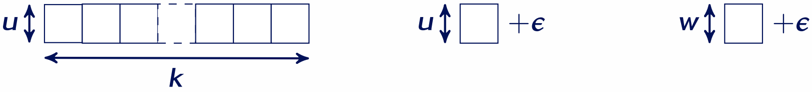
Optimal Storage of Combinatorial State Spaces

Alfons Laarman (alfons@laarman.com)
Theory Group, LIACS

Universiteit Leiden
The Netherlands

Optimal Compression



Uncompressed Information Theory This paper

Model Checking

Enumerative Model Checking

Example (Peterson mutex protocol)

```
1   bool flag[2] = {0,0};
2   bool turn = 0;
```

```
1   flag[0] = 1;                 ||   1   flag[1] = 1;
2   turn = 1;                    ||   2   turn = 0;
3   !flag[1] || turn==0;         ||   3   !flag[0] || turn==1;
    /* critical section */       ||       /* critical section */
4   flag[0] = 0; goto 1;         ||   4   flag[1] = 0; goto 1;
```

Transition System (S, I, δ)

Enumerative Model Checking

Example (Peterson mutex protocol)

```
1    bool flag[2] = {0,0};
2    bool turn = 0;
```

Process 1 ∥ Process 2:

```
1    flag[0] = 1;
2    turn = 1;
3    !flag[1] || turn==0;
     /* critical section */
4    flag[0] = 0; goto 1;
```

```
1    flag[1] = 1;
2    turn = 0;
3    !flag[0] || turn==1;
     /* critical section */
4    flag[1] = 0; goto 1;
```

Transition System (S, I, δ)

- S : $\langle pc_1, pc_2, flag[0], flag[1], turn \rangle$

Enumerative Model Checking

Example (Peterson mutex protocol)

```
1   bool flag[2] = {0,0};
2   bool turn = 0;
```

```
1   flag[0] = 1;                    ||   1   flag[1] = 1;
2   turn = 1;                       ||   2   turn = 0;
3   !flag[1] || turn==0;            ||   3   !flag[0] || turn==1;
    /* critical section */          ||       /* critical section */
4   flag[0] = 0; goto 1;            ||   4   flag[1] = 0; goto 1;
```

Transition System (S, I, δ)

- S : $\langle pc_1, pc_2, flag[0], flag[1], turn \rangle$
- $I := \{\langle 1, 1, 0, 0, 0 \rangle\}$

Enumerative Model Checking

Example (Peterson mutex protocol)

```
1    bool flag[2] = {0,0};
2    bool turn = 0;
```

```
1    flag[0] = 1;
2    turn = 1;
3    !flag[1] || turn==0;
     /* critical section */
4    flag[0] = 0; goto 1;
```

||

```
1    flag[1] = 1;
2    turn = 0;
3    !flag[0] || turn==1;
     /* critical section */
4    flag[1] = 0; goto 1;
```

Transition System (S, I, δ)

- S : $\langle pc_1, pc_2, flag[0], flag[1], turn \rangle$
- $I := \{\langle 1,1,0,0,0 \rangle\}$
- Defining δ:
  - $\delta(\langle 1,1,0,0,0 \rangle) =$

Enumerative Model Checking

Example (Peterson mutex protocol)

```
bool flag[2] = {0,0};
bool turn = 0;
```

```
flag[0] = 1;
turn = 1;
!flag[1] || turn==0;
/* critical section */
flag[0] = 0; goto 1;
```

||

```
flag[1] = 1;
turn = 0;
!flag[0] || turn==1;
/* critical section */
flag[1] = 0; goto 1;
```

Transition System (S, I, δ)

- S: $\langle pc_1, pc_2, flag[0], flag[1], turn \rangle$
- $I := \{\langle 1,1,0,0,0 \rangle\}$
- Defining δ:
  - $\delta(\langle 1,1,0,0,0 \rangle) = \{\langle 2,1,1,0,0 \rangle, \langle 1,2,0,1,0 \rangle\}$
  - $\delta(\langle 2,1,1,0,0 \rangle) = \ldots$, etc

Enumerative Model Checking

$\downarrow$

$\langle 1, 1, 0, 0, 0 \rangle$

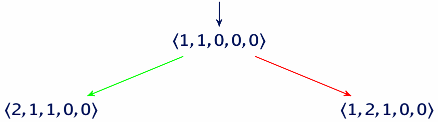
peterson($i \in \{0, 1\}$)

```
1  flag[i] = 1;
2  turn = 1 − i;
3  !flag[1 − i] || turn==i;
   /* critical section */
4  flag[i] = 0; goto 1;
```

Reachability from $I := \{\langle 1, 1, 0, 0, 0 \rangle\}$

- $S: \langle pc_1, pc_2, flag[0], flag[1], turn \rangle$

Enumerative Model Checking



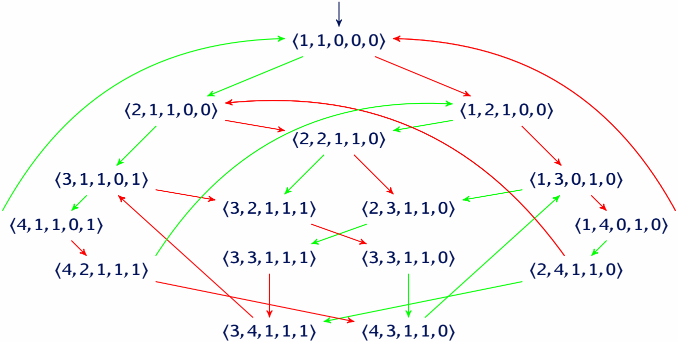
peterson($i \in \{0, 1\}$)

```
1  flag[i] = 1;
2  turn = 1 − i;
3  !flag[1 − i] || turn==i;
   /* critical section */
4  flag[i] = 0; goto 1;
```

