Large-Scale Knowledge Processing (1st part, lecture-6)

Shin-ichi Minato

Division of Computer Science and Information Technology, Graduate School of Information Science and Technology, Hokkaido University
Review of the last lecture

ZDD applications (1): Database analysis

- ZDD-vector and “Itemset-histogram algebra”
  - ZDD-representation for polynomials with integer coefficients
  - Operation algorithms for itemset-histogram data
  - VSOP: Interpreter for manipulating itemset-histogram data

- Frequent itemset mining using ZDDs
  - ZDD-growth: ZDD-based frequent itemset mining method
  - Algorithm for generating maximal/closed itemsets

- ZDD variable ordering in database analysis
- Mining internal structures of combinatorial itemset data
  - Simple disjunctive decomposition of Boolean functions
Exercises of the last lecture

- Draw a ZDD-vector representing the polynomial
  \( a b c + 3 a b + 2 a c + 6 a + b c + 3 b + 2 c + 6 \),
  which is equivalent to \((a+1)(b+2)(c+3)\).

- Use ordinary 3-bit binary encoding.

<table>
<thead>
<tr>
<th>tuple</th>
<th>value (bin.)</th>
<th>(F_0)</th>
<th>(F_1)</th>
<th>(F_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a b c)</td>
<td>1 (001)</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(a b)</td>
<td>3 (011)</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(a c)</td>
<td>2 (010)</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(a)</td>
<td>6 (110)</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(b c)</td>
<td>1 (001)</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(b)</td>
<td>3 (011)</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(c)</td>
<td>2 (010)</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6 (110)</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Exercises of the last lecture

- Draw a ZDD-vector representing the polynomial $a \ b \ c + 3 \ a \ b + 2 \ a \ c + 6 \ a + b \ c + 3 \ b + 2 \ c + 6$, which is equivalent to $(a+1)(b+2)(c+3)$.
- Use (-2) base binary encoding.

<table>
<thead>
<tr>
<th>tuple</th>
<th>val (-2 base)</th>
<th>$F_4$</th>
<th>$F_3$</th>
<th>$F_2$</th>
<th>$F_1$</th>
<th>$F_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \ b \ c$</td>
<td>1 (00001)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a \ b$</td>
<td>3 (00111)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$a \ c$</td>
<td>2 (00110)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>6 (11010)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b \ c$</td>
<td>1 (00001)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>3 (00111)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>2 (00110)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6 (11010)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Topics of this lecture

ZDD extensions for sequences and permutations

- Basic operations of ZDDs (review)
- Basic operations of ZDD-vectors (review)
- Sequence BDDs for manipulating sets of sequences
  - Sets of sequences
  - Encoded ZDDs for representing sets of sequences
  - SeqBDDs and their basic operations
  - Tries and SeqBDDs
- Permutation decision diagrams ($\pi$DDs)
  - Sets of permutations
  - Permutation represented by a set of transpositions
  - $\pi$DDs and their basic operations
  - Applications
Extension of ZDDs (Comb. to higher model)

Current ZDDs

Applications in asymmetric world
Data mining, Machine learning
Advanced searching etc.

Applications with higher data model
Sequence data analysis
Numerical data processing
Processing of trees or semi-structured data

Further outputs
Advanced ZDD-like structure
Develop special new algebraic operations.

Direct outputs
(Combinatorial)
(Higher model)
- Multisets
- Sequences
- Permutations
- Partitions
- Trees, DAGs
- Networks
- etc.

Still many applications remains where ZDDs would be effective.
### Algebraic operations in (ordinary) ZDDs

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>Returns empty set. (0-terminal node)</td>
</tr>
<tr>
<td>${\lambda}$</td>
<td>Returns the set of only null-combination. (1-terminal node)</td>
</tr>
<tr>
<td>$P.\text{top}$</td>
<td>Returns item-ID at root node of $P$.</td>
</tr>
<tr>
<td>$P.\text{offset}(v)$</td>
<td>Subset of combinations not including item $v$.</td>
</tr>
<tr>
<td>$P.\text{onset}(v)$</td>
<td>Gets $P \setminus P.\text{offset}(v)$ and then deletes $v$ from each combination.</td>
</tr>
<tr>
<td>$P.\text{change}(v)$</td>
<td>Inverts existence of $v$ (add / delete) on each combination.</td>
</tr>
<tr>
<td>$P \cup Q$</td>
<td>Returns union set.</td>
</tr>
<tr>
<td>$P \cap Q$</td>
<td>Returns intersection set.</td>
</tr>
<tr>
<td>$P \setminus Q$</td>
<td>Returns difference set. (in $P$ but not in $Q$.)</td>
</tr>
<tr>
<td>$P.count$</td>
<td>Counts number of combinations.</td>
</tr>
</tbody>
</table>

(Extended operations introduced by Minato [9])

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \times Q$</td>
<td>Cartesian product of $P$ and $Q$.</td>
</tr>
<tr>
<td>$P/Q$</td>
<td>Quotient of $P$ divided by $Q$.</td>
</tr>
<tr>
<td>$P%Q$</td>
<td>Reminder of $P$ divided by $Q$.</td>
</tr>
</tbody>
</table>

- $\emptyset$, $\{\lambda\}$, $P.\text{top}$ can be executed in a constant time.
- Other operations are almost linear time for ZDD size.
ZDD construction (applying set operations)
ZDD-vector for itemset-histogram (review)

- Integer values are encoded into binary bit-vectors.
- Each digit of bit-vector can be represented by a ZDD.
- A ZDD-vector represents a itemset-histogram.

