HOKKAIDO UNIVERSITY

Title	πDD: Permutation Decision Diagram based on Permutation Family Algebra
Author(s)	Minato, Shin-ichi
Description	ERATO湊離散構造処理系プロジェクト ： 2010年度初冬のワークショップ（ERATO合宿）. 2010年11月29日（月）～12月1日（水）. 札幌北広島クラッセホテル.
Relation	2010年度科学技術振興機構ERATO湊離散構造処理系プロジェクト講究録. p.389-394.
Issue Date	2011-06
Doc URL	https://hdl.handle.net/2115/48359
Type	conference presentation
File Information	01.06_PiDD.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

πDD: Permutation Decision Diagram based on Permutation Family Algebra

Shin-ichi Minato
Hokkaido University / JST ERATO

Direction of our research

Primary output

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

Still many applications remains where ZDDs would be effective.

(Combinatorial)

(Higher model)

Further outputs

- Multisets
- Sequences
- Permutations
- Partitions
- Trees, DAGs
- Networks
- etc.

Advanced ZDD-like structure → Applications with higher data model
Sequence data analysis
Numerical data processing
Processing of trees or semi-structured data

Develop special new algebraic operations.

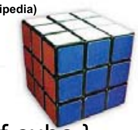
Background

- BDD: Boolean function
  - Boolean algebra
- ZDD: Family of sets
  - Family algebra
- ZDD-Vector: Histogram of itemsets
  - Itemset histogram algebra
- Sequence BDD: Family of sequences
  - Sequence family algebra
- πDD: Family of permutations
  - Permutation family algebra

Family of permutations

- Family of sets:
  - Don't consider order and duplication of items
  - "abcc" and "bca" are the same.
- Family of sequences:
  - Distinguishes all finite sequences.
  - φ, {λ}, { ab, aba, bbc }, { a, aa, aaa, aaaa }, etc.
- Family of permutations:
  - Set of orders in a fixed number of items.
  - φ, { 123 }, {12, 21}, { 123456, 132456, 246135 }

Applications



(© Wikipedia)

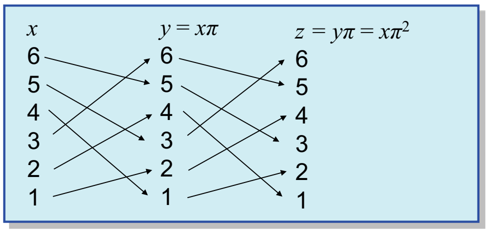
- Rubik's cube: Let P = { π | any primitive move of cube.}
  - P includes 12 (= 2 ways × 6 faces) permutations.
  - Cartesian product $P \times P$ represents all possible patterns obtained by twice of primitive moves.
  - P^{20} will have all possible patterns. (but maybe too large.)
- 15 puzzle, Tower of Hanoi
  - Optimization of packing / arranging strategy
- "Amida-drawing" (rudder-style swapping graph)
  - One-to-one matching problems between two parties.
  - A permutation corresponds to a bijective relation.
- Design of loss-less codes.
  - Analysis of reversible logic. (related to quantum logic circuit.)

(© Wikipedia)

Permutations

- Notation of permutation is often confusing.

(ex.) π = "246135" (≠ "415263")



$x\pi$	$x\pi^2$
5	3
3	6
1	2
6	5
4	1
2	4

Required properties for πDDs

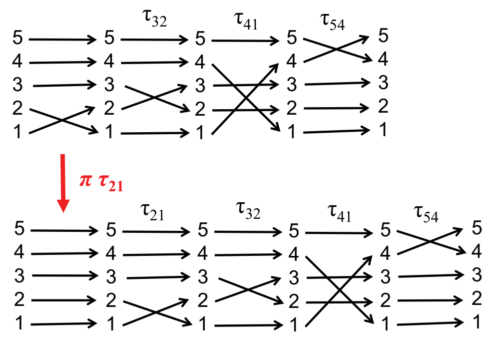
- Empty set φ should be 0-terminal node.
- Singleton set of the identical permutation: { "123456789…" } should be 1-terminal node.
  - We may write { e } since we don't have to consider the dimension (number of items) for the identical relation.
- { "132", "321" } and {"132456789", "321456789" } had better be represented in a same DD.
  - "Dimension of permutation" $Dim(\pi)$ is defined as the largest ID relevant to the permutation. (We put $Dim(e) = 0$.)
  - "Dimension of family of permutations" $Dim(P)$ is the largest dimension of permutation in the family. (We put $Dim(\varphi)$=0.) → $Dim(P)$ should be the top-ID of πDD for P.

