Perfect Hashing in Theory and Practice

Stefan Edelkamp
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- Dynamic Perfect Hashing
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- Dynamic Perfect Hashing
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
Motivation: Find a Phone Number

a) In a sorted list

<table>
<thead>
<tr>
<th>Phone Number</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>04212182312</td>
<td>Sabine</td>
</tr>
<tr>
<td>04212183316</td>
<td>Martin</td>
</tr>
<tr>
<td>04212184039</td>
<td>Andree</td>
</tr>
<tr>
<td>04212184767</td>
<td>Stefan</td>
</tr>
<tr>
<td>04212187089</td>
<td>Otthein</td>
</tr>
<tr>
<td>04212187824</td>
<td>Michael</td>
</tr>
<tr>
<td>04212188175</td>
<td>Hagen</td>
</tr>
<tr>
<td>04212188797</td>
<td>Lothar</td>
</tr>
<tr>
<td>04212189740</td>
<td>Thomas</td>
</tr>
</tbody>
</table>

Number of steps *increases* when the amount of information grows.

b) In a Hash Table

Use a "hash function" to generate and remember random locations.

Got it!
Hashing

- Set of Keys $S$, Universum of all possible Keys $U$

- Hash Function

- Hash Table $T = [0, \ldots, m-1]$
Adress Collisions: Birthday Paradoxon
\[ P(A') = \text{probability of not being any two people having the same birthday} \]
\[ = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365} \times \ldots \times \frac{343}{365} \]
\[ = \frac{365!}{342!} \times \left(\frac{1}{365}\right)^{23} = 0.49270276 \]
\[ \Rightarrow P(A) = \text{probability of two people having the same birthday} \]
\[ \Rightarrow = 1 - 0.49270276 = 0.507297 \text{ (50.729\%)} \]
State space of $n = 2^{30}$ elements uniformly hashed to the $m = 2^{64}$ possible words $\Rightarrow$

$$P(A') = \frac{m!}{(m^n(m - n)!) = \frac{m \cdot \ldots \cdot (m - n + 1)}{m^n}}$$

$$\geq \frac{(m - n + 1)/m}{m/n} = \left(1 - \frac{n}{m}\right)^n$$

$$(1 - 2^{-34}) 2^{30} \geq (.99999999994179233909)^{1073741824} = (.99999999994179233909)^{(10737)^{1000000} + 41824} = 93.94\%$$

Game Players do it anyway, e.g. Checkers is „solved“
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- Dynamic Perfect Hashing
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
Universal Hashing

Idea: draw hash function randomly from class

\[ H: \{ h \mid h: U \rightarrow [0..m-1] \} \]

Def. \( H \) is universal if for any \( h \) in \( H \) we have

\[ |\{ h \in H \mid h(x) = h(y) \}| \leq |H|/m \]

\[ P(h(x) = h(y)) \leq 1/m \]

Example: \( H_m = \{ h = ((ax+b) \mod p) \mod m \mid a \in [1..p-1], b \in [0..p-1] \} \)
Example

\[ x = 1, \ y = 4, \ m = 3, \ p = 5 \implies \]
\[ |H| = 20, \ a \in [0..3], \ b \in [0..4] \implies \text{Collisions:} \]

\[ (1 \times 1 + 0) \mod 5 \mod 3 = 1 = (1 \times 4+0) \mod 5 \mod 3 \]
\[ (1 \times 1 + 4) \mod 5 \mod 3 = 0 = (1 \times 4+4) \mod 5 \mod 3 \]
\[ (4 \times 1 + 0) \mod 5 \mod 3 = 1 = (4 \times 4+0) \mod 5 \mod 3 \]
\[ (4 \times 1 + 4) \mod 5 \mod 3 = 0 = (4 \times 4+4) \mod 5 \mod 3 \]
Set of **Keys** $S$, **Universum** of all possible Keys $U$