<table>
<thead>
<tr>
<th>tuple</th>
<th>value (bin.)</th>
<th>$F_2$</th>
<th>$F_1$</th>
<th>$F_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a\ b\ c$</td>
<td>1 (001)</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$a\ b$</td>
<td>3 (011)</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$a\ c$</td>
<td>2 (010)</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$a$</td>
<td>6 (110)</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$b\ c$</td>
<td>1 (001)</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>3 (011)</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$c$</td>
<td>2 (010)</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6 (110)</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Combining ZDD-vectors to ZDDs (review)

- Special item symbols are defined to combine a ZDD-vector to a ZDD.
  - 20 special symbols can deal with 1,000,000 digits.
  - An itemset histogram data is represented by a 1-word pointer.
Basic operations of Itemset-histogram algebra

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c (c \in N) )</td>
<td>Returns the histogram of only ( \lambda ) with occurrence ( c ). ((c = 0 \text{ means empty}))</td>
</tr>
<tr>
<td>( P.\text{top} )</td>
<td>Returns item-ID of the highest order involved in ( P ).</td>
</tr>
<tr>
<td>( P.\text{offset}(v) )</td>
<td>Returns sub-histogram of itemsets not including item ( v ).</td>
</tr>
<tr>
<td>( P.\text{onset}(v) )</td>
<td>Gets ( P - P.\text{offset}(v) ) and then deletes ( v ) from each itemset.</td>
</tr>
<tr>
<td>( P.\text{change}(v) )</td>
<td>Inverts existence of ( v ) (add / delete) on each itemset.</td>
</tr>
<tr>
<td>( \text{Max}(P,Q) )</td>
<td>Histogram with the greater occurrences between ( P ) and ( Q ).</td>
</tr>
<tr>
<td>( \text{Min}(P,Q) )</td>
<td>Histogram with the less occurrences between ( P ) and ( Q ).</td>
</tr>
<tr>
<td>( P - Q )</td>
<td>Difference of the two histograms. (limited to only positive occurrences.)</td>
</tr>
<tr>
<td>( P.\text{count} )</td>
<td>Counts the variety of itemsets in ( P ).</td>
</tr>
<tr>
<td>( P.\text{upperbound} )</td>
<td>Returns the highest occurrence number among all itemsets in ( P ).</td>
</tr>
<tr>
<td>( P.\text{unitset} )</td>
<td>Returns the histogram with occurrence = 1 for all itemsets in ( P ).</td>
</tr>
<tr>
<td>( P \times Q )</td>
<td>Cartesian product of ( P ) and ( Q ).</td>
</tr>
<tr>
<td>( P/Q )</td>
<td>Quotient of ( P ) divided by ( Q ).</td>
</tr>
<tr>
<td>( P%Q )</td>
<td>Reminder of ( P ) divided by ( Q ).</td>
</tr>
</tbody>
</table>

\( P + Q \) Addition of number of occurrences.
Arithmetic operations for itemset histogram

- ZDDs grow as a result of applying arithmetic operations.

ZDDD package

\[ a - b \]

\[ 3 \times a c - 3 \times b c \]
Sets of sequences (sets of strings)

- Sets of combinations:
  - Don’t consider order and duplication of items
  - “abcc” and “bca” are the same as “abc”.

- Sets of sequences:
  - Distinguishes all finite sequences.
  - \{λ\}, \{ab, aba, bbc\}, \{a, aa, aaa, aaaa\}, etc.
  - Here we exclude infinite sets such as \{a*\}.

- So many real-life applications.
  - Text search and indexing
  - Web (html/xml) data mining
  - Bio informatics (e.g. DNA sequence analysis)
Encoded ZDDs for Sets of sequences

- Pair of (Item - position) is considered different symbol.
  "aaa" $\rightarrow$ "$a_1 a_2 a_3$"
  "aba" $\rightarrow$ "$a_1 b_2 a_3$"

- Alphabet size: $|\Sigma|$
- Maximum length of sequences: $n$
- Total encoded symbols: $|\Sigma| \times n$
- Not very efficient.
  - Many symbols needed.
  - We need to put a fixed maximum length of sequences.
Sequence BDD (SeqBDD)

- Loekito, Bailey, and Pei [2009]
  - Same as ZDD reduction rule.
  - **Only 0-edges** keep variable ordering.
  - 1-edges has no restriction.
  - Still unique representation for a given set of sequences.
  - Each path from root to 1-terminal corresponds to a sequence.