Reachability from $I := \{\langle 1, 1, 0, 0, 0 \rangle\}$

- $S : \langle pc_1, pc_2, flag[0], flag[1], turn \rangle$

Enumerative Model Checking



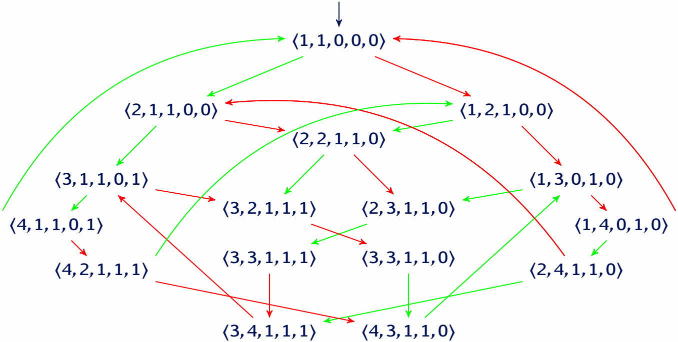
peterson$(i \in \{0,1\})$

```
1  flag[i] = 1;
2  turn = 1 − i;
3  !flag[1 − i] || turn==i;
   /* critical section */
4  flag[i] = 0; goto 1;
```

Reachability from $I := \{\langle 1,1,0,0,0 \rangle\}$

- $S : \langle pc_1, pc_2, flag[0], flag[1], turn \rangle$

Enumerative Model Checking



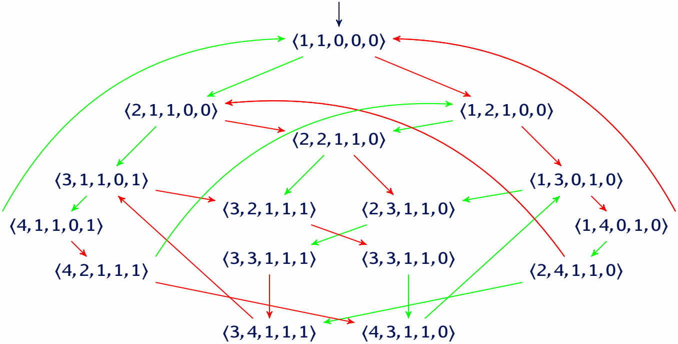
peterson$(i \in \{0,1\})$

```
1  flag[i] = 1;
2  turn = 1 − i;
3  !flag[1 − i] || turn==i;
   /* critical section */
4  flag[i] = 0; goto 1;
```

Reachability from $I := \{\langle 1,1,0,0,0\rangle\}$

- $S : \langle pc_1, pc_2, flag[0], flag[1], turn\rangle$
- Reachability from I
- Check invariant $\varphi \triangleq \neg(pc_1 = pc_2 = 4)$

Enumerative Model Checking



peterson$(i \in \{0,1\})$

```
1  flag[i] = 1;
2  turn = 1 − i;
3  !flag[1 − i] || turn==i;
   /* critical section */
4  flag[i] = 0; goto 1;
```

Reachability from $I := \{\langle 1,1,0,0,0\rangle\}$

- $S: \langle pc_1, pc_2, flag[0], flag[1], turn\rangle$
- Reachability from I
- Check invariant $\varphi \triangleq \neg(pc_1 = pc_2 = 4)$
- Also: LTL, CTL, modal μ-calculus, etc

Locality in Model Checking

Example (Peterson)

```
1  flag[i] = 1;
2  turn = 1 - i;
3  !flag[1 - i] || turn==i;
   /* critical section */
4  flag[i] = 0; goto 1;
```

1	1	0	0	0
2	1	1	0	0
3	1	1	0	1
4	1	1	0	1
1	1	0	0	1

Locality in Model Checking

Example (Peterson)

```
1  flag[i] = 1;
2  turn = 1 − i;
3  !flag[1 − i] || turn==i;
  /* critical section */
4  flag[i] = 0; goto 1;
```

1	1	0	0	0
2	1	1	0	0
3	1	1	0	1
4	1	1	0	1
1	1	0	0	1

Observations

- Locality

Locality in Model Checking

Example (Peterson)

```
1  flag[i] = 1;
2  turn = 1 - i;
3  !flag[1 - i] || turn==i;
   /* critical section */
4  flag[i] = 0; goto 1;
```

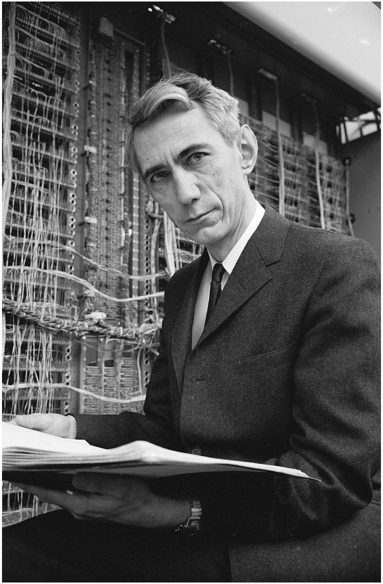
1	1	0	0	0
2	1	1	0	0
3	1	1	0	1
4	1	1	0	1
1	1	0	0	1

Observations

- Locality
- Combinatorial (Similar vectors of size $k \cdot u$ bit)

Analysis of Compression Potential

Claude Shannon (1916 — 2001)



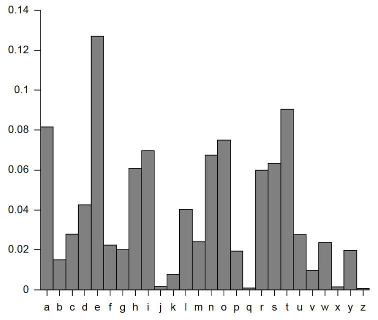
"Father of Information Theory"

Information Theory Refresher

Information Entropy

- Encoding a random English text takes $\log_2(\pm 30)$ bit/char

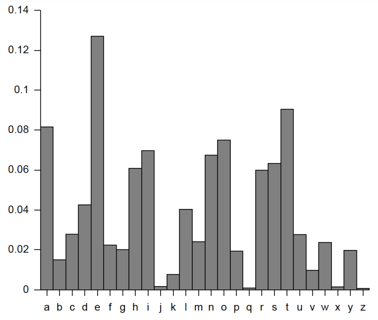
Information Theory Refresher



Information Entropy

- Encoding a random English text takes $\log_2(\pm 30)$ bit/char
- How some characters occur more frequent than others
- Idea: use shorter bit patterns to encode frequent characters