- Hash Function
- Hash Table $T=[0,...,m-1]$
FKS Hashing

Fredman, Komlós and Szemerédi (82)
1. draw $h$ in universal class $H$
2. for all $x$ in $S$ compute $i=h(x)$ and $B_i$
3. if $\left( \sum |B_i|^2 \geq 5n \right)$ goto 1
4. for $i$ in $[0..m-1]$
   a) draw $h_i$ in universal class $H_i$
   b) If not (hi injective) goto a)

$\Rightarrow O(n)$ time, $O(n)$ space
FKS Search

1. \( i := h(x) \)
2. \( T' := T[i] \)
3. extract \((bi, |Bi|, hi)\)
4. return \( x = T'[bi+hi(x)] \)

\( \Rightarrow O(1) \) time
m = 30, p = 31, n = 6
S = \{2, 4, 5, 15, 18, 30\}

| T | ≤ 6n
5 probes to find 30
Correctness FKS

\[ H(k,p,s) = \{ h = (kx \mod p) \mod s \mid k \in [1..p-1] \} \text{ (univ. class)} \]

**Thm:** \( B_i := B(s,S,k,i) = \{ x \in S \mid h \in H(k,p,s): h(x)=i \} \)

\[ \Rightarrow \quad \exists k \in U, s \geq n : \sum_i |B_i|((|B_i|-1)/2) < n^2/s. \]

**Proof:** For pair \( (k,\{x,y\}) \): \( kx \mod p \mod s = ky \mod p \mod s \quad \Rightarrow \quad k*(x-y) \mod p \mod s \in \{s,2s,3s,...,p-s,p-2s,p-3s\} \) 

\( k \) mult. Inverses mod \( p \) \( \Rightarrow \quad \# k's \leq 2(p-1)/s \)

summing over all \( (n(n-1)/2) \) pairs \( (x,y) \) \( \Rightarrow \quad \# \text{ pairs } (k,\{x,y\}) \text{ with} \)

\( kx \mod p \mod s = ky \mod p \mod s \) \( < (p-1)n^2/s \quad \Rightarrow \quad \sum_k \sum_i |B_i|((|B_i|-1)/2) < (p-1)n^2/s \quad \Rightarrow \quad \sum_i |B_i|((|B_i|-1)/2) < n^2/s. \)
Corollary 1: Exists $k$ in $U$: $\sum_i |B_i|^2 < 3n$

Proof: $\sum_i |B_i|^2 = 2 \sum_i |B_i|(|B_i|-1)/2) + 2 \sum_i |B_i|/2 < 2n^2/s + n \leq 3n$

Corollary 2: Exists $k'$ in $U$ for all $i$: $|B'i|=B(n^2,S,k',i) | \leq 1$

Proof: $B'i = \{ x \in S | h = (k'x \mod p) \mod n^2: h(x) = i \}$

$\Rightarrow$ (Thm) $\Rightarrow \sum_i |B'i|(|B'i|-1)/2) \leq 1$

$\Rightarrow |B'i| \leq 1$ for all $i$

$\Rightarrow h$ is one-to-one on $S$
One \( k \) in \( U \) not sufficient for fast construction, need \( \geq |U|/2 \)

\( H = \{ h = (kx \mod p) \mod n \mid k \in [1..p-1]\} \) univ. class

**Corollary 3:** For at least half of all possible \( k \) in \( U \)

\[ \sum |B_i|^2 < 5n \] [Stage 1 Hash]

**Proof:** Most a half values smaller than 2* average

(Thm 1) \( \sum_i |B_i|(|B_i|-1)/2 < 2n. \)

**Corollary 4:** For at least half of all possible \( k' \) in \( U \)

\[ |B'i| \leq 1 \] [Stage 2 Hash]

**Proof:** Analogous to Corollary 2

**Corollary 5:** Space can be reduced to \( n + o(n) \).

**Proof:** Substitute \( n \) with \( g(n) \) given \( \lim g(n)/n = 0 \)
Further Results on (Minimum) Perfect Hashing

Lower bound: At least $\log e = 1.44$ bits per element

(Proof: e.g., Mehlhorn (82)+ Dietzfelbinger et al. (09))

Mehlhorn (82): At least $\theta(n + \log \log |U|)$ bits.