(Ordered) ZDD  
Sequence BDD
Basic operations of sequence family algebra

Table 3: Algebraic operations for SeqBDDs

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>Returns empty set. (0-terminal node)</td>
</tr>
<tr>
<td>${\lambda}$</td>
<td>Returns the set of only null-combination. (1-terminal node)</td>
</tr>
<tr>
<td>$P.top$</td>
<td>Returns item-ID at root node of $P$.</td>
</tr>
<tr>
<td>$P.onset(x)$</td>
<td>Selects the subset of sequences that begin with letter $x$, and then removes $x$ from the head of each sequence.</td>
</tr>
<tr>
<td>$P.offset(x)$</td>
<td>Selects the subset of sequences that do not begin with letter $x$.</td>
</tr>
<tr>
<td>$P.push(x)$</td>
<td>Appends $x$ to the head of every sequence in $P$.</td>
</tr>
<tr>
<td>$P \cup Q$</td>
<td>Returns union set.</td>
</tr>
<tr>
<td>$P \cap Q$</td>
<td>Returns intersection set.</td>
</tr>
<tr>
<td>$P \setminus Q$</td>
<td>Returns difference set. (in $P$ but not in $Q$.)</td>
</tr>
<tr>
<td>$P.count$</td>
<td>Counts number of combinations.</td>
</tr>
<tr>
<td>$P \times Q$</td>
<td>Cartesian product of $P$ and $Q$. (Concatenations of all pairs in $P$ and in $Q$)</td>
</tr>
</tbody>
</table>

- ZDD-like algebraic operations.
  - onset, offset, and push operations are different.
  - Other operations are almost same.
SeqBDD construction (applying basic operations)

```
\{\lambda\} \quad \{a\} \quad \{ba, b\} \\
1 \quad a \quad b \\
\{a, \lambda\} \quad \{a, \lambda\} \quad \{ba, a, b\} \\
\{a\} \quad \{a\} \quad \{ba, a, b\} \\
0 \quad b \quad 0 \\
\{\lambda\} \quad \{ba, b\} \quad \{aba, aa, ab\} \\
1 \quad a \quad 1 \\
```

Push\((a)\)

Union

Push\((b)\)

Union

Push\((a)\)
Correspondence of Trie and SeqBDD

Trie

\{aaa, aba, bbc, bc\}

SeqBDD equivalent to trie

\{aaa, aba, bbc, bc\}

Reduced SeqBDD

\{aaa, aba, bbc, bc\}
Trie and SeqBDD

- Trie:
  - Tree-structured lexicographical index of sequences
  - A label of a letter for each branch
  - A path from root to leaf corresponds to a sequence

- In principle, a SeqBDD is equivalent to a trie with sharing their sub-trees.
  - “DAWG”, which is similar idea, was already proposed in the area of string processing.
  - DAWG is only for indexing. **SeqBDD supports not only indexing but also set operations between SeqBDDs.**
Manipulating permutations

- Rubik’s cube: Let \( P = \{ \pi \mid \text{any primitive move of cube.} \} \)
  - \( P \) includes 18 (= 3 ways \( \times \) 6 faces) permutations.
  - Cartesian product \( P \times P \) represents all possible patterns obtained by twice of primitive moves.
  - \( P^{20} \) will have all possible patterns. (but maybe too large.)
- 15 puzzle / Card games
  - Optimization of packing / arranging strategy
- “Amida”-drawing (primitive sorting networks)
  - One-to-one matching problems between two parties.
  - Any bijective relation corresponds to a permutation.
- Design of loss-less codes.
  - Analysis of reversible logic. (related to quantum logic circuit.)
Sets of permutations

- Sets of combinations: (→ ordinary BDDs/ZDDs)
  - Don’t consider order and duplication of items
  - “abcc” and “bca” are the same as “abc”.
- Sets of sequences: (→ SeqBDDs)
  - Distinguishes all finite sequences.
  - \{\lambda\}, \{ab, aba, bbc\}, \{a, aa, aaa, aaaa\}, etc.
- Sets of permutations: (→ \(\pi\)DD)
  - Set of orders in a fixed number of items.
  - \(\varphi\), \{123\}, \{12, 21\}, \{123456, 132456, 246135\}
Decomposition of permutation

- Transpositions $\tau_{(x,y)}$: exchange of two items $x$ and $y$.
- Any $n$-item permutation $\pi$ can be decomposed by at most $(n-1)$ transpositions.

Let $x = \text{dim}(\pi)$, then $\pi \tau_{(x,x\pi)}$ must not move $x$. Thus, $\text{dim}(\pi \tau_{(x,x\pi)}) \leq \text{dim}(\pi) - 1$

$\rightarrow$ Repeating this process provides a decomposed form.

$\pi = (3,5,2,1,4)$
Decomposition of permutation

- Transpositions $\tau_{(x,y)}$: exchange of two items $x$ and $y$.
- Any $n$-item permutation $\pi$ can be decomposed by at most $(n-1)$ transpositions.

Let $x = \dim(\pi)$, then $\pi \tau_{(x,x\pi)}$ must not move $x$. Thus, $\dim(\pi \tau_{(x,x\pi)}) \leq \dim(\pi) - 1$

$\rightarrow$ Repeating this process provides a decomposed form.

$\pi = (3,5,2,1,4) = (3,4,2,1) \tau_{(5,4)}$

![Diagram showing the decomposition of a permutation using transpositions]
Decomposition of permutation

- Transpositions \( \tau_{(x,y)} \): exchange of two items \( x \) and \( y \).
- Any \( n \)-item permutation \( \pi \) can be decomposed by at most \( (n-1) \) transpositions.

