The Art of Counting Graphs

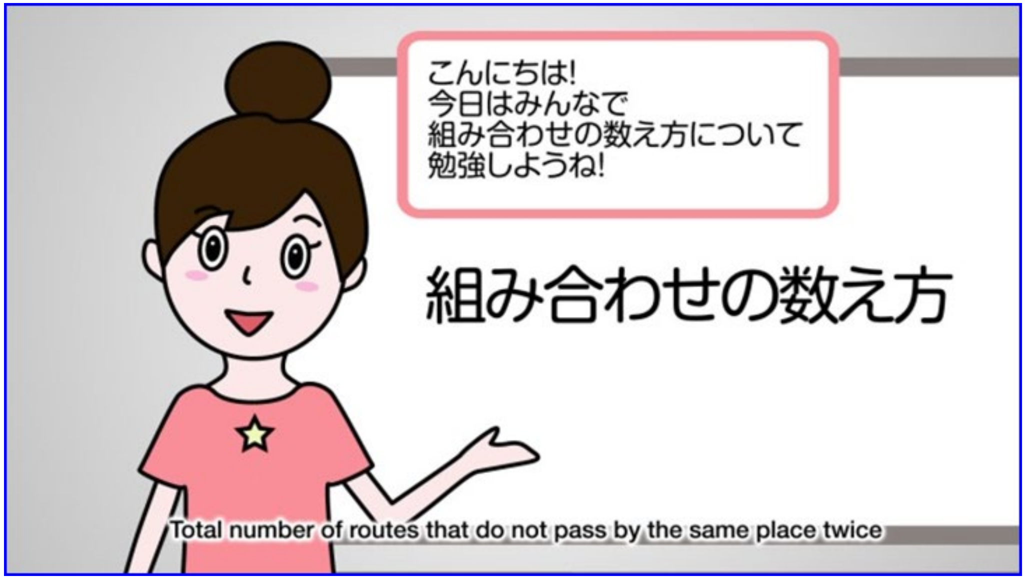
Shin-ichi Minato
Kyoto University

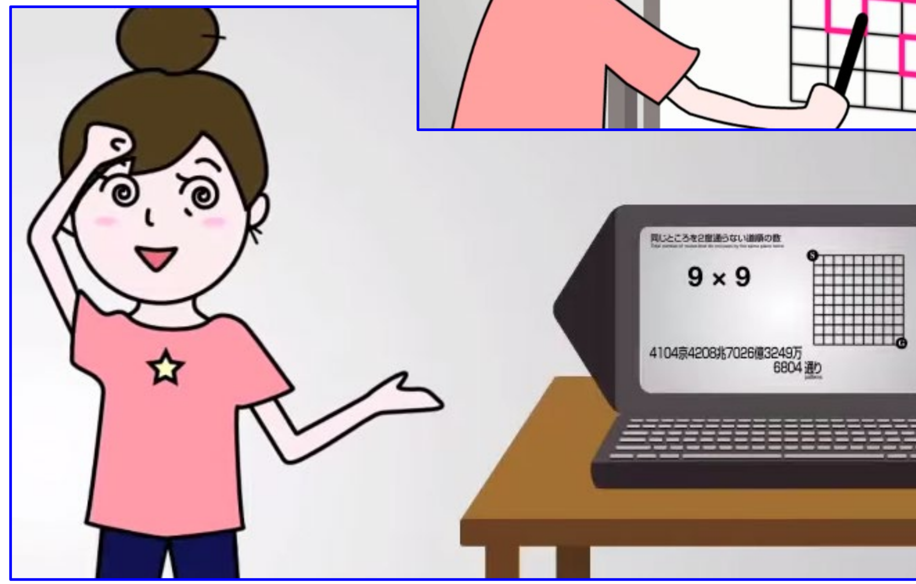
Introduction: Shin-ichi Minato



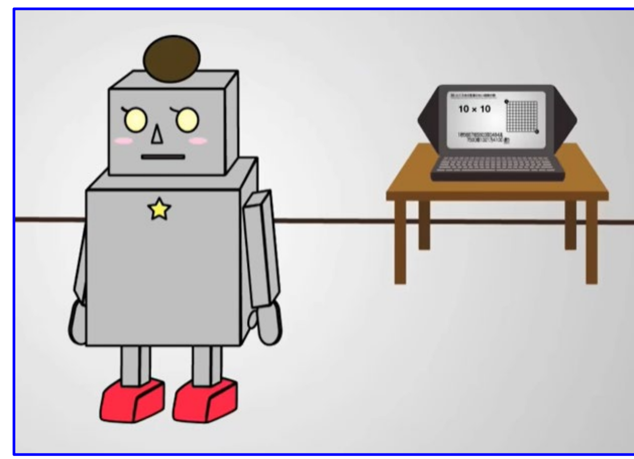
- **Prof. of Kyoto University (from 2018)**
  - Worked for Hokkaido University, Sapporo,
  - Worked for NTT Labs. from 1990 to 2004.
- **Main research area:**
  - 1990's: VLSI CAD (logic design and verification)
    - Proposed **"Zero-suppressed BDD" (ZDD)** at DAC 1993
  - 2000's: Large-scale discrete structure manipulation for data mining, graph algorithms, knowledge compilation, etc.
    - Proposed fast data mining **"LCM over ZDDs"** at PAKDD 2008
    - ZDD-based methods for various graph problems in Knuth-Book
    - Proposed **"Permutation DD" (πDD)** at SAT-2011
    - Proposed fast statistical testing **"LAMP 2.0"** at ECML/PKDD 2014
  - 2020～now: Research director of **"AFSA (Algorithmic Foundation for Social Advancement) project"**
    - Five years, 40 PI researchers, nation-wide research project
    - My current interest: Integration of **"enumeration, optimization, and satisfiability"** techniques

Animation Movie on graph counting [2012]



