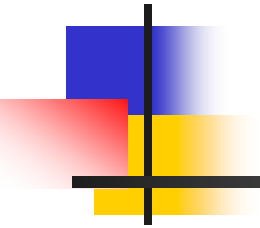
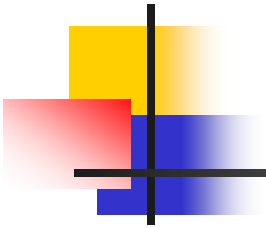


ZDD and its applications to intelligent processing

Shin-ichi Minato

Graduate School of Information Science and Technology

Hokkaido University, Japan.

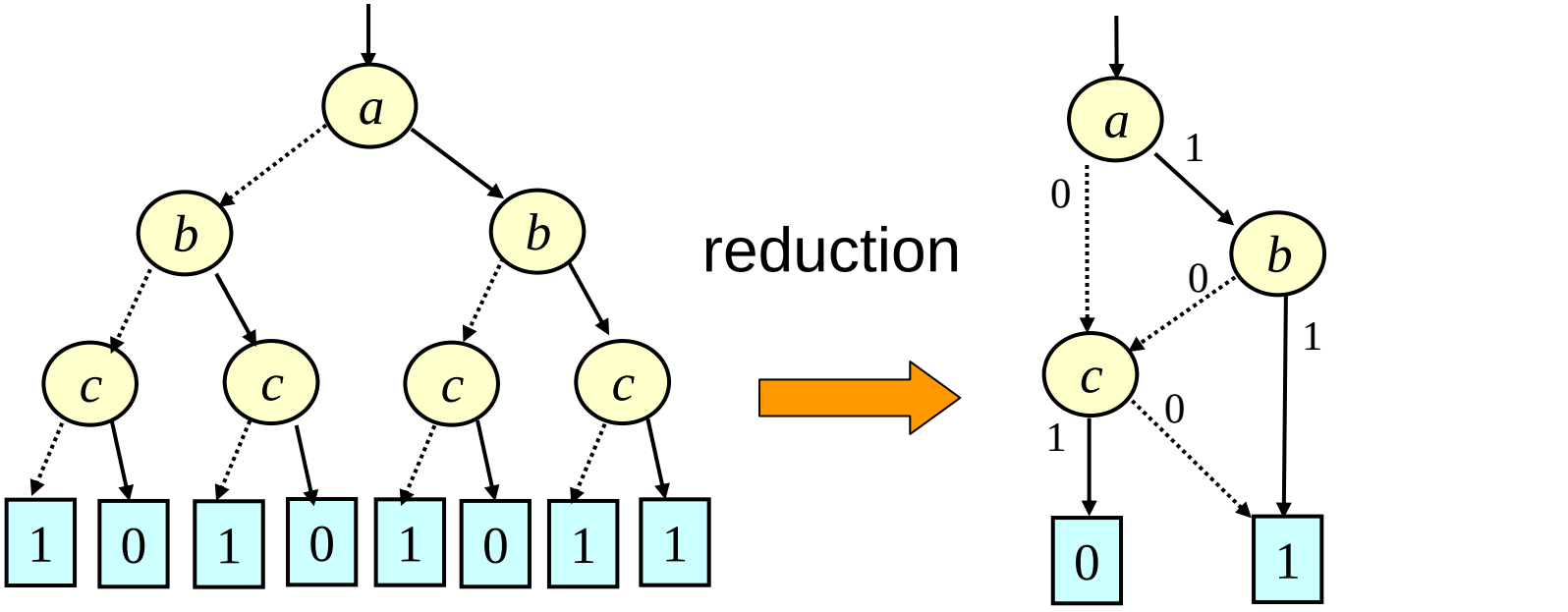


Background

- BDD-based algorithms have been developed mainly in VLSI logic design area. (since early 1990's.)
  - Equivalence checking for combinational circuits.
  - Symbolic model checking for logic / behavioral designs.
  - Logic synthesis / optimization.
  - Test pattern generation.
- Recently, BDDs are applied for not only VLSI design but also for more general purposes.
  - **Data mining (Fast frequent itemset mining) [Minato2005,2008,2010]**
  - Computation of Bayesian networks for probabilistic system analysis.[Minato2007]

BDD (Binary Decision Diagram) [Bryant86]

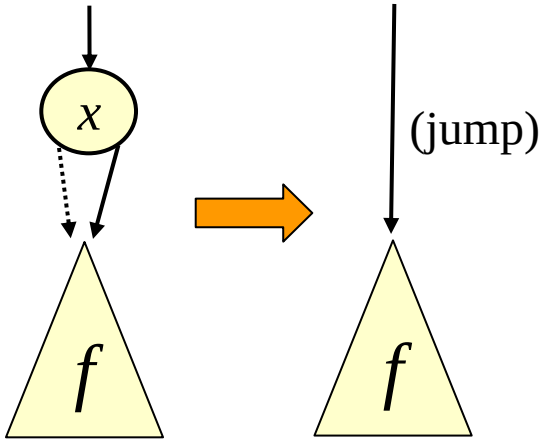
- Graph representation of Boolean function data.
  - Canonical form obtained by applying reduction rules to a binary tree with a fixed variable ordering.