Let \( x = \text{dim}(\pi) \), then \( \pi \tau_{(x,x\pi)} \) must not move \( x \).
Thus, \( \text{dim}(\pi \tau_{(x,x\pi)}) \leq \text{dim}(\pi) - 1 \)

→ Repeating this process provides a decomposed form.

\[ \pi = (3,5,2,1,4) = (3,1,2) \tau_{(4,1)} \tau_{(5,4)} \]
Decomposition of permutation

- Transpositions $\tau_{(x,y)}$: exchange of two items $x$ and $y$.
- Any $n$-item permutation $\pi$ can be decomposed by at most $(n-1)$ transpositions.

Let $x = \text{dim}(\pi)$, then $\pi \tau_{(x,x\pi)}$ must not move $x$. Thus, $\text{dim}(\pi \tau_{(x,x\pi)}) \leq \text{dim}(\pi) - 1$

$\rightarrow$ Repeating this process provides a decomposed form.

$\pi = (3,5,2,1,4) = \tau_{(2,1)} \tau_{(3,2)} \tau_{(4,1)} \tau_{(5,4)}$

Deterministic process.
$\rightarrow$ canonical form for any given $\pi$. 
Main idea of $\pi$DDs

- Using a pair of IDs ($x, y$) for each decision node.

Let $x = \text{dim}(P)$, and $x > y > 0$

$$P = P_0 \cup P_1 \tau_{(x,y)}$$

$$P_0 = \{ \pi \mid \pi \in P, x\pi \neq y \}$$

$$P_1 = \{ \pi\tau_{(x,y)} \mid \pi \in P, x\pi = y \}$$

$\text{dim}(P_0) \leq \text{dim}(P)$

$\text{dim}(P_1) < \text{dim}(P)$
Node reduction rules for $\pi$DDs

- Same reduction rules as ZDDs.
  - Ordinary BDD rules don’t work.
πDDs of single permutation

\[ \varphi \{ \pi_e \} \]

\[
\begin{align*}
0 & \rightarrow 1 \\
(2,1) & \rightarrow 3,2
\end{align*}
\]

\[
\begin{align*}
(1,3,2) & \rightarrow (3,2)
\end{align*}
\]

\[
\begin{align*}
(3,1,2) & \rightarrow (3,1)
\end{align*}
\]

\[
\begin{align*}
(3,2,1) & \rightarrow (2,3,1)
\end{align*}
\]

\[
\begin{align*}
(2,3,1) & \rightarrow 3,2
\end{align*}
\]

\[
\begin{align*}
3 & \rightarrow 3 \\
2 & \rightarrow 2 \\
1 & \rightarrow 1
\end{align*}
\]

2017.12.20

Large-Scale Knowledge Processing
**π**DDs for sets of permutations

\[
\{ (2,1) \} \\
\{ \pi_e, (2,1) \} \\
\{ (2,1) \}
\]

\[
\{ (2,1), (3,1,2) \} \\
\{ (2,1), (3,1,2) \}
\]

\[
\{ \pi_e, (2,1), (1,3,2), (3,1,2), (3,2,1), (2,3,1) \} \\
\{ \pi_e, (2,1), (1,3,2), (3,1,2) \}
\]

\[
\{ (2,1), (3,1,2), (3,2,1), (2,3,1) \} \\
\{ (2,1), (3,1,2) \}
\]

\[
\{ \pi_e, (2,1), (1,3,2), (3,1,2), (3,2,1), (2,3,1) \} \\
\{ \pi_e, (2,1) \}
\]

\[
\{ (2,1), (3,1,2), (3,2,1), (2,3,1) \} \\
\{ (2,1), (3,1,2) \}
\]

\[
\{ \pi_e, (2,1) \}
\]
Algebraic operations for \( \pi \text{DDs} \)

- "Permutation family algebra"

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>Returns the empty set. (0-terminal node)</td>
</tr>
<tr>
<td>( {\pi_e} )</td>
<td>Returns the singleton set. (1-terminal node)</td>
</tr>
<tr>
<td>( P.\text{top} )</td>
<td>Returns the IDs ((x, y)) at the root node of ( P ).</td>
</tr>
<tr>
<td>( P \cup Q )</td>
<td>Returns ( {\pi \mid \pi \in P \text{ or } \pi \in Q} ).</td>
</tr>
<tr>
<td>( P \cap Q )</td>
<td>Returns ( {\pi \mid \pi \in P, \pi \in Q} ).</td>
</tr>
<tr>
<td>( P \setminus Q )</td>
<td>Returns ( {\pi \mid \pi \in P, \pi \notin Q} ).</td>
</tr>
<tr>
<td>( P.\tau(x, y) )</td>
<td>Returns ( P \cdot \tau(x, y) ).</td>
</tr>
<tr>
<td>( P \ast Q )</td>
<td>Returns ( {\alpha \beta \mid \alpha \in P, \beta \in Q} ).</td>
</tr>
<tr>
<td>( P.\text{cofact}(x, y) )</td>
<td>Returns ( {\pi \tau(x, y) \mid \pi \in P, x\pi = y} ).</td>
</tr>
<tr>
<td>( P.\text{count} )</td>
<td>Returns the number of permutations.</td>
</tr>
</tbody>
</table>
Synthesis of $\pi$DDs by algebraic operations