Required properties for πDDs (cont.)

- Each path from root node to 1-terminal node should correspond to a permutation in P.
  - Number of paths equals the cardinality of P.
- Giving of canonical (unique) representation for a family of permutations.
  - Efficient equivalence checking
- ZDD-like algebraic operations over πDDs.
  - Computation time depends on πDD size, not directly depend on cardinality of P.

Decomposition of permutation by τ_{xy}

π = "35214"



τ_{xy} (transposition)

$\begin{cases} x \to y \\ y \to x \end{cases}$

$\pi\ \tau_{xy}$ where $x\pi = y$

$\begin{cases} x \to y \\ z \to x \end{cases} \Rightarrow \begin{cases} x \to x \\ z \to y \end{cases}$

Any n-item permutation can be decomposed by at most $(n-1)$ transpositions.

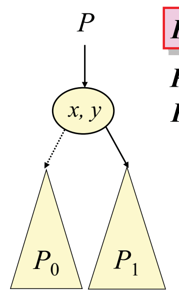
π = "35214" = $e\ \tau_{21}\ \tau_{32}\ \tau_{41}\ \tau_{54}$

→ canonical form

Main idea of πDDs

- **Using a pair of IDs for each decision node.**

Let x as $Dim(P)$, and $x > y > 0$



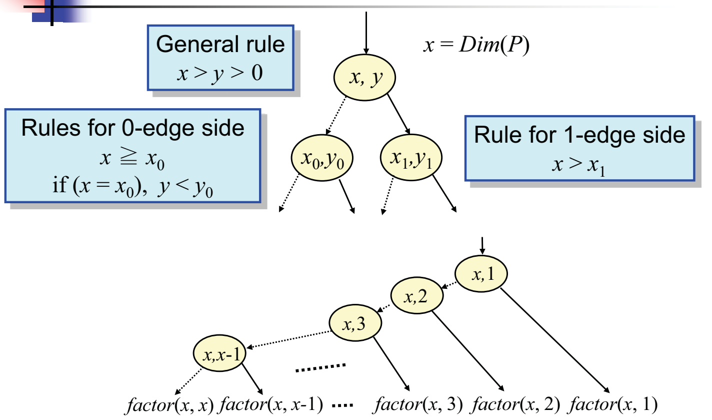
$\boldsymbol{P = P_0 \cup P_1 \tau_{xy}}$

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

$\boldsymbol{P_1 = \{ \pi \in (P\ \tau_{xy}) \mid x\pi = x \}}$

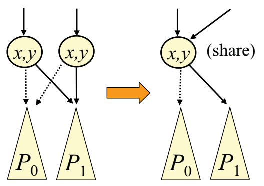
→ $Dim(P_0) \leqq Dim(P)$
$Dim(P_1) < Dim(P)$

Rule of variable ordering in πDDs

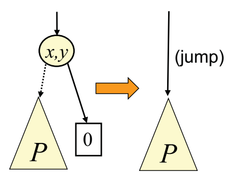


Node reduction rules for πDDs

- Same reduction rules as ZDDs.
  - Ordinary BDD rules don't work.