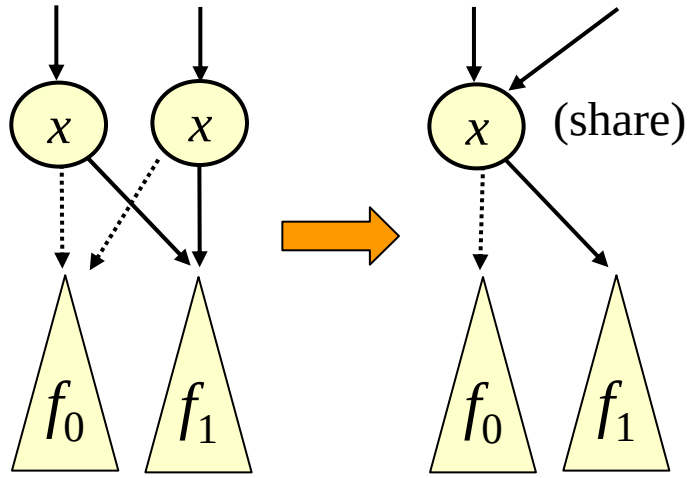


Binary decision tree equivalent to truth table

Reduced Ordered BDD

BDD reduction rules





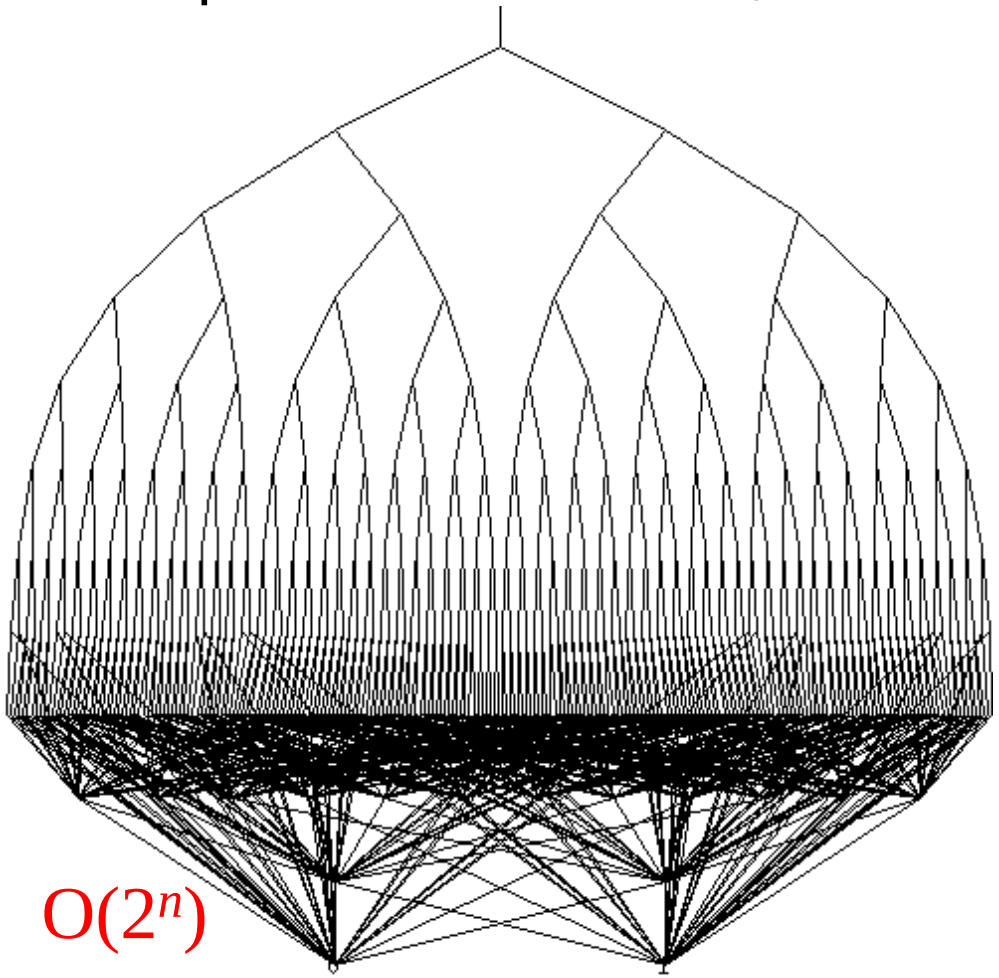
Eliminate all redundant nodes.

