Demonstration

1st iteration

> > (>)
MAP Demonstration

1st iteration

> > (> ○ )
MAP Demonstration

\[ \textcolor{red}{>} \quad \textcolor{blue}{>} \quad \textcolor{green}{(>\,\textcolor{white}{\quad} \textcolor{white}{\quad} \textcolor{white}{\quad})} \]

Diagram with nodes and directed edges.
MAP Demostration

2nd iteration

> (> ○ )
MAP Demostration

2nd iteration

> > (> ○ )
MAP Demonstration

2nd iteration

> > (> ○ )
MAP Demonstration

2nd iteration

● > ● > ● (● ○ ●)
MAP Demonstration

2nd iteration

○ > ○ > ○ (○○○○)

12 / 18
MAP Demostration

2nd iteration

○ > ○ > ○ (> ○ )
MAP Demostration

2nd iteration

Acc. cycle!
1 iteration = I/O efficient BFS with recoloring
  recoloring = reopening of states, which obtain higher color
1 iteration = I/O efficient BFS with recoloring
- recoloring = reopening of states, which obtain higher color
- If partition is small, run internal memory NDFS on it
I/O Efficient MAP

1 iteration = I/O efficient BFS with recoloring
   recoloring = reopening of states, which obtain higher color

If partition is small, run internal memory NDFS on it

In practice: In second iteration all partitions fit in RAM
I/O Complexity Comparison

MAP:

\[ O(|F| \cdot ((d + |E|/M + |F|) \cdot scan(|V|) + sort(|V|))) \]

- \((V, E)\) ... graph
- \(F\) ... set of accepting vertices
- \(d\) ... diameter of the graph
I/O Complexity Comparison

MAP:

$O(|F| \cdot ((d + |E|/M + |F|) \cdot scan(|V|) + sort(|V|)))$

- $(V, E)$ ... graph
- $F$ ... set of accepting vertices
- $d$ ... diameter of the graph

EJ:

$O((l + |F||E|/M)scan(|F||V|))$

- $l$ ... length of the shortest counterexample
  ($l = h_{BFS}$ in case of no counterexample)
I/O Complexity Comparison

MAP:
\[ O(|F| \cdot ((d + |E|/M + |F|) \cdot \text{scan}(|V|) + \text{sort}(|V|))) \]
- \((V, E)\) ... graph
- \(F\) ... set of accepting vertices
- \(d\) ... diameter of the graph

EJ:
\[ O((l + |F||E|/M)\text{scan}(|F||V|)) \]
- \(l\) ... length of the shortest counterexample
  \((l = h_{BFS}\) in case of no counterexample\)

OWCTY:
\[ O(|l_{SCC}| \cdot (h_{BFS} + |p_{max}| + |E|/M)\text{scan}(|V|)) \]
- \(l_{SCC}\) ... the longest path in the SCC graph
- \(h_{BFS}\) ... height of BFS tree
- \(p_{max}\) ... the longest path through trivial SCCs
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level 2 &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level 2 &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of known &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*    *    *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of known &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*    *    *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*    *    *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*    *    *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

Copy new states

Copy all states (incl. duplicates)
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*    *    *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*    *    *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

**Standard Travers.**

- **Hard disk**
  - **Closed set:**
  - **Open set:** (queue)

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
</table>

**Modified Travers.**

- **Hard disk**
  - **Closed set:**
    - *
    - *
  - **Open set:** (queue)

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*     *     *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* * *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Image showing RAM content: * *]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[Image showing RAM content: * * *]</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* * * *</td>
</tr>
</tbody>
</table>
**Merge Omission and Revisiting**

**Merge omission**

**Standard Travers.**

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="Candidate set" /></td>
</tr>
</tbody>
</table>

**Modified Travers.**

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="Candidate set" /></td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

- **COPY NEW STATES**
- **COPY ALL STATES (INCL. DUPLICATES)**
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image1" alt="Standard Travers RAM" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image2" alt="Modified Travers RAM" /></td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard disk</td>
<td></td>
</tr>
<tr>
<td>Closed set:</td>
<td></td>
</tr>
<tr>
<td>Open set: (queue)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard disk</td>
<td></td>
</tr>
<tr>
<td>Closed set:</td>
<td></td>
</tr>
<tr>
<td>Open set: (queue)</td>
<td></td>
</tr>
</tbody>
</table>

* * *

* * *

* * *
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

 Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* * * * * * * * * *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

**Merge omission**

**Standard Travers.**

- **Closed set:**
  - Hard disk
  - Open set: (queue)

**Modified Travers.**

- **Closed set:**
  - Hard disk
  - Open set: (queue)

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* * * * * * * * * * * * * * * * * * * * * * * * *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>* * * * * * * * * *</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

**Standard Travers.**

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
</table>

**Modified Travers.**

<table>
<thead>
<tr>
<th>RAM</th>
<th>Candidate set: (part of level n &amp; duplicates)</th>
</tr>
</thead>
</table>
Merge Omission and Revisiting

Merge omission

<table>
<thead>
<tr>
<th>Standard Travers.</th>
<th>Modified Travers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard disk</td>
<td>Hard disk</td>
</tr>
<tr>
<td>Closed set:</td>
<td>Closed set:</td>
</tr>
<tr>
<td>Open set: (queue)</td>
<td>Open set: (queue)</td>
</tr>
<tr>
<td>RAM</td>
<td>RAM</td>
</tr>
<tr>
<td>Candidate set: (part of level n &amp; duplicates)</td>
<td>Candidate set: (part of level n &amp; duplicates)</td>
</tr>
</tbody>
</table>
Merge Omission and Revisiting

Merge omission

- **Standard Travers.**: Each state exactly **once** in **open set**
Merge Omission and Revisiting

Merge omission

Standard Travers.