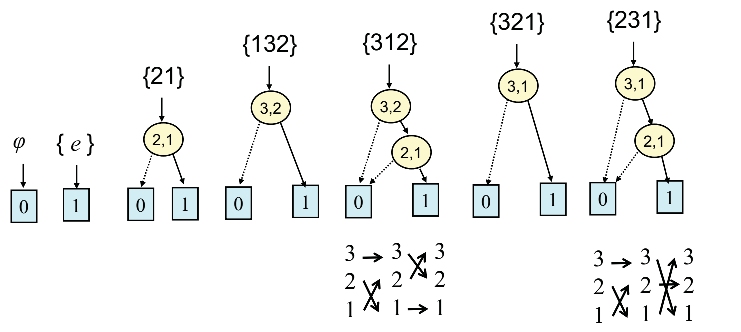


Node sharing

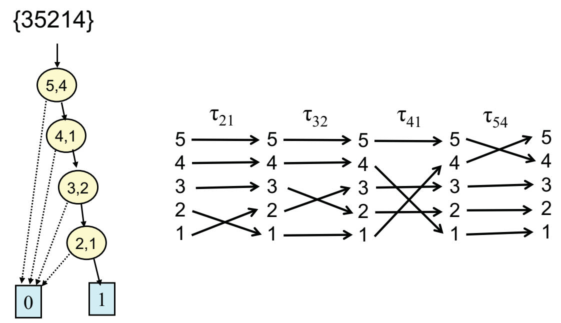


Zero-suppressed node elimination

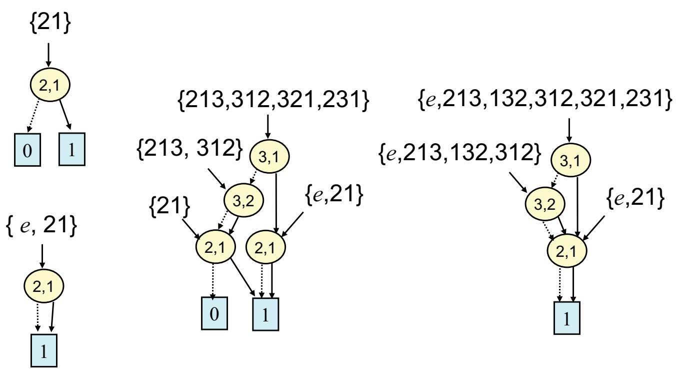
πDDs of single permutation



πDDs of single permutation



πDDs for sets of permutations



Related work in Knuth-book

The idea is to take such a sequence for $\{1,\ldots,n-1\}$ and to insert the number n into each permutation in all ways. For example, if $n = 4$ the sequence (123, 132, 312, 321, 231, 213) leads to the columns of the array

1234	1324	3124	3214	2314	2134	
1243	1342	3142	3241	2341	2143	(3)
1423	1432	3412	3421	2431	2413	
4123	4132	4312	4321	4231	4213	

when 4 is inserted in all four possible positions. Now we obtain the desired sequence by reading downwards in the first column, upwards in the second, downwards in the third, ..., upwards in the last: (1234, 1243, 1423, 4123, 4132, 1432, 1342, 1324, 3124, 3142, ..., 2143, 2134).

In Section 5.1.1 we studied the inversions of a permutation, namely the pairs of elements (not necessarily adjacent) that are out of order. Every interchange of adjacent elements changes the total number of inversions by ± 1. In fact, when we consider the so-called inversion table $c_1 \ldots c_n$ of exercise 5.1.1–7, where c_j is the number of elements lying to the right of j that are less than j, we find that the permutations in (3) have the following inversion tables:

0000	0010	0020	0120	0110	0100	
0001	0011	0021	0121	0111	0101	(4)
0002	0012	0022	0122	0112	0102	
0003	0013	0023	0123	0113	0103	

And if we read these columns alternately down and up as before, we obtain precisely the reflected Gray code for mixed radices (1, 2, 3, 4), as in Eqs. (46)–(51)

7.2.1.2. Generating all permutations. (Vol. 4. Fascicle 2)

"Inversion table." Each permutation can be represented as combinations.

Related work in Knuth-book (cont.)

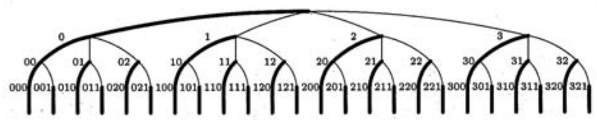


Fig. 19. Algorithm G implicitly traverses this tree when $n = 4$.

The tree in Fig. 19 illustrates Algorithm G in the case $n = 4$. According to (17), every permutation $a_0a_1a_2a_3$ of $\{0, 1, 2, 3\}$ corresponds to a three-digit control string $c_3c_2c_1$, with $0 \le c_3 \le 3$, $0 \le c_2 \le 2$, and $0 \le c_1 \le 1$. Some nodes

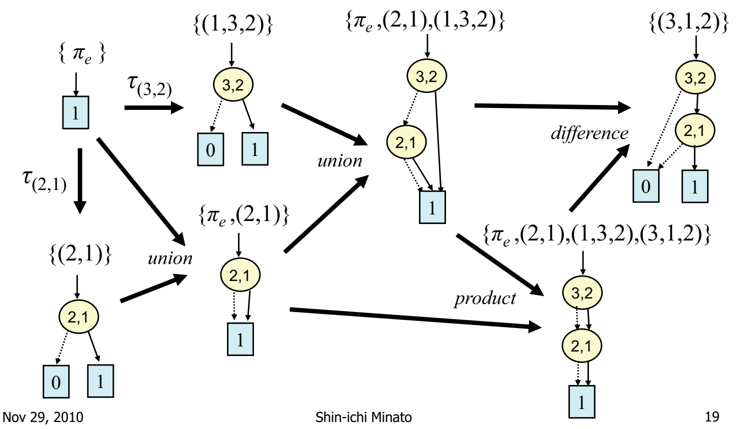
πDD has a strong relationship with Knuth's tree structure for generating all permutations.