\[
\begin{align*}
\{ \pi_e \} & \xrightarrow{\tau_{(3,2)}} \{ (1,3,2) \} \\
\{ (2,1) \} & \xrightarrow{\tau_{(2,1)}} \{ \pi_e, (2,1) \} \\
\{ \pi_e, (2,1) \} & \xrightarrow{\text{union}} \{ \pi_e, (2,1), (1,3,2) \} \\
\{ \pi_e, (2,1), (1,3,2) \} & \xrightarrow{\text{difference}} \{ \pi_e, (2,1), (1,3,2), (3,1,2) \} \\
\end{align*}
\]
Product operation for disjoint permutations

\[ \{\pi_e, (1,2,3,5,4), (1,2,3,6,5,4), (1,2,3,5,6,4), (1,2,3,4,6,5), (1,2,3,6,4,5)\} \]

\[ \{\pi_e, (2,1), (3,2,1), (2,3,1), (1,3,2), (3,1,2)\} \]

\[ \{\pi_e, (1,2,3,5,4), (1,2,3,6,5,4), (1,2,3,5,6,4), (1,2,3,4,6,5), (1,2,3,6,4,5), (2,1,3), (2,1,3,5,4), (2,1,3,6,5,4), (2,1,3,5,6,4), (2,1,3,4,6,5), (2,1,3,6,4,5), (3,2,1), (3,2,1,5,4), (3,2,1,6,5,4), (3,2,1,5,6,4), (3,2,1,4,6,5), (3,2,1,6,4,5), (2,3,1), (2,3,1,5,4), (2,3,1,6,5,4), (2,3,1,5,6,4), (2,3,1,4,6,5), (2,3,1,6,4,5), (1,3,2), (1,3,2,5,4), (1,3,2,6,5,4), (1,3,2,5,6,4), (1,3,2,4,6,5), (1,3,2,6,4,5), (3,1,2), (3,1,2,5,4), (3,1,2,6,5,4), (3,1,2,5,6,4), (3,1,2,4,6,5), (3,1,2,6,4,5)\} \]
For $n$ item permutations, we may layout $k$ primitive (adjacent) swaps. How many $k$ is the minimum for generating a given permutation?

Any permutations of up to $k$ swaps can be computed by the following simple equations. → using $\pi$DDs.

\[
\begin{align*}
P_0 &= \pi_e \\
P_1 &= P_0 \cup \left( \bigcup_{i=1}^{n-1} \tau(i,i+1) \right) \\
P_k &= P_{k-1} \ast P_1 \quad (\text{for } k \geq 2)
\end{align*}
\]
Experiment (1) for primitive sorting networks

- Experimental results for $n = 10$.
  - All 10! permutations are generated by 45 swaps. $\rightarrow (10! - 1)$ with 44 swaps.
  - $\pi$DDs have a peak size at $P_{27}$. Final size is only 45 nodes.