Fredman and Komlós (84): $n \log e + \log \log |U| + \theta(n)$ bits

Schmidt and Siegel (90): Existence of $n + \log \log |U|$ bits for $O(1)$ minimum perfect hash function, no construction

Dietzfelbinger, Gil, Matias, Pipinger (92): Generalization of FKS to dynamic setting (via polynomial hash functions)

Majewski, Wormald, Havas, Czech (96): $O(1)$ perfect hashing with $O(n \log n)$ bits (using hypergraph theory)

Hagerup/Tholey (01): $O(1)$ minimal perfect hashing, $n \log e + \log \log |U| + O(n(\log \log n)^2/\log n + \log \log \log |U|)$ (01)

Edelkamp/Meyer (01): Suffix-Lists with space close to lower bound
Lower Bound(s)

\[ |H| \geq \binom{u}{n} \binom{m}{n} \]

subsets \( S \) of \( U \)

\[ \log |H| \geq \sum \log(1 - i/u) - \sum \log(1 - i/m) \rightarrow NR-1 \]

\[ \log |H| \geq (m-n+1) \log (m-n+1/m) - (n-u) \log (1-n/u) \]

\( u \gg n \rightarrow NR-2 \rightarrow m = 1.23n \) requires \( \geq 0.89 \) bits per key

\( m = n \) requires \( \geq 1.44 \) bits per key
Botelho, Pagh and Ziviani (07) Implementation of memory-based hash function based FKS + hypergraph theory

- $r$-uniform random hypergraphs with $r$ hash functions on the keys of $S$; $r=2$: Two tables of size $(2+\varepsilon)m$, divide in $m=255$
- For $r = 3$ they obtained a space usage of approximately $2.62n$ bits for a minimum perfect hash function

Features:

- Hash value computed efficiently
- Constant access to identifier
- All keys need to be known beforehand

Botelho and Ziviani (07)

Works well for data stored on disk
Practical Minimal Perfect Hashing

cmph: C Library for Minimum Perfect Hash Functions
http://cmph.sourceforge.net/
Currently best: HDC based on „Hash Displace and Compress“ by Belazzougui, Botelho and Dietzfelbinger (09)
(...following Hash & Displace by Pagh (99) and Tarjan/Yao (79))
O(n) construction O(1) evaluation, close to 1.44n bits, e.g
- For m = n it has 2.07 bits per key,
- For m = 2n it has 0.67 bits per key,
- For m = 1.23n it has 1.4 bits per key

SUX4J: Succinct Data Structures Umbrella (http://Sux4j.dsi.unimi.it)
Java based minimal perfect hashing using ~ 2.65 bits per element + implementations of monotone minimal perfect hashing...
Monotone Minimal Perfect Hashing

**Known:** Given that keys can be in any order → order-preserving hash requires $\Theta(n \log n)$ bits

Belazzougui, Boldi, Pagh, Vigna (09) show (& implement)

**Thm:** Given keys in lexicographic order

a) $O(n \log \log \log |U|)$ space and $O(\log \log |U|)$ search time

b) $O(n \log \log |U|)$ space and $O(1)$ search time

Use $O(1)$ rank and select
- $\text{rank}(p,0001001010) = \# 1 \text{ till } p$
- $\text{select}(r,0001001010) = \text{position of } r \text{ th } 1$

**Bucketing**

a) longest common prefixes

b) relative ranking
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- **Dynamic Perfect Hashing**
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
Dynamic Perfect Hashing: Cuckoo Hashing

Pagh & Rodler (03)

2 Tables of size $n + \varepsilon$, 2 univ. hash functions

- $O(1)$ look-up
- Fast insert
Cuckoo Hashing

Search: The hash function provides *two* possible locations.

04212188175 Hagen
Not here
04212183316 Martin
04212188797 Lothar
04212184039 Andree

Where to find 04212187824?