Algebraic operations for πDDs

- "Permutation family algebra"

φ, $\{e\}$	***Empty*** and ***identical permutation***. (0/1-terminal)
P.topx	Returns the ***dimension*** of P. (item ID x of the root node)
P.topy	Returns the ***largest ID*** with P.topx. (item ID y of the root node)
P.factor(x, y)	Returns $\{ \pi \in (P\,\tau_{xy}) \mid x\pi = x \}$
$P\,\tau_{xy}$	Returns $P\,\tau_{xy}$
∪, ∩, \	Returns ***union***, ***intersection***, and ***difference set***.
P.count	***Counts number*** of combinations in P.
$P * Q$	***Cartesian product set*** of P and Q.
P / Q	***Quotient set*** of P divided by Q. (Right-side division)
P % Q	***Remainder set*** of P divided by Q. (Right-side division)

Synthesis of πDDs by algebraic operations



Binary set operations between πDDs

- P_0 and $P_1\tau_{xy}$ are disjoint.
- ∩, ∪, \ operations are independent of τ_{xy} operation.
  → Those operation can be done recursively as well as ordinary BDDs/ZDDs.

$$P \cap Q = (P_0 \cup P_1\tau_{xy}) \cap (Q_0 \cup Q_1\tau_{xy})$$
$$= (P_0 \cap Q_0) \cup (P_1 \cap Q_1)\,\tau_{xy}$$

Recursive algorithm.

Cartesian product operation

- $P * Q = \{\pi_p\pi_q \mid \forall\pi_p \in P, \forall\pi_q \in Q\}$.
  - Not independent of τ_{xy} operation.

$$P * Q = (P_0 \cup P_1\tau_{xy}) * (Q_0 \cup Q_1\tau_{xy})$$
$$= (P_0 * Q_0) \cup (P_0 * Q_1\tau_{xy})$$
$$\cup (P_1\tau_{xy} * Q_0) \cup (P_1\tau_{xy} * Q_1\tau_{xy})$$
$$= (P_0 * Q_0) \cup (P_0 * Q_1)\tau_{xy}$$
$$\cup (P_1\tau_{xy} * Q_0) \cup (P_1\tau_{xy} * Q_1)\tau_{xy}$$

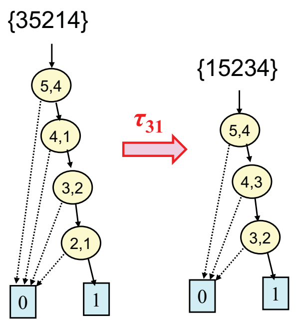
Cannot derive simple recursive algorithm.

$\pi\,\tau_{xy}$ operation

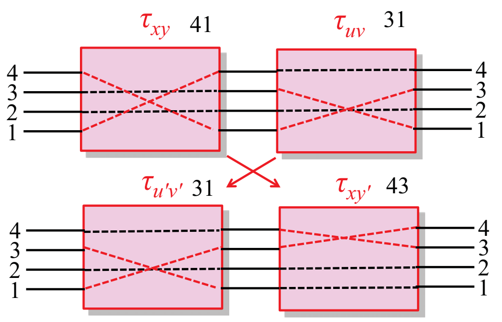
Let $\pi = (35214) = \tau_{21}\tau_{32}\tau_{41}\tau_{54}$

$\pi\,\tau_{31}$
$= (35214)\,\tau_{31}$
$= (15234)$
$= \tau_{32}\tau_{43}\tau_{54}$

$\pi\,\tau_{31}$
$=(\tau_{21}\tau_{32}\tau_{41}\tau_{54})\tau_{31}$
$=(\tau_{21}\tau_{32}\tau_{41})(\tau_{54}\tau_{31})$
$=(\tau_{21}\tau_{32}\tau_{41})(\tau_{31}\tau_{54})$
$=(\tau_{21}\tau_{32})(\tau_{41}\tau_{31})\tau_{54}$
$=(\tau_{21}\tau_{32})(\tau_{31}\tau_{43})\tau_{54}$
$=(\tau_{21})(\tau_{32}\tau_{31})\tau_{43}\tau_{54}$
$=(\tau_{21})(\tau_{21}\tau_{32})\tau_{43}\tau_{54}$
$=(\tau_{21}\tau_{21})\tau_{32}\tau_{43}\tau_{54}$
$=\tau_{32}\tau_{43}\tau_{54}$