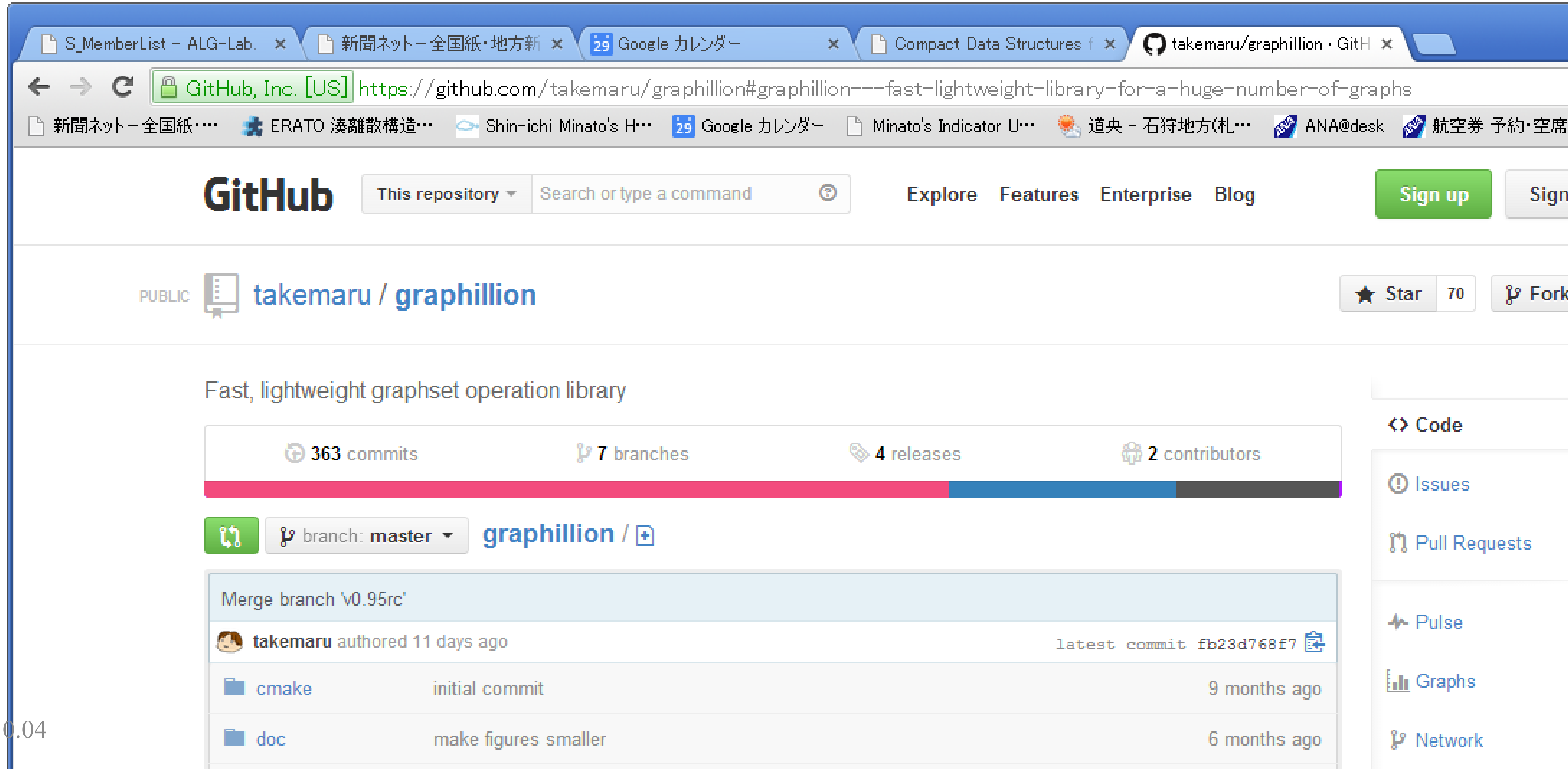


- Shows strong power of combinatorial explosion, and importance of algorithmic techniques.
- **3 million views** in 10 years on YouTube.



Open software: "Graphillion.org"

- **Toolbox for ZDD-based graph enumeration.**
  - **Easy interface using Python graph library.**



International Competition on Graph Counting Algorithm (ICGCA)

AFSA

https://afsa.jp/icgca

AFSA

Home | Registration | Submission | Symposium

International Competition on Graph Counting Algorithms

- **Home**
- **What's new**
- **Overview**
- **Problem and benchmarks**
  - Input
  - Output
  - Number of benchmarks
  - Example benchmarks
  - Example codes
- **Rules**
  - Contestants
  - Evaluation metrics
  - Input format

What's new

August 29, 2023

- The program for the ICGCA symposium, where the results will be unveiled, is available here.

July 18, 2023

- The GNU multi-precision library (libgmp-dev) and Xlib (libx11-dev) will be available on the evaluation environment upon requests from some contestants.

July 13, 2023

- The wrong description for the evaluation script has been fixed; the directory path was written as **/home/{user}/submission/**, but the correct one is **/home/icgca/{user}/submission/**.

July 12, 2023

- The evaluation script had a bug on multi-threading and has been fixed, so please re-download it from here.

July 07, 2023

International Competition on Graph Counting Algorithm (ICGCA)

AFSA

AFSA

Home Registration Submission Symposium

Sep. 7, 2023 @ Osaka, Japan

Input format

July 07, 2023

The Art of Counting Graphs

Shin-ichi Minato
Kyoto University

The Art of Counting Graphs:

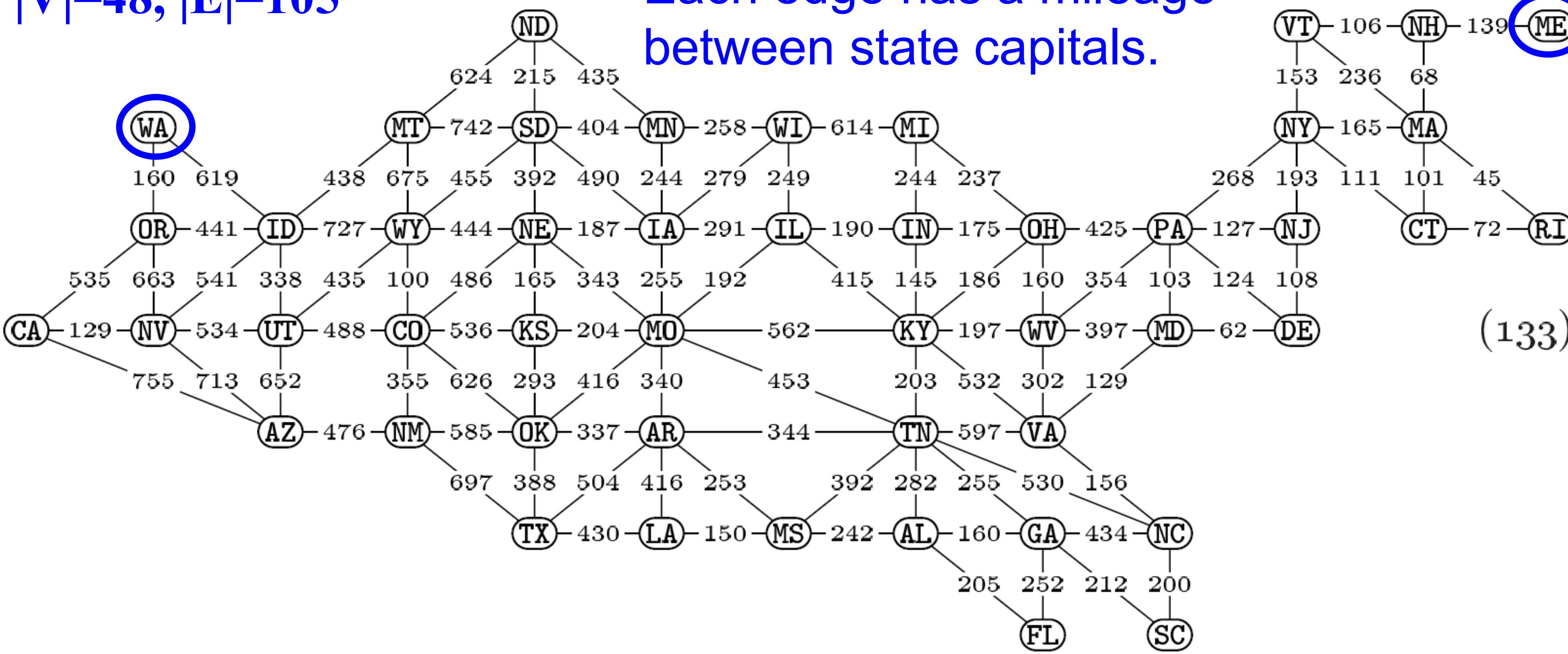
“Combinatorial Enumeration and Ranking”

Shin-ichi Minato
Kyoto University

Motivating Problem (Exercise in Knuth-Book)

|V|=48, |E|=105

Each edge has a mileage between state capitals.



- **Let us enumerate all Hamiltonian paths from WA to ME.**
- **Efficient DP algorithm (Frontier-based method) is shown.**
  - **Generated ZDD size: 3,616 nodes**
  - **All Hamiltonian paths: 6,876,928 ways**
  - **Computation time: 0.03 sec.**