Got it!
04212187824 Michael
04212187089 Otthein
04212182312 Sabine
04212189740 Thomas
04212184767 Stefan
Cuckoo Hashing

**Insert:** New information is inserted; if necessary, kick out old information.

04212187395 Gerrit
04212182312 Sabine
04212183316 Martin
04212188797 Lothar
04212184039 Andree

04212187824 Michael
04212189740 Thomas
04212188175 Hagen
04212187089 Otthein
04212184767 Stefan
Cuckoo Hashing Insert Algorithm

1. Compute $h_1(x)$
2. If $T[h_1(x)]$ empty, $T[h_1(x)]:=x$
   else $y:=T[h_1(x)]$ and $T[h_1(x)]:=x$
3. Look at $T[h_1(y)]$ and $T[h_2(y)]$ that is not occupied by $x$. If empty, insert $y$. If not, put $y$ there and evict $z$. Set $x:=y$ and $y:=z$
4. Goto 3 $O(\log n)$ times until an empty spot is found. Otherwise, pick a new pair of hash functions and rehash.
Cuckoo Hashing Failures

Bad case 1: inserted element runs into cycles.
Bad case 2: inserted element has very long path before insertion completes (Could be on a long cycle).

Observation: Bad cases occur with small probability when load is sufficiently low

Solution: re-hash everything if a failure occurs.

Load less than 50%, n elements gives failure rate of \( \Theta(1/n) \); maximum insert time \( O(\log n) \)

Better Space Utilization: Dietzfelbinger & Weidling (07) generalize cuckoo hashing to bucketed cuckoo hashing that uses more than 1 entry per table
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- Dynamic Perfect Hashing
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
Perfect Hashing for Permutation Games

n-TOPSPIN

nxm Sliding Tile

n-PanCake
Constant-Bit State Space Search

- Assumes Perfect Hash Function Rank
- Applies to State Space Search, if \( m \geq |\text{Reach}(\text{init})| \)
- Minimal, if \( m = |S| \)
- Inverse Unrank needed to reconstruct state

Features:
- Two-Bit BFS (Cooperman/Finkelstein 92, Korf 08)
- One-Bit Breadth-First Search
- One-Bit Reachability (Edelkamp/Sulewski 09)
Two-Bit BFS

Interpretation: 3=UNSEEN, \{0,1,2\} = depth mod 3

\[
\begin{align*}
E: & \quad 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 1 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \\
G: & \quad 3 \ 3 \ 3 \ 3 \ 3 \ 2 \ 3 \ 3 \ 1 \ 3 \ 3 \ 2 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \\
E: & \quad 3 \ 3 \ 3 \ 3 \ 3 \ 2 \ 3 \ 3 \ 1 \ 3 \ 3 \ 2 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \ 3 \\
G: & \quad 3 \ 3 \ 3 \ 0 \ 3 \ 3 \ 2 \ 3 \ 3 \ 1 \ 3 \ 3 \ 2 \ 3 \ 3 \ 3 \ 0 \ 3 \ 3 \ 3 \ 3 \ 3 \\
E: & \quad 3 \ 3 \ 3 \ 0 \ 3 \ 3 \ 2 \ 3 \ 3 \ 1 \ 3 \ 3 \ 2 \ 3 \ 3 \ 3 \ 0 \ 3 \ 3 \ 3 \ 3 \ 3 \\
\ldots
\end{align*}
\]
One-Bit BFS

Assuming: Move-Alternation Property

\[ E: 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 \]
\[ G:0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 \]
\[ E: 0 0 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 \]
\[ G:0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 \]
\[ E: 0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 \]
\[ G:0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 \]
\[ E: 0 0 1 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 \]
\[ G:0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 \]

\[ \Rightarrow \text{Thm: } |\text{BFS-Layer}(i)| = \text{PopCount}(i) - \text{PopCount}(i-1) \]
One-Bit Reachability

\[ \begin{align*}
E: & \quad 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
G: & \quad 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\
E: & \quad 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \\
G: & \quad 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \\
E: & \quad 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \\
\end{align*} \]

\[ \Rightarrow \text{Thm: } \text{Scan}(i) \leq \text{Depth}(i) \]
Parity in Sliding-Tile