Swap of cascaded $\tau_{xy}\tau_{uv}$



If $(x>u>y=v)$
$y' = u$
$u' = u,$
$v' = v$

We can keep canonical form: $u' < x$

Rules to swap $\tau_{xy}\tau_{uv}$ to $\tau_{u'v'}\tau_{xy'}$

if $(u<v)$	(consider $\tau_{xy}\tau_{vu}$)	
if $(x<u$ or $u=v)$	(no swap needed)	
if $(x>y=u>v)$	$y'=v,\ \ u'=u$	
if $(x>u>y=v)$	$y'=u,\ \ u'=u$	
if $(x=u>y>v)$	$y'=y,\ \ u'=y$	
if $(x=u>y=v)$	$y'=u,\ \ u'=y$	(disappear)
otherwise	$y'=y,\ \ u'=u$	(no change)

$y' = y\,\tau_{uv}$

if $(y=u)$	$y'=v$
else if $(y=v)$	$y'=u$
else	$y'=y$

if $(x=u)$	$u'=y$
else	$u'=u$

Algorithm of $(P\,\tau_{uv})$

if $(u=v)$ return P
if $(u<v)$ return $P\,\tau_{vu}$

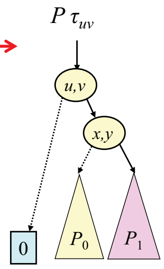
$$P\,\tau_{uv} = (P_0 \cup P_1\tau_{xy})\,\tau_{uv}$$

if $(x<u)$

if $(x \geqq u)$

$$P\,\tau_{uv} = P_0\,\tau_{uv} \cup P_1\,(\tau_{xy}\,\tau_{uv}) = (P_0\,\tau_{uv}) \cup (P_1\,\tau_{u'v})\,\tau_{xy'}$$

Recursive calls with cache.



Cartesian product operation

- $P * Q = \{\pi_p\pi_q \mid {}^{\forall}\pi_p \in P, {}^{\forall}\pi_q \in Q\}$.
- Now we got a recursive algorithm using operation $(P\,\tau_{uv})$.

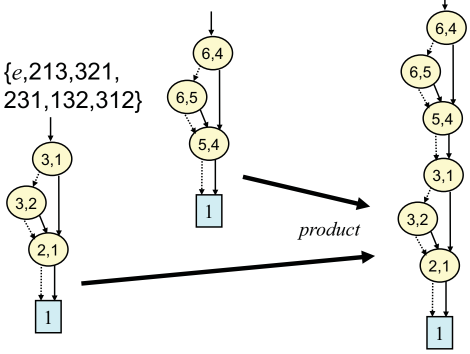
$$P * Q = P * (Q_0 \cup Q_1\tau_{xy}) = (P * Q_0) \cup (P * Q_1)\,\tau_{xy}$$

Recursive calls with cache.

Product operation for disjoint permutations

{e,123546,123654,
123564,123465,123645}

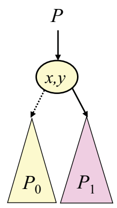
{e,213,321,
231,132,312}



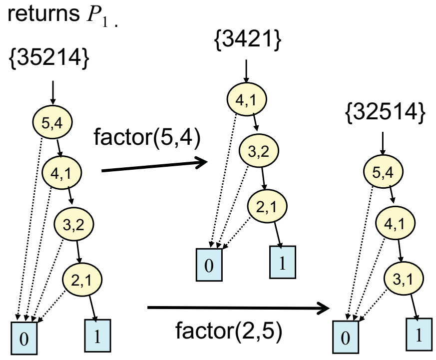
{e,123546,123654,
123564,123465,123645,
213456,213546,213654,
213564,213465,213645,
321456,321546,321654,
321564,321465,321645,
231456,231546,231654,
231564,231465,231645,
132456,132546,132654,
132564,132465,132645,
312456,312546312654,
312564,312465,312645}

Factor operation for πDDs

- P.factor(u, v) returns $\{\pi \in (P\,\tau_{uv}) \mid u\pi = u\}$.
  - If (u,v) corresponds to the root node-ID (x,y), P.factor(x, y) returns P_1.