Share all equivalent nodes.

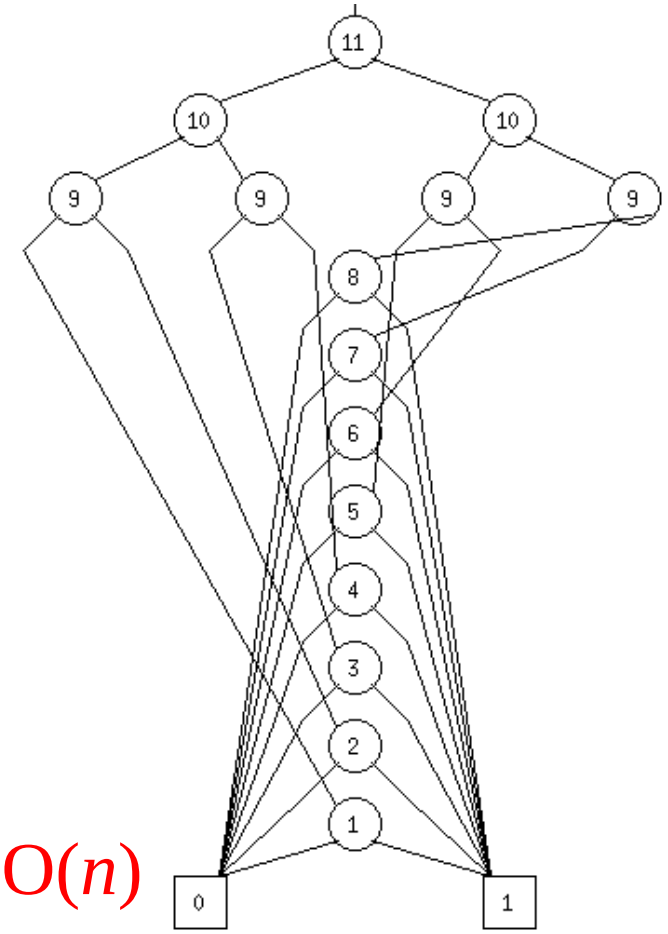
Gives a unique and compressed representation for a given Boolean function under a fixed variable ordering.

Effect of BDD reduction rules

- Exponential advantage can be seen in extreme cases.
  - Depends on instances, but effective for many practical ones.



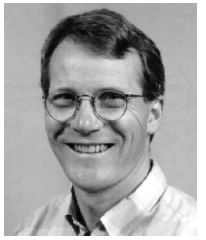
$O(2^n)$



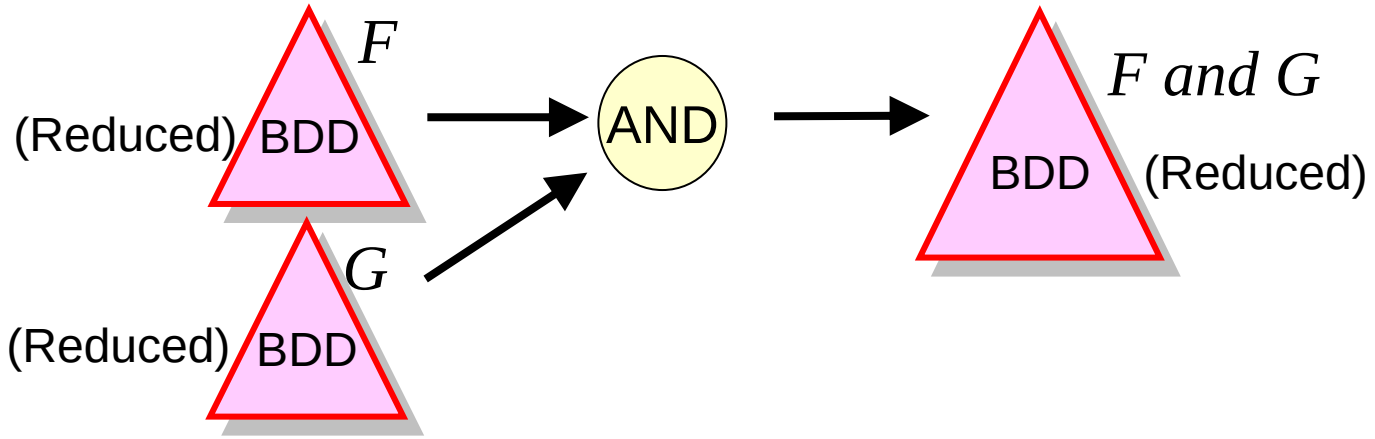
$O(n)$

BDD-based logic operation algorithm

- If we generate BDDs from the binary tree: always requires exponential time & space. (→ impracticable for large number of variables)
- **Innovative BDD synthesis algorithm**
  - **Proposed by R. Bryant in 1986.**
  - Best cited paper for many years in EE&CS areas.