Solvable states have same **parity** as the goal

⇒ **parity** is key concept to rank state to \([0..(nm)!/2-1]\)

**Caution:** Swapping a tile with the blank is move

⇒ partition state space wrt. „blank“ position

**Obs:** Moving blank left from Bi to Bi-1 or right from Bi to Bi+1 does not change the rank

**Proof:** Relative order of tiles does not change
Thm: If $n$ even, $k$ odd then \textit{parity} remains unchanged

$\Rightarrow$ Compression to $[0..(n-1)!/2-1]$
Lexicographic

- Permutations with opposite parity next to each other → compression
- No linear time and space algorithm known

Myrvold Ruskey (05)

- Linear time and space
- News (Edelkamp & Sulewski 09): including parity computation
Algorithm $\text{rank}(n, \pi, \pi^{-1})$

1: for all $i$ in $\{1, \ldots, n-1\}$ do
2: \hspace{1em} $l \leftarrow \pi_{n-i}$
3: \hspace{1em} swap($\pi_{n-i}, \pi_{\pi_{n-i}^{-1}}$)
4: \hspace{1em} swap($\pi_{\pi_{n-i}^{-1}}, \pi_{n-i}^{-1}$)
5: \hspace{1em} $\text{rank}_i \leftarrow l$
6: return $\prod_{i=1}^{n-1} (\text{rank}_{n-i+1} + i)$
### Example Myrvold Ruskey with Parity

<p>| (1,2,3,0) | 0 | (2,1,3,0) | 1 |
| (3,2,0,1) | 0 | (2,3,0,1) | 1 |
| (1,3,0,2) | 0 | (3,1,0,2) | 1 |
| (1,2,0,3) | 1 | (2,1,0,3) | 0 |
| (2,3,1,0) | 0 | (3,2,1,0) | 1 |
| (2,0,3,1) | 0 | (0,2,3,1) | 1 |
| (3,0,1,2) | 0 | (0,3,1,2) | 1 |
| (2,0,1,3) | 1 | (0,2,1,3) | 0 |
| (1,3,2,0) | 1 | (3,1,2,0) | 0 |
| (3,0,2,1) | 1 | (0,3,2,1) | 0 |
| (1,0,3,2) | 1 | (0,1,3,2) | 0 |
| (1,0,2,3) | 0 | (0,1,2,3) | 1 |
| (1,2,3,0) | 0  | (2,1,3,0) | 1  |
| (3,2,0,1) | 0  | (2,3,0,1) | 1  |
| (1,3,0,2) | 0  | (3,1,0,2) | 1  |
| (1,2,0,3) | 1  | (2,1,0,3) | 0  |
| (2,3,1,0) | 0  | (3,2,1,0) | 1  |
| (2,0,3,1) | 0  | (0,2,3,1) | 1  |
| (3,0,1,2) | 0  | (0,3,1,2) | 1  |
| (2,0,1,3) | 1  | (0,2,1,3) | 0  |
| (1,3,2,0) | 1  | (3,1,2,0) | 0  |
| (3,0,2,1) | 1  | (0,3,2,1) | 0  |
| (1,0,3,2) | 1  | (0,1,3,2) | 0  |
| (1,0,2,3) | 0  | (0,1,2,3) | 1  |</p>
<table>
<thead>
<tr>
<th>(1,2,3,0)</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3,2,0,1)</td>
<td>0</td>
</tr>
<tr>
<td>(1,3,0,2)</td>
<td>0</td>
</tr>
<tr>
<td>(1,2,0,3)</td>
<td>1</td>
</tr>
<tr>
<td>(2,3,1,0)</td>
<td>0</td>
</tr>
<tr>
<td>(2,0,3,1)</td>
<td>0</td>
</tr>
<tr>
<td>(3,0,1,2)</td>
<td>0</td>
</tr>
<tr>
<td>(2,0,1,3)</td>
<td>1</td>
</tr>
<tr>
<td>(1,3,2,0)</td>
<td>1</td>
</tr>
<tr>
<td>(3,0,2,1)</td>
<td>1</td>
</tr>
<tr>
<td>(1,0,3,2)</td>
<td>1</td>
</tr>
<tr>
<td>(1,0,2,3)</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(2,1,3,0)</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,3,0,1)</td>
<td>1</td>
</tr>
<tr>
<td>(3,1,0,2)</td>
<td>1</td>
</tr>
<tr>
<td>(2,1,0,3)</td>
<td>0</td>
</tr>
<tr>
<td>(3,2,1,0)</td>
<td>1</td>
</tr>
<tr>
<td>(0,2,3,1)</td>
<td>1</td>
</tr>
<tr>
<td>(0,3,1,2)</td>
<td>1</td>
</tr>
<tr>
<td>(0,2,1,3)</td>
<td>0</td>
</tr>
<tr>
<td>(3,1,2,0)</td>
<td>0</td>
</tr>
<tr>
<td>(0,3,2,1)</td>
<td>0</td>
</tr>
<tr>
<td>(0,1,3,2)</td>
<td>0</td>
</tr>
<tr>
<td>(0,1,2,3)</td>
<td>1</td>
</tr>
</tbody>
</table>
Algorithm `unrank(r)`