Modified Travers.

- Standard Travers.: Each state exactly **once** in open set
- Modified Travers.: Some states **more times** in open set
Merge Omission and Revisiting

Merge omission

- Standard Travers.: Each state exactly **once** in open set
- Modified Travers.: Some states **more times** in open set
- But the following invariant holds in both versions:
  
  Closet set contains exactly all explored states.
Merge omission

- **Standard Travers.:** Each state exactly \textbf{once} in \textit{open set}
- **Modified Travers.:** Some states \textbf{more times} in \textit{open set}
- But the following invariant holds in both versions:
  \[ \text{Closed set contains exactly all explored states.} \]
- \[ \Rightarrow \] Heuristic omitting merge operations
Experimental Setting

- A single PC: Intel Pentium4 2 GHz (muticore), 2 GB RAM, 60 GB hard disc space available
- Implementation of the variant using RAM for delayed operations
- Comparison of external versions of MAP, OWCTY and EJ
- MAP-rr = modified version of MAP (omitting merge ops.)
Experimental Comparison

|       | $|V|$   | EJ    | OWCTY  | MAP    | MAP-rr |
|-------|--------|-------|--------|--------|--------|
| Lamport | $7 \cdot 10^7$ | OOS   | 02:37:17 | 03:16:36 | 02:37:56 |
| Peterson | $3 \cdot 10^8$ | OOS   | 18:20:03 | 25:09:35 | 15:24:29 |
| Szymanski1 | $4 \cdot 10^8$ | OOS   | 45:52:25 | 59:35:25 | 29:09:12 |

|       | $|V|$   | EJ    | OWCTY  | MAP    | MAP-rr |
|-------|--------|-------|--------|--------|--------|
| Szymanski2 | $4 \cdot 10^6$ | 00:00:50 | 00:20:07 | 00:00:04 | 00:00:02 |
| Elevator | $4 \cdot 10^4$ | 00:01:02 | 00:00:25 | 00:00:05 | 00:00:01 |
| Bakery   | $5 \cdot 10^8$ | 00:25:59 | 68:23:34 | 00:00:09 | 00:00:23 |

Times are given in hh:mm:ss format
OOS = out of hard disk space
Conclusions

- Several I/O efficient LTL model checking algorithms exist:
  - **EJ** – on-the-fly, but bad practical complexity
  - **OWCTY** – not on-the-fly, but good complexity upper-bound and fast on models with valid properties
  - **MAP** – on-the-fly, bad complexity upper-bound, fast in practice (comparable to OWCTY on valid, winner on invalid)

Performing large-scale model checking on cheap HW

Ways to do better:
- Heuristics applicable (merge omission, Bloom filters, compression)
- Better HW would bring better performance

Ongoing research – P. Šimeček, S. Edelkamp and P. Sanders: Semi-External LTL Model Checking
Conclusions

- Several I/O efficient LTL model checking algorithms exist:
  - EJ – on-the-fly, but bad practical complexity
  - OWCTY – not on-the-fly, but good complexity upper-bound and fast on models with valid properties
  - MAP – on-the-fly, bad complexity upper-bound, fast in practice (comparable to OWCTY on valid, winner on invalid)

- Performing large-scale model checking on cheap HW
Conclusions

Several I/O efficient LTL model checking algorithms exist:

- **EJ** – on-the-fly, but bad practical complexity
- **OWCTY** – not on-the-fly, but good complexity upper-bound and fast on models with valid properties
- **MAP** – on-the-fly, bad complexity upper-bound, fast in practice (comparable to OWCTY on valid, winner on invalid)

Performing large-scale model checking on cheap HW

Ways to do better:

- Heuristics applicable (merge omission, Bloom filters, compression)
- Better HW would bring better performance
Conclusions

Several I/O efficient LTL model checking algorithms exist:

- **EJ** – on-the-fly, but bad practical complexity
- **OWCTY** – not on-the-fly, but good complexity upper-bound and fast on models with valid properties
- **MAP** – on-the-fly, bad complexity upper-bound, fast in practice (comparable to OWCTY on valid, winner on invalid)

Performing large-scale model checking on cheap HW

Ways to do better:

- Heuristics applicable (merge omission, Bloom filters, compression)
- Better HW would bring better performance

Ongoing research – P. Šimeček, S. Edelkamp and P. Sanders: Semi-External LTL Model Checking