R. Bryant (CMU)



A BDD can be constructed from the two operands of BDDs. (Computation time is linear to BDD size.)

Boolean function and combinatorial itemset

a	b	c	F	
0	0	0	0	
1	0	0	0	
0	1	0	0	
1	1	0	1	→ ab
0	0	1	1	→ c
1	0	1	1	→ ac
0	1	1	0	
1	1	1	0	

Boolean function:

$F = (a\ b\ {\sim}c)\ \mathrm{V}\ ({\sim}b\ c)$

Combinatorial itemset:

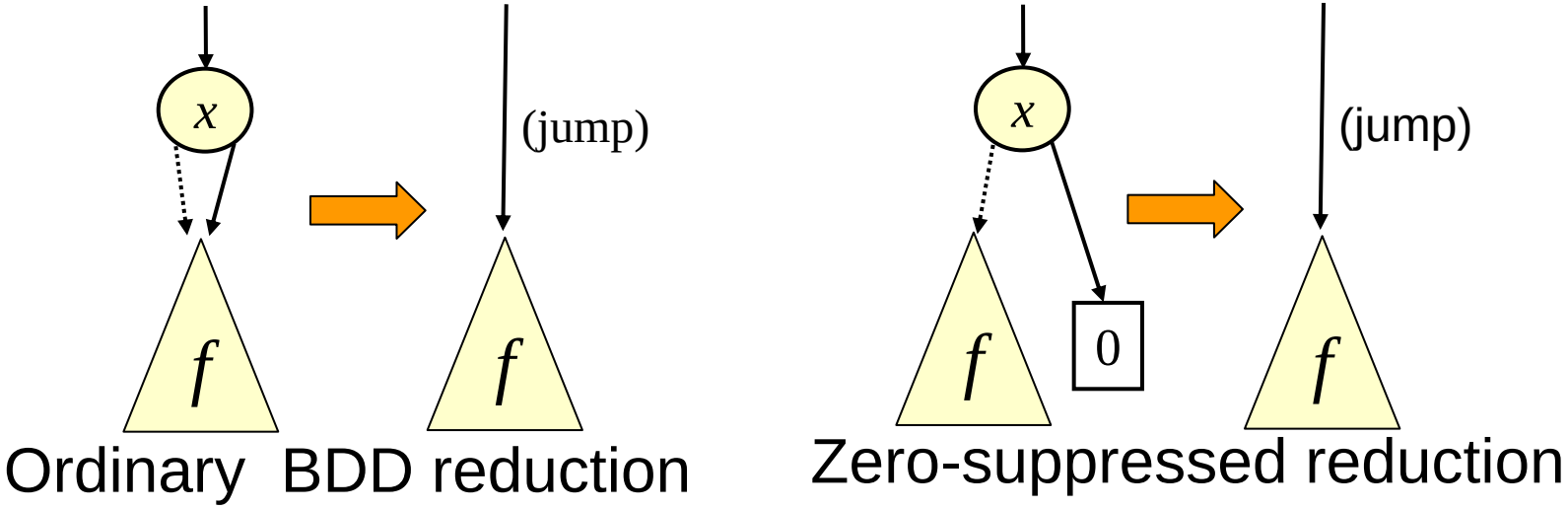
$F = \{ab, ac, c\}$

(customer's choice)

- Operations of combinatorial itemsets can be done by BDD-based logic operations.
  - Union of sets → logical OR
  - Intersection of sets → logical AND
  - Complement set → logical NOT