1: \( \pi := id \)
2: \textbf{parity} := false
3: \textbf{while} \( n > 0 \) \textbf{do}
4: \( i := n - 1 \)
5: \( j := r \mod n \)
6: \textbf{if} \( i \neq j \) \textbf{then}
7: \textbf{parity} := \neg \text{parity}
8: \textbf{swap}(\pi_i, \pi_j)
9: \( r := r \div n \)
10: \( n := n - 1 \)
11: \textbf{return} (\text{parity}, \pi)
Parallelism for the „masses“

- Current CPUs have 2, 4 or 8 cores
- Current (GP)GPUs have 540 cores

⇒ Huge potential to be exploited
State Space Search on the GPU
### BFS-Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>n x m</th>
<th>States</th>
<th>GPU Time</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sliding-Tile (One-Bit)</strong></td>
<td>6 x 2</td>
<td>479,001,600</td>
<td>149s</td>
<td>1,110s</td>
</tr>
<tr>
<td></td>
<td>7 x 2</td>
<td>39,916,800</td>
<td>13,590s</td>
<td>0.0t</td>
</tr>
<tr>
<td><strong>Pancake (Two Bit)</strong></td>
<td>12</td>
<td>479,001,600</td>
<td>290s</td>
<td>9,287s</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>39,916,800</td>
<td>27s</td>
<td>920s</td>
</tr>
<tr>
<td><strong>Top-Spin (Two-Bit)</strong></td>
<td>12</td>
<td>479,001,600</td>
<td>290s</td>
<td>9,287s</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>39,916,800</td>
<td>27s</td>
<td>920s</td>
</tr>
</tbody>
</table>
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- Dynamic Perfect Hashing
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
Selection Games

Peg-Solitaire

Nine-Men-Morris

Frogs and Toads

Fox and Geese
Hashing Selection Games

Binomial Coefficients

{11110000, 11101000, 11100100, 11100010, 11100001, 11010000, ...
   10000111, 01111000, 01110100, ...
   01001110, 01000111, 00111100, ...
   01000111, 00001111}

Found "1"

{01110000, 01100000, ...
   01000110, 01000111, 00111100, ...
   00001111}

Found "0"
Another Point of View

1 seen

0 seen

\[ \begin{align*}
&\binom{n}{k} \\
&\binom{n-1}{k-1}
\end{align*} \]
Edelkamp, Messerschmidt & Sulewski (09)