Information Theory Refresher



Information Entropy

- Encoding a random English text takes $\log_2(\pm 30)$ bit/char
- How some characters occur more frequent than others
- Idea: use shorter bit patterns to encode frequent characters
- $H(A) = -\Sigma_{\alpha \in A} p(\alpha) \log_2(p(\alpha))$ bits

Optimum Compression

$\langle 1,1,0,0,0,0\rangle \longrightarrow \langle 1,2,0,0,0,0\rangle \longrightarrow \langle 1,2,0,1,2,0\rangle \longrightarrow$

Information theoretical *lower bound*

- View states as stream: $\left\langle v_1^1, \ldots v_K^1 \right\rangle, \left\langle v_1^2, \ldots v_K^2 \right\rangle, \ldots$

Optimum Compression

$$\langle 1,1,0,0,0,0\rangle \xrightarrow{p(\frac{k-1}{k})} \langle 1,2,0,0,0,0\rangle \xrightarrow{p(\frac{1}{k})} \langle 1,2,0,1,2,0\rangle \longrightarrow$$

Information theoretical *lower bound*

- View states as stream: $\left\langle v_1^1,\dots v_K^1\right\rangle,\left\langle v_1^2,\dots v_K^2\right\rangle,\dots$
- $p(v_j^i \neq v_j^{i-1}) = \frac{1}{k}$ (*locality!*)
- $p(v_j^i = v_j^{i-1}) = \frac{k-1}{k}$

Optimum Compression

$p(\frac{k-1}{k})$ $p(\frac{1}{k})$

$\langle 1,1,0,0,0,0 \rangle$ —— $\langle 1,2,0,0,0,0 \rangle$ —— $\langle 1,2,0,1,2,0 \rangle$ ⟶

Information theoretical *lower bound*

- View states as stream: $\left\langle v_1^1, \dots v_K^1 \right\rangle, \left\langle v_1^2, \dots v_K^2 \right\rangle, \dots$
- $p(v_j^i \neq v_j^{i-1}) = \frac{1}{k}$ *(locality!)*
- $p(v_j^i = v_j^{i-1}) = \frac{k-1}{k}$
- Entropy per state $\approx u + \log_2(k) + \epsilon$ bits

Optimum Compression

$p(\frac{k-1}{k})$ $p(\frac{1}{k})$

$\langle 1,1,0,0,0,0\rangle \longrightarrow \langle 1,2,0,0,0,0\rangle \longrightarrow \langle 1,2,0,1,2,0\rangle \longrightarrow$

Information theoretical *lower bound*

- View states as stream: $\left\langle v_1^1,\dots v_K^1\right\rangle,\left\langle v_1^2,\dots v_K^2\right\rangle,\dots$
- $p(v_j^i \neq v_j^{i-1}) = \frac{1}{k}$ *(locality!)*
- $p(v_j^i = v_j^{i-1}) = \frac{k-1}{k}$
- Entropy per state $\approx u + \log_2(k) + \epsilon$ bits
- Store differences as new-value (u) × location ($\log_2(k)$):

$\langle 1,1,0,0,0,0\rangle \longrightarrow 2@1 \longrightarrow 1@3,2@4 \longrightarrow$

Optimum Compression

$p(\frac{k-1}{k})$ $p(\frac{1}{k})$

$\langle 1,1,0,0,0,0\rangle \longrightarrow \langle 1,2,0,0,0,0\rangle \longrightarrow \langle 1,2,0,1,2,0\rangle \longrightarrow$

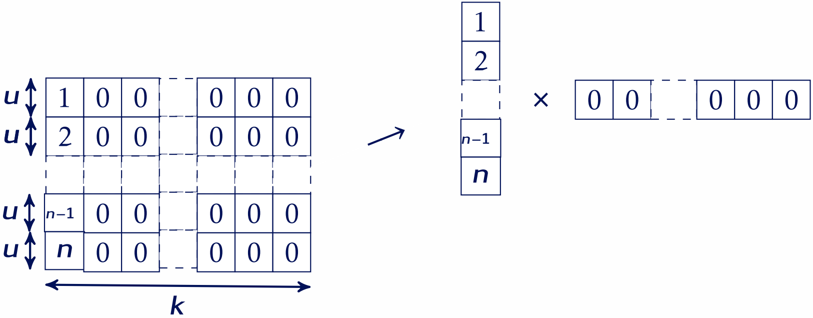
Information theoretical *lower bound*

- View states as stream: $\left\langle v_1^1,\ldots v_K^1\right\rangle,\left\langle v_1^2,\ldots v_K^2\right\rangle,\ldots$
- $p(v_j^i \neq v_j^{i-1}) = \frac{1}{k}$ (*locality!*)
- $p(v_j^i = v_j^{i-1}) = \frac{k-1}{k}$
- Entropy per state $\approx u + \log_2(k) + \epsilon$ bits
- Store differences as new-value (u) $\times$ location ($\log_2(k)$):

  $\langle 1,1,0,0,0,0\rangle \longrightarrow 2@1 \longrightarrow 1@3, 2@4 \longrightarrow$
- Yields $\mathcal{O}(n^2)$ data structure [Evangelista et al. — ATVA'13]