<table>
<thead>
<tr>
<th>$P_k$</th>
<th>$\pi$DD size</th>
<th># of perm.</th>
<th>total $#\tau$</th>
<th>$P_k$</th>
<th>$\pi$DD size</th>
<th># of perm.</th>
<th>total $#\tau$</th>
<th>$P_k$</th>
<th>$\pi$DD size</th>
<th># of perm.</th>
<th>total $#\tau$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$P_{16}$</td>
<td>3956</td>
<td>528441</td>
<td>3412177</td>
<td>$P_{32}$</td>
<td>8655</td>
<td>3497165</td>
<td>24691907</td>
</tr>
<tr>
<td>$P_1$</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>$P_{17}$</td>
<td>4685</td>
<td>690778</td>
<td>4522462</td>
<td>$P_{33}$</td>
<td>7669</td>
<td>3544208</td>
<td>25039740</td>
</tr>
<tr>
<td>$P_2$</td>
<td>31</td>
<td>54</td>
<td>97</td>
<td>$P_{18}$</td>
<td>5455</td>
<td>878737</td>
<td>5821218</td>
<td>$P_{34}$</td>
<td>6590</td>
<td>3576891</td>
<td>25279788</td>
</tr>
<tr>
<td>$P_3$</td>
<td>63</td>
<td>209</td>
<td>546</td>
<td>$P_{19}$</td>
<td>6249</td>
<td>1089826</td>
<td>7296041</td>
<td>$P_{35}$</td>
<td>5470</td>
<td>3598561</td>
<td>25439624</td>
</tr>
<tr>
<td>$P_4$</td>
<td>109</td>
<td>649</td>
<td>2152</td>
<td>$P_{20}$</td>
<td>7047</td>
<td>1319957</td>
<td>8915085</td>
<td>$P_{36}$</td>
<td>4374</td>
<td>3612201</td>
<td>25539440</td>
</tr>
<tr>
<td>$P_5$</td>
<td>172</td>
<td>1717</td>
<td>6704</td>
<td>$P_{21}$</td>
<td>7834</td>
<td>1563651</td>
<td>10645703</td>
<td>$P_{37}$</td>
<td>3353</td>
<td>3620296</td>
<td>25598543</td>
</tr>
<tr>
<td>$P_6$</td>
<td>261</td>
<td>4015</td>
<td>17632</td>
<td>$P_{22}$</td>
<td>8591</td>
<td>1814400</td>
<td>12433871</td>
<td>$P_{38}$</td>
<td>2444</td>
<td>3624785</td>
<td>25630975</td>
</tr>
<tr>
<td>$P_7$</td>
<td>390</td>
<td>8504</td>
<td>40751</td>
<td>$P_{23}$</td>
<td>9293</td>
<td>2065149</td>
<td>14239194</td>
<td>$P_{39}$</td>
<td>1671</td>
<td>3627083</td>
<td>25647411</td>
</tr>
<tr>
<td>$P_8$</td>
<td>558</td>
<td>16599</td>
<td>84985</td>
<td>$P_{24}$</td>
<td>9905</td>
<td>2308843</td>
<td>15996836</td>
<td>$P_{40}$</td>
<td>1055</td>
<td>3628151</td>
<td>25654943</td>
</tr>
<tr>
<td>$P_9$</td>
<td>773</td>
<td>30239</td>
<td>162995</td>
<td>$P_{25}$</td>
<td>10397</td>
<td>2538974</td>
<td>17671711</td>
<td>$P_{41}$</td>
<td>602</td>
<td>3628591</td>
<td>25657983</td>
</tr>
<tr>
<td>$P_{10}$</td>
<td>1034</td>
<td>51909</td>
<td>291537</td>
<td>$P_{26}$</td>
<td>10735</td>
<td>2750063</td>
<td>19206325</td>
<td>$P_{42}$</td>
<td>305</td>
<td>3628746</td>
<td>25659023</td>
</tr>
<tr>
<td>$P_{11}$</td>
<td>1353</td>
<td>84592</td>
<td>491272</td>
<td>$P_{27}$</td>
<td>10894</td>
<td>2938022</td>
<td>20584666</td>
<td>$P_{43}$</td>
<td>136</td>
<td>3628700</td>
<td>25659303</td>
</tr>
<tr>
<td>$P_{12}$</td>
<td>1727</td>
<td>131635</td>
<td>786100</td>
<td>$P_{28}$</td>
<td>10897</td>
<td>3100359</td>
<td>21772380</td>
<td>$P_{44}$</td>
<td>59</td>
<td>3628799</td>
<td>25659355</td>
</tr>
<tr>
<td>$P_{13}$</td>
<td>2169</td>
<td>196524</td>
<td>1201963</td>
<td>$P_{29}$</td>
<td>10614</td>
<td>3236212</td>
<td>22773147</td>
<td>$P_{45}$</td>
<td>45</td>
<td>3628800</td>
<td>25659360</td>
</tr>
<tr>
<td>$P_{14}$</td>
<td>2688</td>
<td>282578</td>
<td>1764353</td>
<td>$P_{30}$</td>
<td>10157</td>
<td>3346222</td>
<td>23579581</td>
<td>$P_{46}$</td>
<td>45</td>
<td>3628800</td>
<td>25659360</td>
</tr>
<tr>
<td>$P_{15}$</td>
<td>3286</td>
<td>392588</td>
<td>2495497</td>
<td>$P_{31}$</td>
<td>9497</td>
<td>3432276</td>
<td>24214975</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experiment (2) for primitive sorting networks

- Experimental result for $n$ lines of primitive sorting networks.
  - Determined the minimum swaps $m$ to generate all $n!$ permutations.
  - High compression rate (>10000 times) in $\pi$DD size (peak and final).

<table>
<thead>
<tr>
<th>$n$</th>
<th>$m$</th>
<th>$\pi$DD size</th>
<th>$\pi$DD size</th>
<th># of perm.</th>
<th>total $#\tau$</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(peak)</td>
<td>(final)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>24</td>
<td>46</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>27</td>
<td>10</td>
<td>120</td>
<td>326</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>89</td>
<td>15</td>
<td>720</td>
<td>2556</td>
<td>0.01</td>
</tr>
<tr>
<td>7</td>
<td>21</td>
<td>292</td>
<td>21</td>
<td>5040</td>
<td>22212</td>
<td>0.02</td>
</tr>
<tr>
<td>8</td>
<td>28</td>
<td>972</td>
<td>28</td>
<td>40320</td>
<td>212976</td>
<td>0.06</td>
</tr>
<tr>
<td>9</td>
<td>36</td>
<td>3241</td>
<td>36</td>
<td>362880</td>
<td>2239344</td>
<td>0.26</td>
</tr>
<tr>
<td>10</td>
<td>45</td>
<td>10894</td>
<td>45</td>
<td>3628800</td>
<td>25659360</td>
<td>1.19</td>
</tr>
<tr>
<td>11</td>
<td>55</td>
<td>36906</td>
<td>55</td>
<td>39916800</td>
<td>318540960</td>
<td>5.77</td>
</tr>
<tr>
<td>12</td>
<td>66</td>
<td>125904</td>
<td>66</td>
<td>479001600</td>
<td>4261576320</td>
<td>27.06</td>
</tr>
<tr>
<td>13</td>
<td>78</td>
<td>435221</td>
<td>78</td>
<td>6227020800</td>
<td>61148511360</td>
<td>126.80</td>
</tr>
<tr>
<td>14</td>
<td>91</td>
<td>1520439</td>
<td>91</td>
<td>87178291200</td>
<td>937030429440</td>
<td>666.29</td>
</tr>
</tbody>
</table>
Analysis of Rubik’s cube