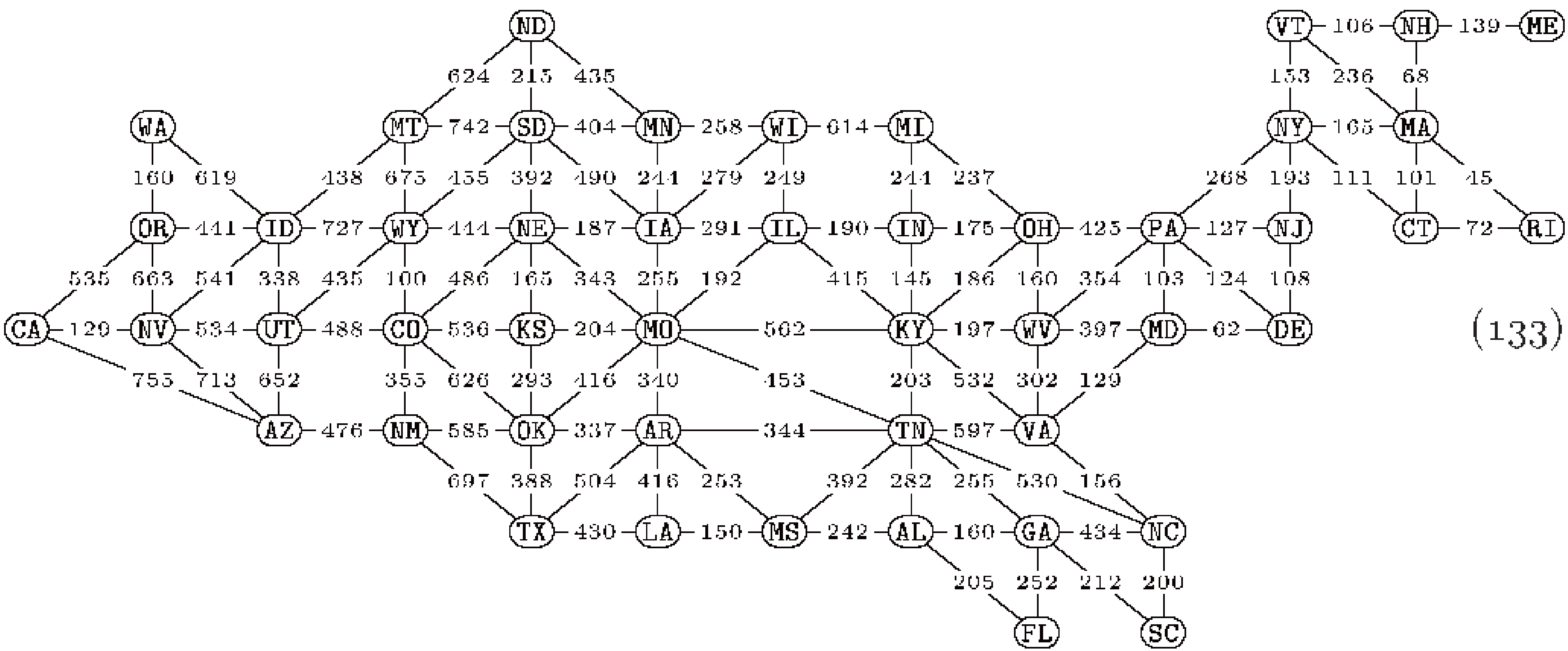
After ZDD construction, is it easy or not easy?

AFSA

- **Easy tasks by using ZDDs:** **(Linear-time for ZDD size)**
  - Counting **number of solutions**. (6,876,928 ways)
  - Finding **shortest/longest paths**. (11,698 / 18,040 miles)
  - Computing the **average length** of all feasible solutions.
- **Seems easy but still not easy tasks:**
  - Counting all paths **less than the average length**.
  - Finding the **median of** all feasible solutions.
  - Show **ranking** of a given solution.
  - Constructing ZDDs for **all paths no more than 10% increase** from the shortest path.
  - Constructing ZDDs enumerating the **top 5% solutions**.

More difficult variation of the problem

- **Let us enumerate all "self-avoiding tours" visiting 24 (a half number) of the 48 States.**
  - **ZDD size: 26,798 nodes, Computation time: 0.09 sec.**
  - **Total solutions: 398,924,116 ways.**
- **Let us cover the total population as many as possible.**



Population data of 48 States [2020 US Census]