Binomial: Peg-Solitaire, Fox and Geese, Frogs & Toads

Binomial Table:  
- \(O(n+n^2+kn)\) Time  
- \(O(n^2)\) Space

Multinomial: Nine-Men-Morris

Multinomial Table:  
- \(O(n+n^3+kn)\) Time  
- \(O(n^3)\) Space

Factorial Table:  
- \(O(n+kn)\) Time  
- \(O(n)\) Space
Our Results in 09

Solved = Solvability Status of all Reachable States Known

- Peg-Solitaire: Solved, 12m CPU, 1m GPU
- Frogs and Toads (4x4): Solved, 30min GPU
- Fox and Geese: Solved (outcome depending on the number of geese), 1 month on 8 CPU Cores
- Nine-Men Morris: Solved (draw) few days on GPU
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- Dynamic Perfect Hashing
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
Automata-based (LTL) Model Checking $M \models \phi$:

- Model $M$ given in some formal spec.
- Property $\phi$ to check given in some formal spec.
- Algorithm detects accepting cycles in (Büchi) Automtation constructed wrt. $M$ and $\phi$
Model Checking Algorithm

Search **Accepting Cycles** / Lasso

**Non-Optimal Algorithm:** Nested/Double DFS by Courcoubetis, Vardi, Wolper, Yannakakis (92)

**Optimal Algorithm:** Search Minimum Accepting Cycles, one with min(d+n)
Edelkamp, Sanders and Simecek (08):  
Generate state space (External on HDD)  
Create a perfect hash function from disk (RAM)  
Allocate a bit-vector, one bit for each state (RAM)  
Run Double DFS, mark visited states by setting corresponding bit  

PHF  

Visited bits  

X X X X X X X X X
<table>
<thead>
<tr>
<th>Model</th>
<th>Number of Vertices</th>
<th>$v_{max}$</th>
<th>$\epsilon_s$</th>
<th>MPHF Size (bits/vertex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elev.2(16),P4</td>
<td>173,916,122</td>
<td>30 bytes</td>
<td>94</td>
<td>4.941</td>
</tr>
<tr>
<td>Lamport(5),P4</td>
<td>74,413,141</td>
<td>24 bytes</td>
<td>99</td>
<td>4.941</td>
</tr>
<tr>
<td>MCS(5),P4</td>
<td>119,663,657</td>
<td>28 bytes</td>
<td>91</td>
<td>4.941</td>
</tr>
<tr>
<td>Peterson(5),P4</td>
<td>284,942,015</td>
<td>32 bytes</td>
<td>177</td>
<td>4.941</td>
</tr>
<tr>
<td>Phils(16,1),P3</td>
<td>61,230,206</td>
<td>50 bytes</td>
<td>47</td>
<td>4.941</td>
</tr>
<tr>
<td>Ret.(16,8,4),P2</td>
<td>31,087,573</td>
<td>91 bytes</td>
<td>553</td>
<td>4.941</td>
</tr>
<tr>
<td>Szyman.(5),P4</td>
<td>419,183,762</td>
<td>32 bytes</td>
<td>223</td>
<td>4.941</td>
</tr>
</tbody>
</table>
Flash Memory (Solid State Disks)

- Expecting falling prices
  - Decreasing discrepancy to RAM (?)
- Increasing Capacity
- Low power consumption
- SSD are replacing HDD (at least in mobile devices)
- Faster random read access time than HDD
- Same random write access time than HDD
- Good for static dictionaries (like perfect hash functions)
Minimal Accepting Cycles

Gastin and Moro (06)

First stage
  - create whole state space with BFS
  - collect all accepting states

Second stage
  - BFS at each accepting state checks for a cycle

Third stage
  - BFS to find the shortest lasso
Edelkamp & Sulewski (08) Semi-External Flash-Memory MC to generate Minimal-Counter Example

- Duplicate Detection via External Minimum Perfect Hash Function
- Minimum Perfect Hash Function stored on SSD

Brim, Edekamp, Simecek and Sulewski (08) On-the-Fly Flash-Memory Model Checking

- Early Duplicate Detection feasible for external memory model checking (CPU usage > 70%)
## Some Results