- Here, we focus only the moves of eight corner cubes.
  - After solving the corner cubes, we can solve the other cubes independently.
- Primitive turns along X-, Y-, and Z-axis can be written as follows.

\[
\begin{align*}
\pi_x &= T(3,5)T(3,17)T(3,15)T(1,6)T(1,16)T(1,14)T(2,4)T(2,18)T(2,13) \\
\pi_y &= T(2,14)T(2,24)T(2,12)T(3,13)T(3,23)T(3,10)T(1,15)T(1,22)T(1,11) \\
\pi_z &= T(1,10)T(1,7)T(1,4)T(3,12)T(3,9)T(3,6)T(2,11)T(2,8)T(2,5)
\end{align*}
\]
Analysis of Rubik’s cube

- Here, we focus only the moves of eight corner cubes.
  - After solving the corner cubes, we can solve the other cubes independently.
- Primitive turns along X-, Y-, and Z-axis can be written as follows.

\[
\begin{align*}
\pi_x &= T_{(3,5)} T_{(3,17)} T_{(3,15)} T_{(1,6)} T_{(1,16)} T_{(1,14)} T_{(2,4)} T_{(2,18)} T_{(2,13)} \\
\pi_y &= T_{(2,14)} T_{(2,24)} T_{(2,12)} T_{(3,13)} T_{(3,23)} T_{(3,10)} T_{(1,15)} T_{(1,22)} T_{(1,11)} \\
\pi_z &= T_{(1,10)} T_{(1,7)} T_{(1,4)} T_{(3,12)} T_{(3,9)} T_{(3,6)} T_{(2,11)} T_{(2,8)} T_{(2,5)}
\end{align*}
\]
Analysis of Rubik’s cube

- Here, we focus only the moves of eight corner cubes.
  - After solving the corner cubes, we can solve the other cubes independently.

- Primitive turns along X-, Y-, and Z-axis can be written as follows.

\[
\begin{align*}
\pi_x & = \mathcal{T}(3,5)\mathcal{T}(3,17)\mathcal{T}(3,15)\mathcal{T}(1,6)\mathcal{T}(1,16)\mathcal{T}(1,14)\mathcal{T}(2,4)\mathcal{T}(2,18)\mathcal{T}(2,13) \\
\pi_y & = \mathcal{T}(2,14)\mathcal{T}(2,24)\mathcal{T}(2,12)\mathcal{T}(3,13)\mathcal{T}(3,23)\mathcal{T}(3,10)\mathcal{T}(1,15)\mathcal{T}(1,22)\mathcal{T}(1,11) \\
\pi_z & = \mathcal{T}(1,10)\mathcal{T}(1,7)\mathcal{T}(1,4)\mathcal{T}(3,12)\mathcal{T}(3,9)\mathcal{T}(3,6)\mathcal{T}(2,11)\mathcal{T}(2,8)\mathcal{T}(2,5) 
\end{align*}
\]
Analysis of Rubik’s cube

- Here, we focus only the moves of eight corner cubes.
  - After solving the corner cubes, we can solve the other cubes independently.
- Primitive turns along X-, Y-, and Z-axis can be written as follows.

\[
\begin{align*}
\pi_x &= T(3,5)T(3,17)T(3,15)T(1,6)T(1,16)T(1,14)T(2,4)T(2,18)T(2,13) \\
\pi_y &= T(2,14)T(2,24)T(2,12)T(3,13)T(3,23)T(3,10)T(1,15)T(1,22)T(1,11) \\
\pi_z &= T(1,10)T(1,7)T(1,4)T(3,12)T(3,9)T(3,6)T(2,11)T(2,8)T(2,5)
\end{align*}
\]
Analysis of Rubik’s cube

- Here, we focus only the moves of eight corner cubes.
  - After solving the corner cubes, we can solve the other cubes independently.

- Primitive turns along X-, Y-, and Z-axis can be written as follows.

\[
\pi_x = \mathcal{T}(3,5)\mathcal{T}(3,17)\mathcal{T}(3,15)\mathcal{T}(1,6)\mathcal{T}(1,16)\mathcal{T}(1,14)\mathcal{T}(2,4)\mathcal{T}(2,18)\mathcal{T}(2,13)
\]
\[
\pi_y = \mathcal{T}(2,14)\mathcal{T}(2,24)\mathcal{T}(2,12)\mathcal{T}(3,13)\mathcal{T}(3,23)\mathcal{T}(3,10)\mathcal{T}(1,15)\mathcal{T}(1,22)\mathcal{T}(1,11)
\]
\[
\pi_z = \mathcal{T}(1,10)\mathcal{T}(1,7)\mathcal{T}(1,4)\mathcal{T}(3,12)\mathcal{T}(3,9)\mathcal{T}(3,6)\mathcal{T}(2,11)\mathcal{T}(2,8)\mathcal{T}(2,5)
\]

All patterns by one move.

\[
P_1 = \pi_e + \pi_x + \pi_x^2 + \pi_x^3 + \pi_y + \pi_y^2 + \pi_y^3 + \pi_z + \pi_z^2 + \pi_z^3
\]
Analysis of Rubik’s cube

- Here, we focus only the moves of eight corner cubes.
  - After solving the corner cubes, we can solve the other cubes independently.