Tree Compression

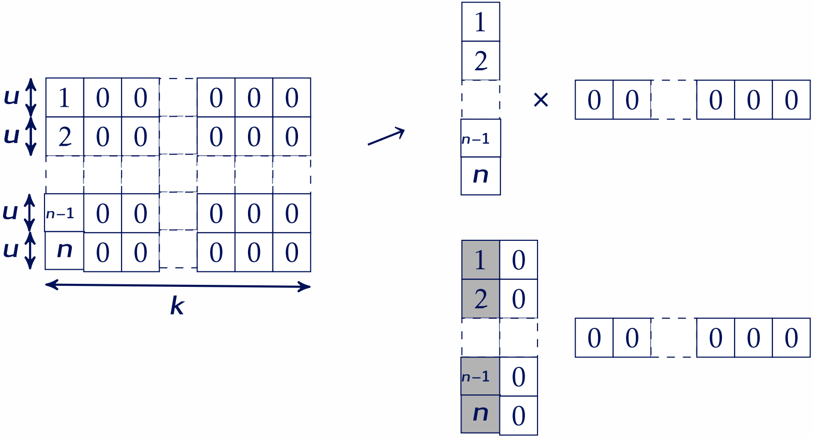
Tree Compression Idea



Best-case compression

- $\Omega(n \cdot k \cdot u)$ vs $\Omega(n \cdot u)$

Tree Compression Idea



Best-case compression

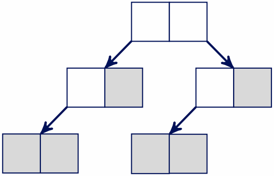
- $\Omega(n \cdot k \cdot u)$ vs $\Omega(n \cdot u)$

Tree Compression

Inserting a vector in the tree

- Fold vectors in binary tree
- Propagate hashed tuples back up

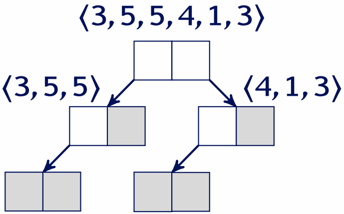
$\langle 3, 5, 5, 4, 1, 3 \rangle$



Tree Compression

Inserting a vector in the tree

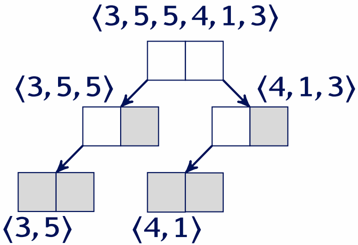
- Fold vectors in binary tree
- Propagate hashed tuples back up



Tree Compression

Inserting a vector in the tree

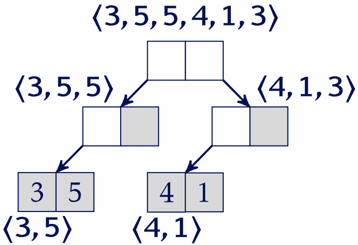
- Fold vectors in binary tree
- Propagate hashed tuples back up



Tree Compression

Inserting a vector in the tree

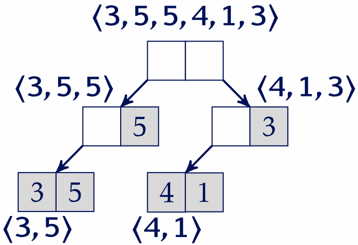
- Fold vectors in binary tree
- Propagate hashed tuples back up



Tree Compression

Inserting a vector in the tree

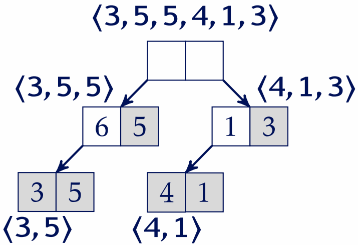
- Fold vectors in binary tree
- Propagate hashed tuples back up



Tree Compression

Inserting a vector in the tree

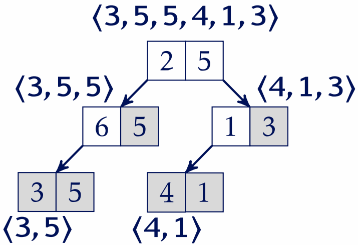
- Fold vectors in binary tree
- Propagate hashed tuples back up



Tree Compression

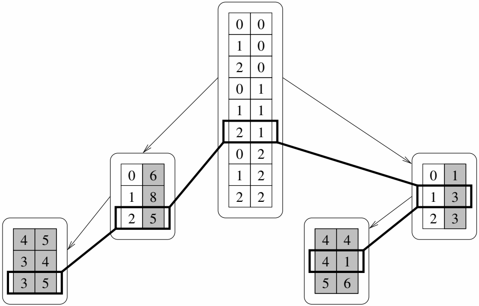
Inserting a vector in the tree

- Fold vectors in binary tree
- Propagate hashed tuples back up



Tree Compression

4	5	6	4	4	1
3	4	8	4	4	1
3	5	5	4	4	1
4	5	6	4	1	3
3	4	8	4	1	3
3	5	5	4	1	3
4	5	6	5	6	3
3	4	8	5	6	3
3	5	5	5	6	3



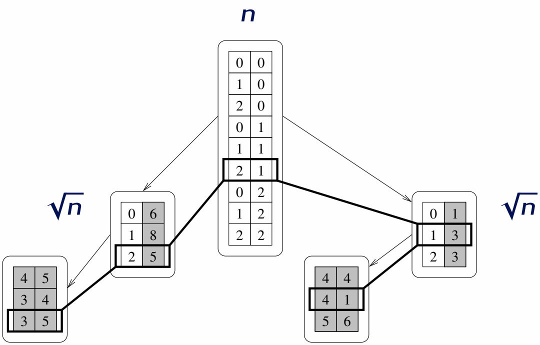
$\mathcal{O}(n \cdot k)$

$\Omega(2n \cdot (k-1))$

Tree Compression

4	5	6	4	4	1
3	4	8	4	4	1
3	5	5	4	4	1
4	5	6	4	1	3
3	4	8	4	1	3
3	5	5	4	1	3
4	5	6	5	6	3
3	4	8	5	6	3
3	5	5	5	6	3