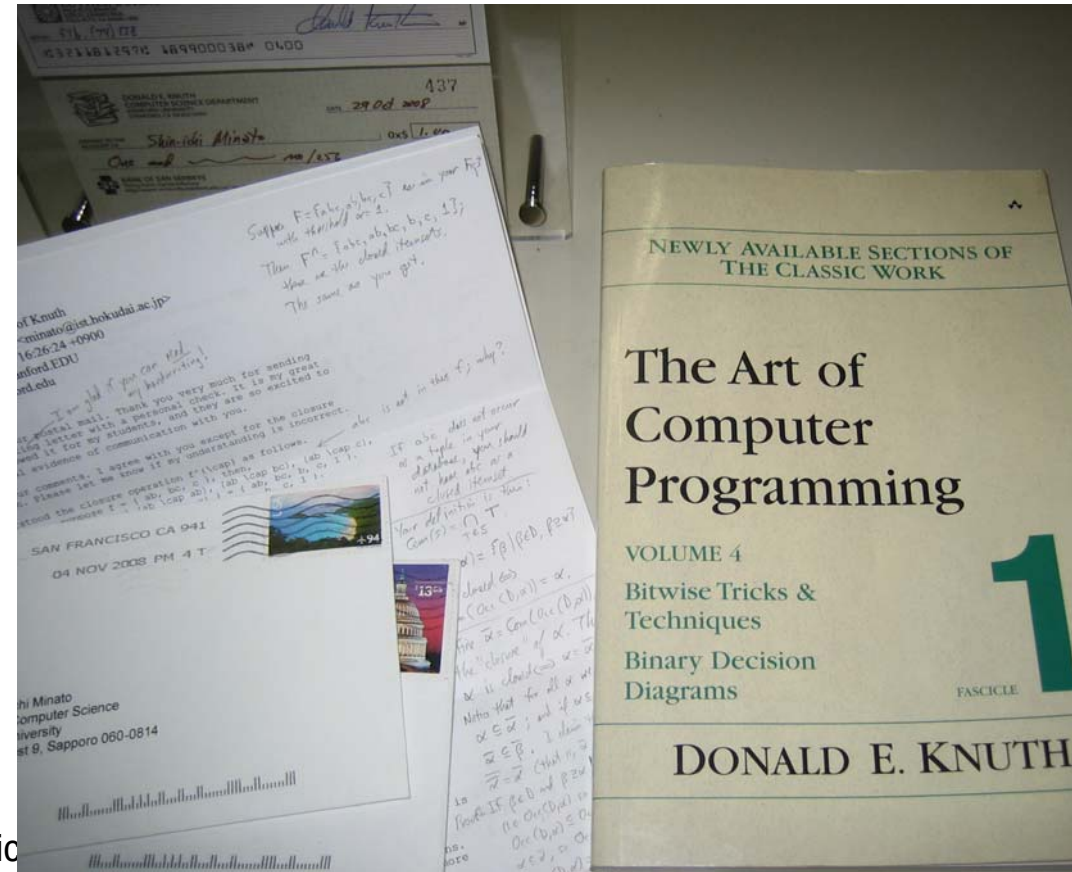
Zero-suppressed BDD (ZDD) [Minato93]

- A variant of BDDs for **combinatorial itemets.**
- Uses a new reduction rule different from ordinary BDDs.
  - Eliminate all nodes whose “1-edge” directly points to 0-terminal.
  - Share equivalent nodes as well as ordinary BDDs.
- If an item x does not appear in any itemset, the ZDD node of x is automatically eliminated.
  - When average appearance ratio of each item is 1%, ZDDs are more compact than ordinary BDDs, up to 100 times.



BDDs/ZDDs in the Knuth's book

- The latest Knuth's book fascicle (Vol. 4-1) includes a **BDD section with 140 pages and 236 exercises**.
- In this section, Knuth used **30 pages for ZDDs, including more than 70 exercises.**
  - I honored to serve proofreading of the draft version of his article.
  - Knuth recommended to use "ZDD" instead of "ZBDD."
  - He named ZDD operation set as "Family Algebra."
  - Knuth has developed his own BDD/ZDD package.
  - His recent lecture at Oxford was titled "Fun with ZDDs.



Algebraic operations for ZDDs

- Knuth evaluated not only the data structure of ZDDs, but more interested in **the new algebra on ZDDs**.

ϕ, $\{1\}$	***Empty*** and ***singleton set***. (0/1-terminal)	Basic operations (Corresponds to Boolean algebra)
$P.top$	Returns the ***item-ID*** at the top node of P.	
$P.onset(v)$ $P.offset(v)$	Selects the subset of itemsets including or excluding v.	
$P.change(v)$	Switching v (***add / delete***) on each itemset.	
$\cup$, $\cap$, $\setminus$	Returns ***union, intersection,*** and ***difference set***.	
$P.count$	***Counts number*** of combinations in P.	
$P * Q$	***Cartesian product set*** of P and Q.	**New operations introduced by Minato.**
P / Q	***Quotient set*** of P divided by Q.	
$P \% Q$	***Reminder set*** of P divided by Q.	

Formerly I called this "unate cube set algebra," but Knuth reorganized as **"Family algebra."**

Useful for many practical applications.

Frequent itemset mining

- Basic and well-known problem in database analysis.