{321,231,132,213}.factor(3,1)
= {321, 231}τ_{31}
= {123, 213}
= {e, 21}

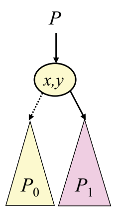


Procedure of Factor operation

$$P.\text{factor}(u, v) = (P_0 \cup P_1\,\tau_{xy}).\text{factor}(u, v)$$

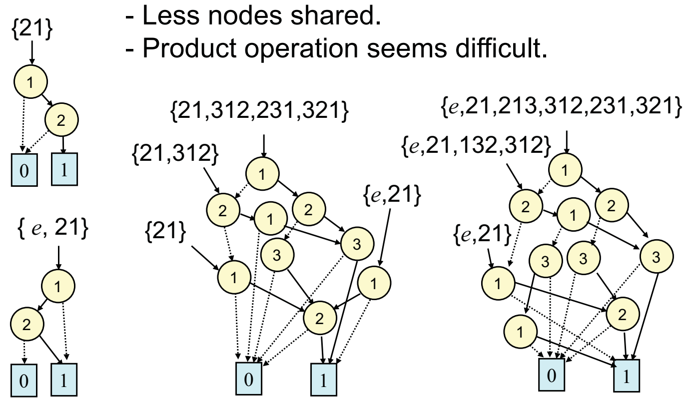
if$(x<u$ or $x<v)$ P.factor$(u, v) = \begin{cases} P & (\text{if } u=v) \\ \varphi & (\text{otherwise}) \end{cases}$

if$(x=u>v>y)$ P.factor$(u, v) = P_0$.factor(u, v)
if$(x=u>y=v)$ P.factor$(u, v) = P_1$
if$(x=u>y>v)$ P.factor$(u, v) = \varphi$
if$(x=v>y=u)$ P.factor$(u, v) = P_1$.factor(u, u)
if$(x>u, y=v)$ P.factor$(u, v) = P_0$.factor(u, v)
if$(x>u, y\neq v)$ P.factor$(u, v) = P_0$.factor$(u, v) \cup (P_1\tau_{xy})$.factor$(u, v)$
$= P_0$.factor$(u, v) \cup ((P_1\tau_{xy})\,\tau_{uv})$.factor$(u, u)$
$= P_0$.factor$(u, v) \cup (P_1(\tau_{uv}\tau_{xy'}))$.factor$(u, u)$
$= P_0$.factor$(u, v) \cup (P_1\tau_{uv})$.factor$(u, u)\,\tau_{xy'}$
$= P_0$.factor$(u, v) \cup P_1$.factor$(u, v)\,\tau_{xy'}$
if$(x=v>u, y\neq u)$ P.factor$(u, v) = P_0$.factor$(u, v) \cup P_1$.factor$(u, y)\,\tau_{xy}$

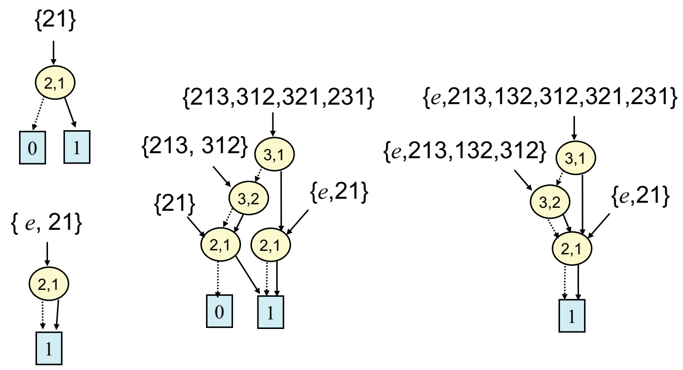


If we use SeqBDDs for permutations?

- Less nodes shared.
- Product operation seems difficult.



πDDs for sets of permutations



Upper bound of πDD sizes

- Number of Families of permutations up to n items: $2^{n!}$
  - $n!$ 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, πDD size can be $O(n!)$ bit.

πDD sizes for typical cases

- φ, { e } : $O(1)$ nodes
- Sets of a single permutation with n items: $O(n)$ nodes
- Sets of any k permutations with n items: $O(k\ n)$ 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.
- πDD can be compact for representing the family consists of many similar sub-permutations.

TODO

- Implementation of the algorithms.
- Determine complexity of operations.
- Applying to interesting problems.
  - Performance evaluation.
- Variable ordering problem.
- Relationship to permutation group theory.
  - If $P * P = P$, then P forms a permutation group.
- Variations.
  - Histogram (multiset) of permutations.
  - Permutations of k out of n items. (allows lack of items)
  - Permutations of multiset items. (allows duplication of items)