$\mathcal{O}(n \cdot k)$

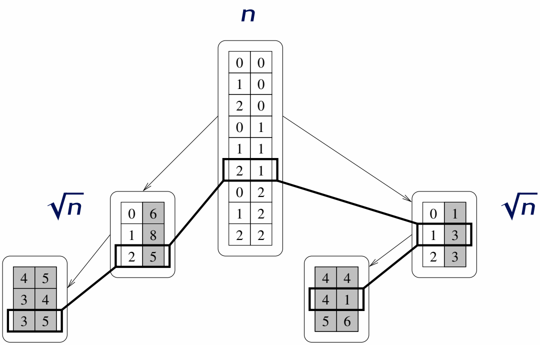


$\Omega(2n \cdot (k-1))$

Tree Compression

4	5	6	4	4	1
3	4	8	4	4	1
3	5	5	4	4	1
4	5	6	4	1	3
3	4	8	4	1	3
3	5	5	4	1	3
4	5	6	5	6	3
3	4	8	5	6	3
3	5	5	5	6	3

$\mathcal{O}(n \cdot k)$



$\Omega(2n \cdot (k-1))$

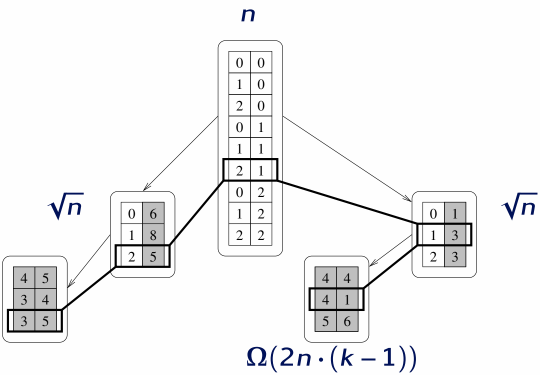
Compression

✓ Combinatorial ⇒ balanced tree ($n + 2\sqrt{n} + 4\sqrt[4]{(n)} \cdots \approx n$ tuples)

Tree Compression

4	5	6	4	4	1
3	4	8	4	4	1
3	5	5	4	4	1
4	5	6	4	1	3
3	4	8	4	1	3
3	5	5	4	1	3
4	5	6	5	6	3
3	4	8	5	6	3
3	5	5	5	6	3

$\mathcal{O}(n \cdot k)$



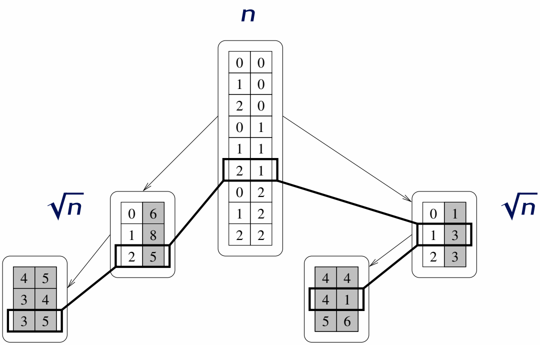
Compression

- ✓ Combinatorial ⇒ balanced tree ($n + 2\sqrt{n} + 4\sqrt[4]{(n)} \cdots \approx n$ tuples)
- ✓ Can compresses states of length k to almost $2 \cdot w$ bits! (w bits for pointers)

Tree Compression

4	5	6	4	4	1
3	4	8	4	4	1
3	5	5	4	4	1
4	5	6	4	1	3
3	4	8	4	1	3
3	5	5	4	1	3
4	5	6	5	6	3
3	4	8	5	6	3
3	5	5	5	6	3

$\mathcal{O}(n \cdot k)$



$\Omega(2n \cdot (k-1))$

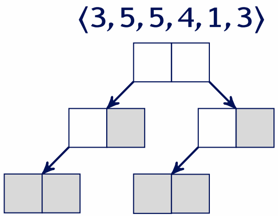
Compression

- ✓ Combinatorial ⇒ balanced tree ($n + 2\sqrt{n} + 4\sqrt[4]{(n)} \cdots \approx n$ tuples)
- ✓ Can compresses states of length k to almost $2 \cdot w$ bits! (w bits for pointers)
- ✗ What about access times?

Incremental Tree Compression
[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree

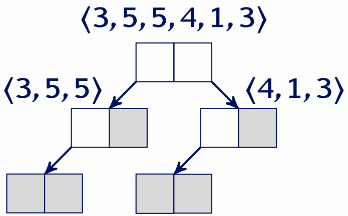


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree

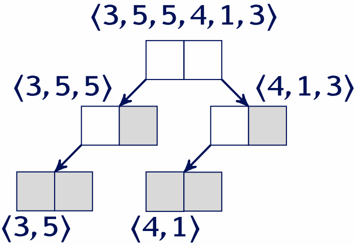


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree

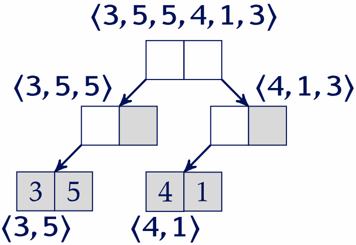


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

Incremental insertion

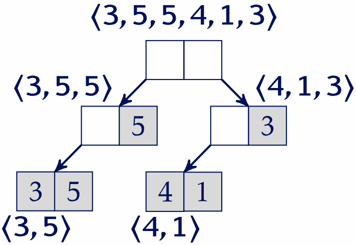
- Fold vectors in binary tree
- Propagate hashed tuples back up



Incremental Tree Compression
[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree
- Propagate hashed tuples back up

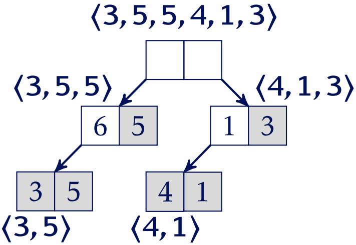


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree
- Propagate hashed tuples back up

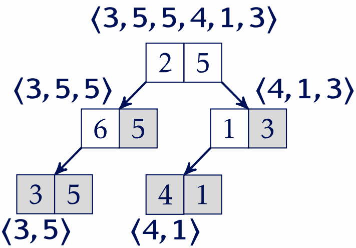


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree
- Propagate hashed tuples back up