Record ID	Tuple
1	*a b c*
2	*a b*
3	*a b c*
4	*b c*
5	*a b*
6	*a b c*
7	*c*
8	*a b c*
9	*a b c*
10	*a b*
11	*b c*

Frequency threshold = 10 → $\{ b \}$

Frequency threshold = 8 → $\{ ab, a, b, c \}$

Frequency threshold = 7 → $\{ ab, bc, a, b, c \}$

Frequency threshold = 5 → $\{abc, ab, bc, ac, a, b, c \}$

Frequency threshold = 1 → $\{abc, ab, bc, ac, a, b, c \}$

Existing itemset mining algorithms

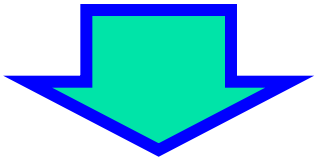
- Frequent itemset mining is one of the fundamental data mining problems.
  - ***Apriori*** [Agrawal1993]
    First efficient method of enumerating all frequent patterns.
    Breadth-first search with dynamic programming.
  - ***Eclat*** [Zaki1997]
    Depth-first search algorithm. Less memory consuming.
    In some cases, faster than Apriori.
  - ***FP-growth*** [Han2000]
    Depth-first search using “FP-tree,” graph-based data structure. (→ ***ZDD-growth*** [Minato2006])
- ***LCM*** *(Linear time Closed itemset Miner)* [Uno2003]
  - with a theoretical bound as output linear time.
  - known as one of the fastest implementation.

Problem in LCM (and the most of others)

- LCM (and most of the other itemset mining algorithms) focuses on just **enumerating** the frequent itemsets.
- It is a different matter how to **store and index** the result of huge number of itemsets.
  - If we want to post-process the mining results, once we have to dump the frequent itemsets into storage.
  - Even LCM is an output linear time algorithm, it may require impracticable time and space.
    (→ number of solution may be exponential.)
  - Usually we control the output size with the minimum support threshold in ad hoc setting, but we do not know if it may lose some important information.

“LCM over ZDDs” [Minato et al. 2008]

- ***LCM:*** *[Uno2003]*
  Output-linear time algorithm of frequent itemset mining.
- ***ZDD:*** *[Minato93]*
  A compact graph-based representation for large-scale sets of combinations.



Combination of the two techniques

Generates large-scale frequent itemsets on the main memory, with a very small overhead from the original LCM.

(→ Sub-linear time and space to the number of solutions when ZDD compression works well.)

LCM over ZDDs: An example

- The results of frequent itemsets are obtained as ZDDs on the main memory. (not generating a file.)

Record ID	Tuple
1	$a\ b\ c$
2	$a\ b$
3	$a\ b\ c$
4	$b\ c$
5	$a\ b$
6	$a\ b\ c$
7	c
8	$a\ b\ c$
9	$a\ b\ c$
10	$a\ b$
11	$b\ c$