- Primitive turns along X-, Y-, and Z-axis can be written as follows.
  \[ \pi_x = \tau(3,5)\tau(3,17)\tau(3,15)\tau(1,6)\tau(1,16)\tau(1,14)\tau(2,4)\tau(2,18)\tau(2,13) \]
  \[ \pi_y = \tau(2,14)\tau(2,24)\tau(2,12)\tau(3,13)\tau(3,23)\tau(3,10)\tau(1,15)\tau(1,22)\tau(1,11) \]
  \[ \pi_z = \tau(1,10)\tau(1,7)\tau(1,4)\tau(3,12)\tau(3,9)\tau(3,6)\tau(2,11)\tau(2,8)\tau(2,5) \]

All patterns by one move.

\[ P_1 = \pi_e + \pi_x + \pi_x^2 + \pi_x^3 + \pi_y + \pi_y^2 + \pi_y^3 + \pi_z + \pi_z^2 + \pi_z^3 \]

All patterns by \( k \) moves.

\[ P_k = P_{k-1} \times P_1 \quad (\text{for } k \geq 2) \]
Experimental result for Rubik’s cube

- If focusing only corner cubes, 11-moves generates all patterns.
  - Only 511 $\pi$DD nodes for all patterns.

- After generating $P_k$, we can analyze various properties of Rubik’s cube.
  - The patterns such that only 2 cubes moved but other 6 cubes remain can be written as:

$$S_k = P_k \cdot \text{cofact}(9, 9) \cdot \text{cofact}(11, 11) \cdot \text{cofact}(15, 15) \cdot \text{cofact}(17, 17) \cdot \text{cofact}(21, 21) \cdot \text{cofact}(23, 23)$$

  $\rightarrow$ 4 patterns in $k=10$, and 5 patterns in $k=11$.

(The sequence of moves can be obtained by $\pi$DD operations.)
Discussion on solving Rubik’s cube

- Rokicki et al. confirmed that all $3 \times 3$ Rubik’s cube can be solved as few as 20 moves. [Rokicki et al. 2010]
  - They don’t enumerate all minimum number of moves. They just checked no patterns require more than 20 moves.
  - At first they applied mathematical pruning, and then explored all patterns by using large-scale PC cloud. (total 35 CPU years)

- Straight-forward application of $\pi$DD to $3 \times 3$ Rubik’s cube, might cause memory overflow.
  - $\pi$DD generates the minimum moves for all patterns, so the computation is stronger than Rokicki’s result.
  - Not using mathematical pruning, only with $\pi$DD’s compression.
  - Using one CPU only.
  - Our method is very flexible to modify the problem regulation. (Definition of primitive moves, restriction of sequences, etc.)
πDD sizes for typical cases

- $\varnothing, \{ e \} : O(1)$ nodes
- Sets of a single permutation with $n$ items: $O(n)$ nodes
- Sets of any $k$ permutations with $n$ items: $O(kn)$ nodes
- Sets of all $n$ rotations with $n$ items: $O(n^2)$ nodes
- Sets of all $n!$ permutations with $n$ items: $O(n^2)$ nodes

- Nodes for each permutation is bounded by “swap distance” from identical permutation.
- $\pi$DD can be compact for representing the family consists of many similar sub-permutations.
Upper bound of $\pi$DD sizes

- Number of Families of permutations up to $n$ items: $2^n!$
  - $n$: 0, 1, 2, 3, 4, 5, …
  - $n!$: 1, 1, 2, 6, 24, 120, …
  - $2^n!$: 2, 2, 4, 64, 16777216, 1329227995784915872903807060280344576, …

- At least $\log n$ bit needed to distinguish $n$ objects.
  - Thus, $\pi$DD size must be $O(n!)$ bit in general.
Future Work on $\pi$DDs

- Research on $\pi$DDs is still ongoing.
- A lot of future work remains.
  - Analyzing more precise complexity for $\pi$DD operations
  - Developing a tool for easily manipulating permutation groups
  - Many practical applications using sets of permutations
  - Extension of permutation family algebra. (division, etc.)
  - $k$-out-of $n$ permutations,
  - multiset of permutations,
  - permutation of multiset items, etc.
Summary

ZDD extensions for sequences and permutations

- Basic operations of ZDDs (review)
- Basic operations of ZDD-vectors (review)
- Sequence BDDs for manipulating sets of sequences
  - Sets of sequences
  - Encoded ZDDs for representing sets of sequences
  - SeqBDDs and their basic operations
  - Tries and SeqBDDs
- Permutation decision diagrams ($\pi$DDs)
  - Sets of permutations
  - Permutation represented by a set of transpositions
  - $\pi$DDs and their basic operations
  - Applications
Exercises

- Draw the SeqBDDs for the following sets of sequences.

  - \{ \lambda, a, aa, aaa, aaaa \}
  - \{ aaa, aab, aba, abb, baa, bab, bba, bbb \}
  - \{ a, ab, abb, abbb, b, ba, baa, baaa \}