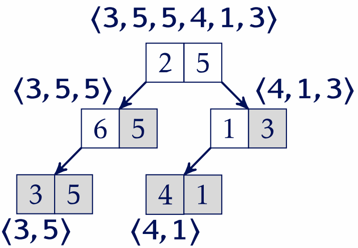


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree
- Propagate hashed tuples back up
- Incremental updates: $(K-1) \rightarrow \log_2(K-1)$ lookups

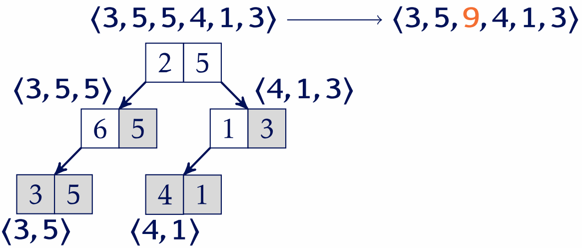


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

Incremental insertion

- Fold vectors in binary tree
- Propagate hashed tuples back up
- Incremental updates: $(K-1) \rightarrow \log_2(K-1)$ lookups

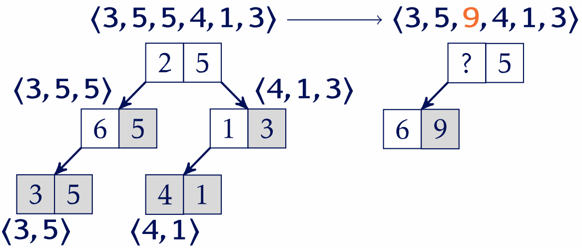


Incremental Tree Compression

[Laarman, van de Pol, Weber spin11]

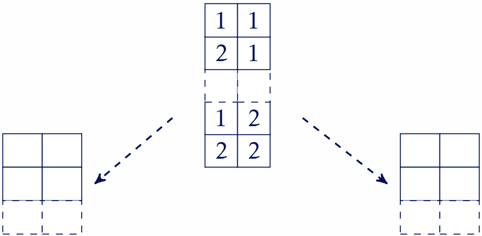
Incremental insertion

- Fold vectors in binary tree
- Propagate hashed tuples back up
- Incremental updates: $(K-1) \rightarrow \log_2(K-1)$ lookups



Compact Tree Compression

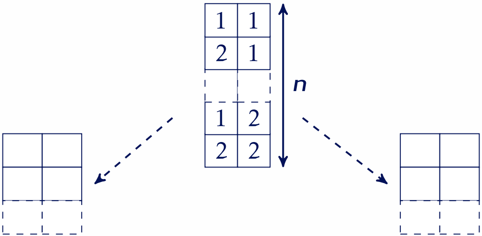
Compact Tree Idea



Idea: store the universe of tuples in tree compactly

- $\log_2(|U|) = 2w$ (internal pointers)

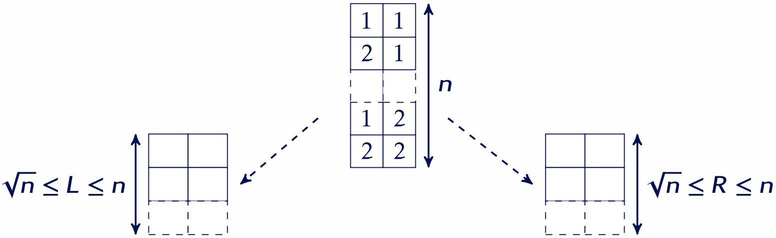
Compact Tree Idea



Idea: store the universe of tuples in tree compactly

- $\log_2(|U|) = 2w$ (internal pointers)

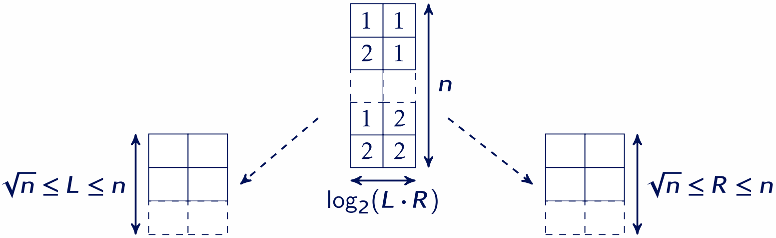
Compact Tree Idea



Idea: store the universe of tuples in tree compactly

- $\log_2(|U|) = 2w$ (internal pointers)

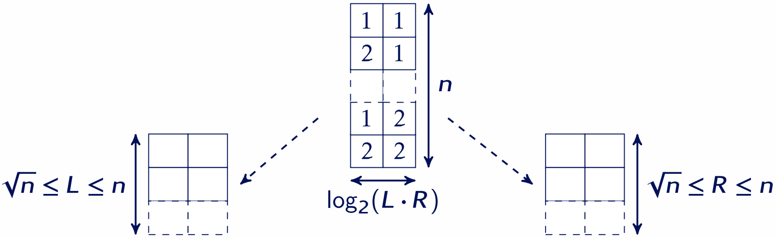
Compact Tree Idea



Idea: store the universe of tuples in tree compactly

- $\log_2(|U|) = 2w$ (internal pointers)

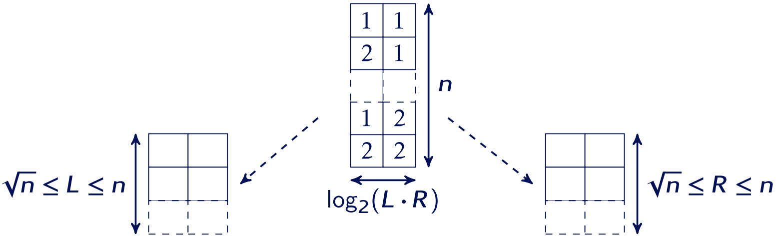
Compact Tree Idea



Idea: store the universe of tuples in tree compactly

- $\log_2(|U|) = 2w$ (internal pointers)
- $|U| = L \cdot R$

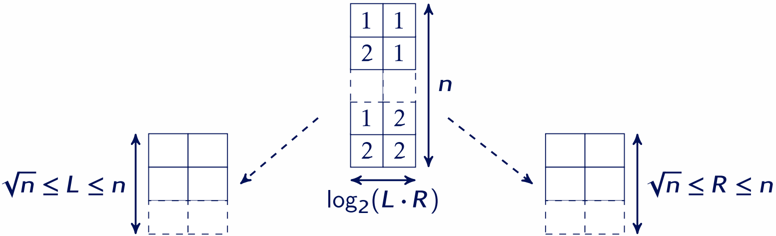
Compact Tree Idea



Idea: store the universe of tuples in tree compactly

- $\log_2(|U|) = 2w$ (internal pointers)
- $|U| = L \cdot R$
- $n \leq |U| \leq n^2$