<table>
<thead>
<tr>
<th>Experiment</th>
<th>StateSpace</th>
<th>PFH in RAM</th>
<th>PHF on external device</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RAM</td>
<td>Time</td>
</tr>
<tr>
<td>Szymanski P3-2</td>
<td>1.5 MB</td>
<td>28.7 KB</td>
<td>0:06</td>
</tr>
<tr>
<td>Szymanski P3-3</td>
<td>65 MB</td>
<td>0.99 MB</td>
<td>2:58</td>
</tr>
<tr>
<td>Lifts P4 - 7</td>
<td>351 MB</td>
<td>4.5 MB</td>
<td>4:27</td>
</tr>
<tr>
<td>Lifts P2 – 8</td>
<td>1559 MB</td>
<td>20.2 MB</td>
<td>19:44</td>
</tr>
</tbody>
</table>
Overview

- Motivation Hashing
- Universal and Perfect Hashing
- Dynamic Perfect Hashing
- Perfect Hashing in Permutation Games
- Perfect Hashing in Selection Games
- Perfect Hashing for Model Checking
- Perfect Hashing with BDDs
(Read Once) Binary Decision Diagram

DAG with each variable on every path appearing at most once wrt. fixed variable ordering. Nodes are variables, edges are labeled either 0 or 1, sinks denoted by 0 and 1

→ Unique Representation of Boolean Functions

Quasi-Reduced (RO)BDD

DAG with each variable on every path appearing exactly once [..]

SatCount(f) = |{(a1,...,an) | f(a1,..,an) = 1}|
BDD (parity)
SAT

0000=1
0011=2
0101=3
0110=4
1001=5
1010=6
1100=7
1111=8
SATCOUNT
Time $O(|G|)$
Perfect Hashing for State Spaces in BDD Representation

Dietzfelbinger & Edelkamp (09)

- Perfect Hash Function (Rank/Unrank) from BDD $G$ for $f(x_1, \ldots, x_n)$ to $[1..\text{satcount}(f)]$ in
  - $O(|G|)$ Space and Preprocessing Time
  - $O(n)$ Ranking and Unranking Time
Rank Quasi-Reduced

\texttt{lex-rank}(G^*,s,v)

\textbf{if} v is 0-sink \textbf{return} 0

\textbf{if} v is 1-sink \textbf{return} 1

\textbf{if} v is node labeled \emph{x_i} with 0-succ. u and 1-succ. w

\textbf{if} (s_i = 1) \textbf{return} \texttt{sat-count}(u) + \texttt{lex-rank}(G^*,s,w)

\textbf{if} (s_i = 0) \textbf{return} \texttt{lex-rank}(G^*,s,u)

\texttt{rank}(G^*,s)

\textbf{return} \texttt{lex-rank}(G^*,s,\text{root of } G^*)
Rank(1111)
Rank(1111) = 4 + 2 + 1 + 1 = 8
Rank(0110) = 2+1+1 = 4
unrank(G*, r)

i := 1; start at root of G

while (i <= n)

at node v for x_i with 0-succ. u and 1-succ. w

if (r > sat-count(u))

r := r - sat-count(u)

follow 1-edge to w, record s_i := 1

else follow 0-edge to u, record s_i := 0

i := i + 1
Unrank(2) =
(2>4) ? No (0)

Unrank(1) =
(2>2) ? No (0)

Unrank(1) =
(2>1) ? Yes (1)

Unrank(0) =
(1>0) ? Yes (1)
Applications

- **Compression of Data Sets**: Symbolic Equivalent to Explicit (Minimum) Perfect Hash Functions of Botelho et al. (07)

- **Constant Bitvector Search**: based on State Spaces Generated Symbolically, e.g. Connect-Four with 4,531,985,218,092 states

- Uniformly Drawn **Random Satisfying Input** for Boolean Functions
„HASH, X. – THERE IS NO DEFINITION FOR THIS WORD - NOBODY KNOWS WHAT HASH IS.“

- AMBROSE BIERCE, DEVIL’S DICTIONARY 1906,
- FOUND IN KNUTH (98), VOL III

END OF TALK