LCM over ZDDs

Freq. thres. $\alpha = 7$

$\{ ab, bc, a, b, c \}$

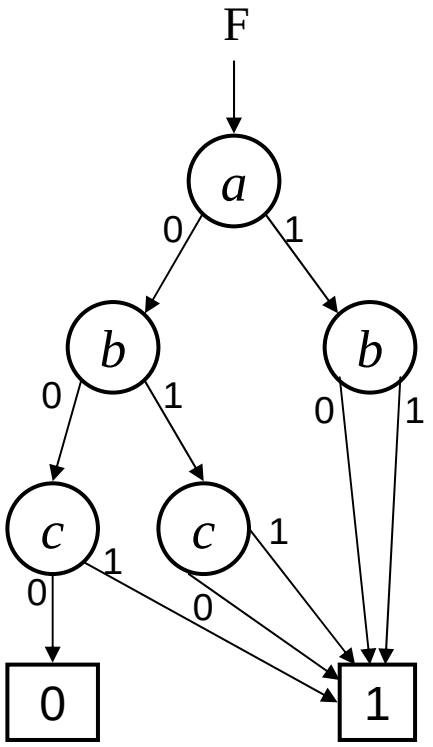


Table 2. Co… CM ov… prev…

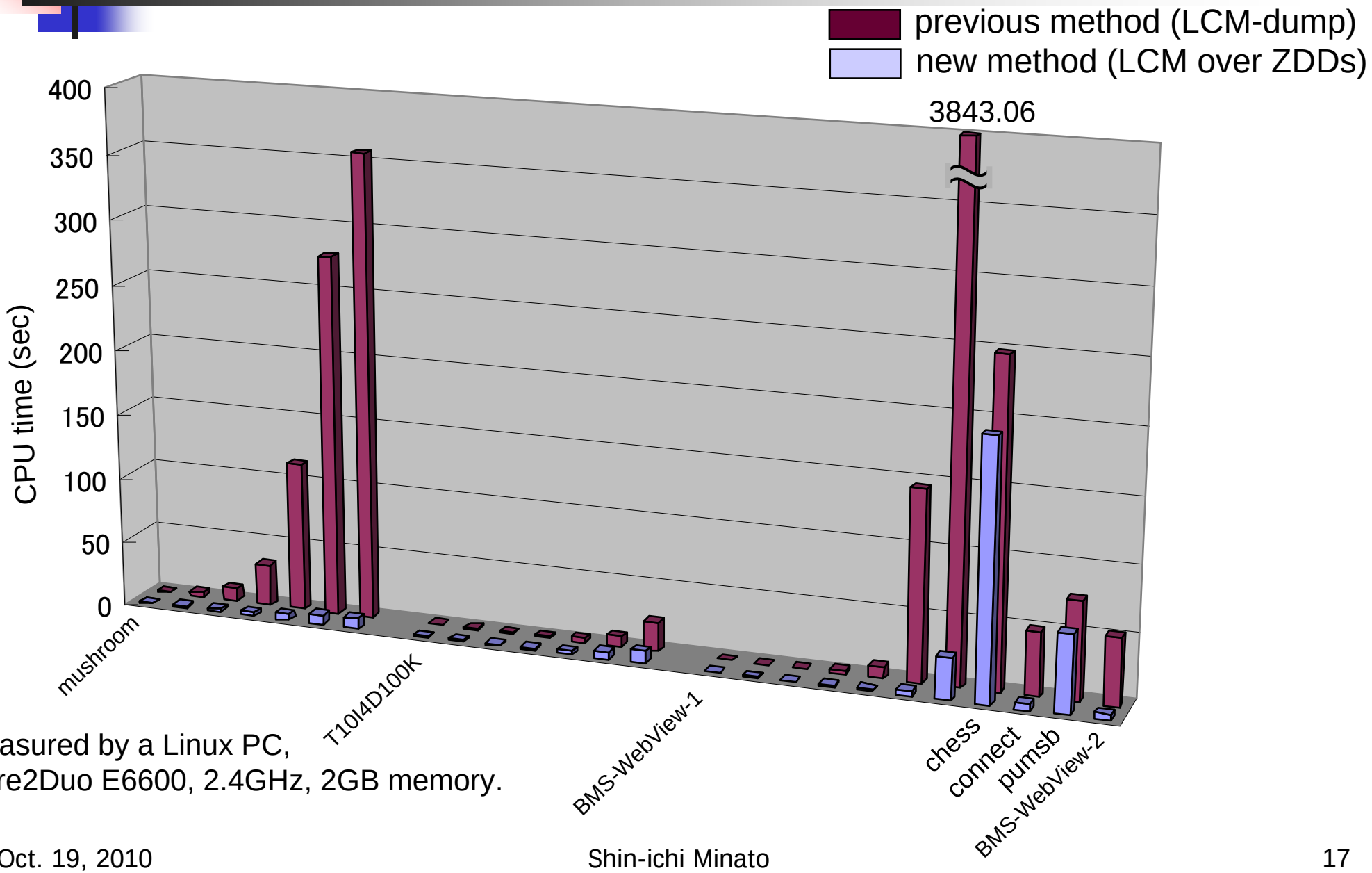
solutions

LCM over ZDDs

Original LCM

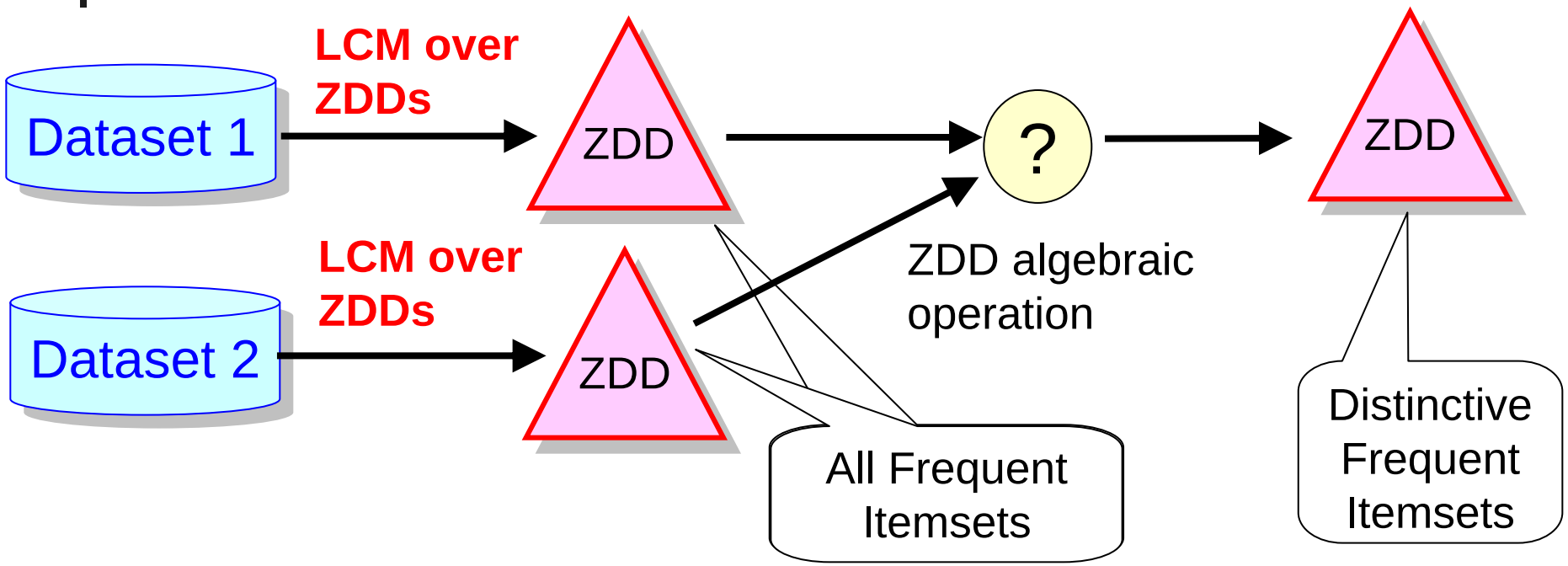
Dataset name: min. support	# itemsets	CMov… |ZBDD|	CMov… Time(s)	…t Time(s)	LC… Time(s)	…-growth Time(s)
mushroom: 1,000	123,287	760	0.50	0.49	0.64	1.78
500	1,442,504	2,254	1.32	1.30	3.29	3.49
300	5,259,786	4,412	2.25	2.22	9.96	5.11
200	18,094,822	6,383	3.21	3.13	31.63	6.24
100	66,076,586	11,584	5.06	4.87	114.21	6.72
70	153,336,056	14,307	7.16	7.08	277.15	6.97
50	198,169,866	17,830	8.17	7.86	357.27	6.39
T10I4D100K: 100	27,533	8,482	0.85	0.85	0.86	209.82
50	53,386	16,872	0.97	0.92	0.98	242.31
20	129,876	58,413	1.13	1.08	1.20	290.78
10	411,366	173,422	1.55	1.36	1.64	332.22
5	1,923,260	628,491	2.86	2.08	3.54	370.54
3	6,169,854	1,576,184	5.20	3.15	8.14	386.72
2	19,561,715	3,270,977	9.68	5.09	22.66	384.60
BMS-WebView-1: 50	8,192	3,415	0.11	0.11	0.12	29.46
40	48,544	10,755	0.18	0.18	0.22	48.54
36	461,522	28,964	0.49	0.42	0.98	67.16
35	1,177,608	38,164	0.80	0.69	2.24	73.64
34	4,849,466	49,377	1.30	1.07	8.58	83.36
33	69,417,074	59,119	3.53	3.13	144.98	91.62
32	1,531,980,298	71,574	31.90	29.73	3,843.06	92.47