Compact Tree Idea

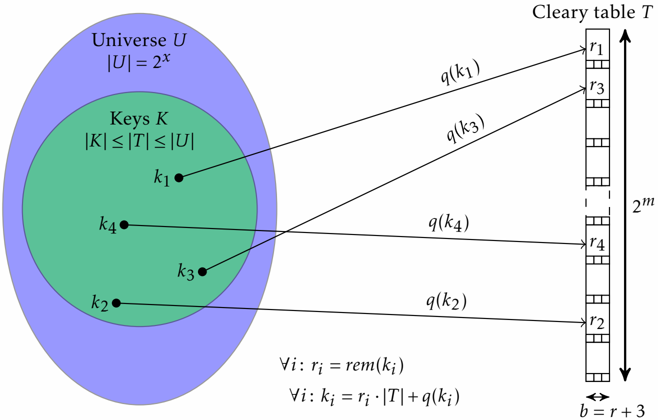


Idea: store the universe of tuples in tree compactly

- $\log_2(|U|) = 2w$ (internal pointers)
- $|U| = L \cdot R$
- $n \leq |U| \leq n^2$
- Can still be larger than optimal: $2w > u + \log_2(k) + \epsilon$

Storing Tuples for $\log_2(|U|) \in \mathcal{O}(\log_2(n))$

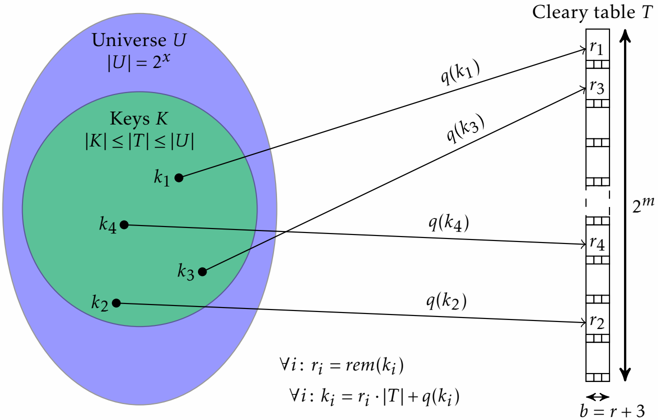
[Cleary '84] [Geldenhuys, Valmari – spin'03][vdVegt, Laarman – memics'11]



- Cleary compact table 'splits' keys into quotient & remainder
- Quotient is used to hash (therefore is $\log_2(|T|)$ bit)

Storing Tuples for $\log_2(|U|) \in \mathcal{O}(\log_2(n))$

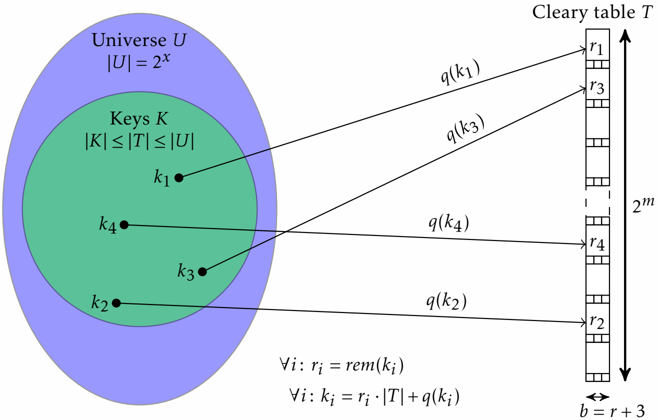
[Cleary '84] [Geldenhuys, Valmari – spin'03][vdVegt, Laarman – memics'11]



- Cleary compact table 'splits' keys into quotient & remainder
- Quotient is used to hash (therefore is $\log_2(|T|)$ bit)
- Only remainder is stored ($r = \log_2(|U|) - \log_2(|T|)$ bit)
  (additionally, 3 administration bits are needed to resolve collisions.)

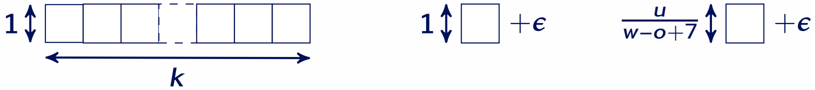
Storing Tuples for $\log_2(|U|) \in \mathcal{O}(\log_2(n))$

[Cleary '84] [Geldenhuys, Valmari – spin'03][vdVegt, Laarman – memics'11]



- Cleary compact table 'splits' keys into quotient & remainder
- Quotient is used to hash (therefore is $\log_2(|T|)$ bit)
- Only remainder is stored ($r = \log_2(|U|) - \log_2(|T|)$ bit)
  (additionally, 3 administration bits are needed to resolve collisions.)
- In practice, compression from $2w$ up to w (i.e., often close to optimal)

Theorem about compression



Uncompressed | Information Theory | This paper

Theorem

Let Tree_{opt} *be the best-case compact-tree compressed vector sizes.*
We have $\mathsf{Tree}_{opt} \leq \frac{w-o+7}{u}\mathsf{Entropy}$ *provided* $8 \leq k \leq \sqrt[4]{n}+4$.