chess: 1,000	29,442,849	53,338	197.58	197.10	248.18	1,500.78
connect: 40,000	23,981,184	3,067	5.42	5.40	49.21	212.84
pumsb: 32,000	7,733,322	5,443	60.65	60.42	75.29	4,189.09
BMS-WebView-2: 5	26,946,004	353,091	4.84	3.62	51.28	118.01

Performance of LCM over ZDDs



measured by a Linux PC,
Core2Duo E6600, 2.4GHz, 2GB memory.

Post Processing after LCM over ZDDs



- We can extract distinctive itemsets by comparing frequent itemsets for multiple sets of databases.
  - Various ZDD algebraic operations can be used for the comparison of the huge number of frequent itemsets.

Conclusion

- We presented our recent results on ZDD-based techniques for data mining and knowledge discovery.
  - Automatic compressed data for a huge size of itemsets.
  - Can be processed efficiently by using various set operations without decompression.
  - Limitation: no results obtained when memory overflow occurs.
- In 1990's, BDDs were only applied for VLSI design area.
  - On that time, the main memory capacity was not sufficient for database applications.
  - Recently, BDD/ZDD-based techniques becomes practicable for many database application.
- We started a new nation-wide project "ERATO": **"Discrete Structure Manipulation System"** promoted by JST, scientific agency of Japan.