Experiments [±500 models]

Language	**Models**
DVE	All benchmarks from the BEEM database
Promela	All SPIN case studies, e.g.: GARP/i/x509/BRP protocols, etc
Petri nets	Subset from MCC 2016 competition also considered by [Jensen et al. NASA FM'17]
mcrl2 process algebra	Several industrial case studies

Model Checkers

- LTSmin [Kant, Laarman, et al. – TACAS'15]
- SPIN, only Promela [Holzmann '97]
- `verifypn` [Jensen et al. NASA FM'17]

Experiments [±500 models]

Language	**Models**
DVE	All benchmarks from the BEEM database
Promela	All SPIN case studies, e.g.: GARP/i/x509/BRP protocols, etc
Petri nets	Subset from MCC 2016 competition also considered by [Jensen et al. NASA FM'17]
mcrl2 process algebra	Several industrial case studies

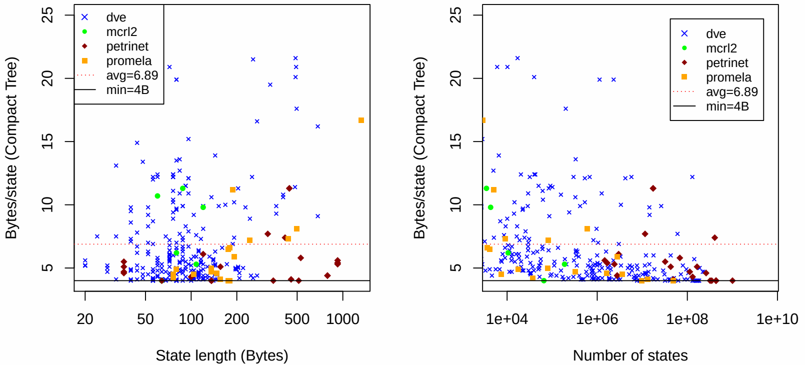
Model Checkers

- LTSmin [Kant, Laarman, et al. – TACAS'15]
- SPIN, only Promela [Holzmann '97]
- `verifypn` [Jensen et al. NASA FM'17]

Tree Configuration

- $w = u = 30$
- Optimal is compressed size is 4 bytes / state

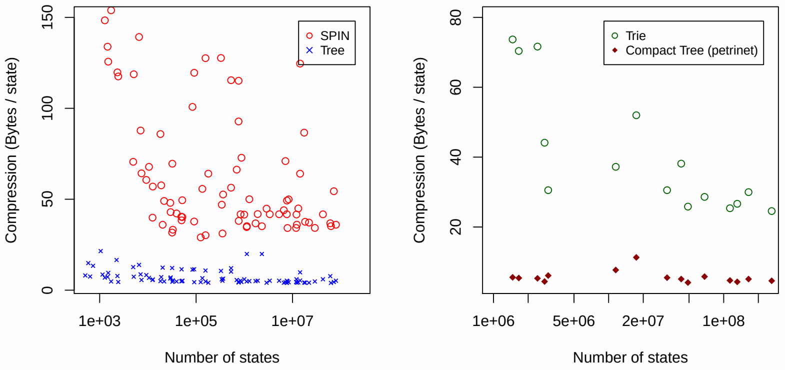
Experiments: Compression



Compression reaches information theoretic optimum

- Lossless compression up to 4 bytes / state
- Mean compression of 6.9 bytes / state
- Also for long vectors and large state spaces

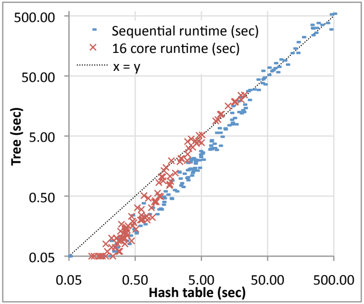
Experiments: Compression



Comparison

- SPIN uses collapse compression [Holzmann '97]
- verifypn uses a trie [Jensen et al. NASA FM'17]

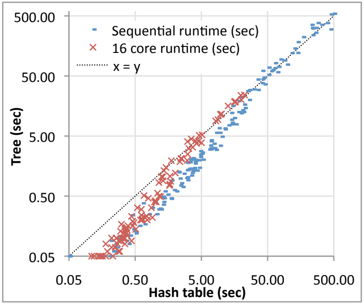
Experiments: Performance

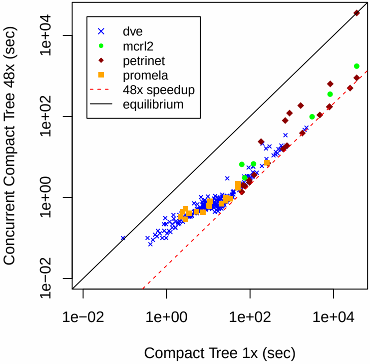


Runtime performance and parallel scalability

- As fast as hash table

Experiments: Performance

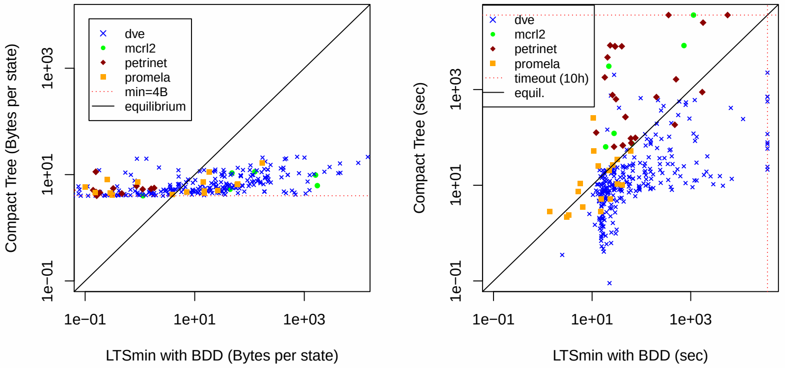




Runtime performance and parallel scalability

- As fast as hash table
- Scalable on 48 core machine

Experiments: BDD Comparison



Compact Tree vs BDDs

- BDD compression is non-linear
- Performance is input-dependent

“Optimal” Compression for Free

Conclusion

- “Optimal” compression for free
- Parallelizable

“Optimal” Compression for Free

Conclusion

- “Optimal” compression for free
- Parallelizable

Future Work

- Apply tree compression elsewhere
- Apply information theory to decision diagrams (non-linear)
- Distributed tree with bit array