AFSA

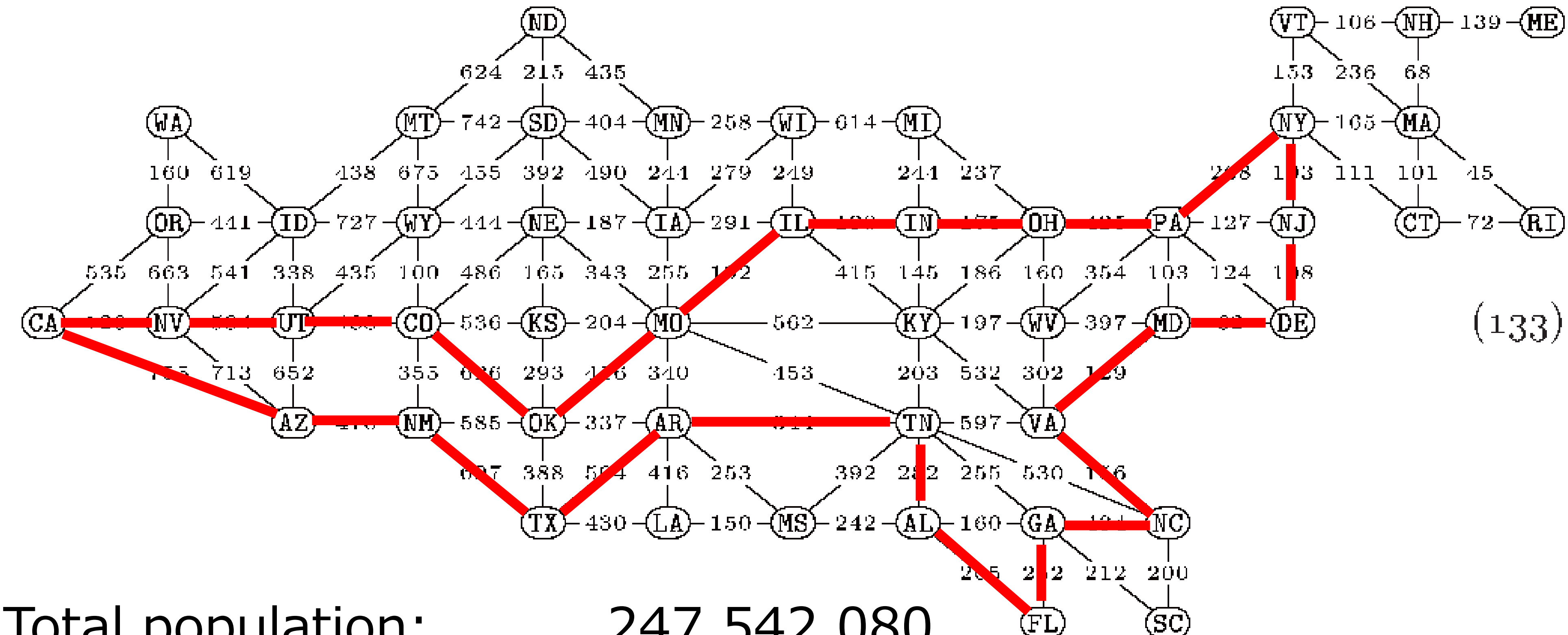
State	Code	Population
Alabama	AL	5,024,279
Arizona	AZ	7,151,502
Arkansas	AR	3,011,524
California	CA	39,538,223
Colorado	CO	5,773,714
Connecticut	CT	3,605,944
Delaware	DE	989,948
Florida	FL	21,538,187
Georgia	GA	10,711,908
Idaho	ID	1,839,106
Illinois	IL	12,812,508
Indiana	IN	6,785,528
Iowa	IA	3,190,369
Kansas	KS	2,937,880
Kentucky	KY	4,505,836
Louisiana	LA	4,657,757
Maine	ME	1,362,359
Maryland	MD	6,177,224
Massachusetts	MA	7,029,917
Michigan	MI	10,077,331
Minnesota	MN	5,706,494
Mississippi	MS	2,961,279
Missouri	MO	6,154,913
Montana	MT	1,084,225
Nebraska	NE	1,961,504
Nevada	NV	3,104,614
New Hampshire	NH	1,377,529
New Jersey	NJ	9,288,994
New Mexico	NM	2,117,522
New York	NY	20,201,249
North Carolina	NC	10,439,388
North Dakota	ND	779,094
Ohio	OH	11,799,448
Oklahoma	OK	3,959,353
Oregon	OR	4,237,256
Pennsylvania	PA	13,002,700
Rhode Island	RI	1,097,379
South Carolina	SC	5,118,425
South Dakota	SD	886,667
Tennessee	TN	6,910,840
Texas	TX	29,145,505
Utah	UT	3,271,616
Vermont	VT	643,077
Virginia	VA	8,631,393
Washington	WA	7,705,281
West Virginia	WV	1,793,716
Wisconsin	WI	5,893,718
Wyoming	WY	576,851
Total		328,571,074

Population data of 48 States [2020 US Census]

AFSA

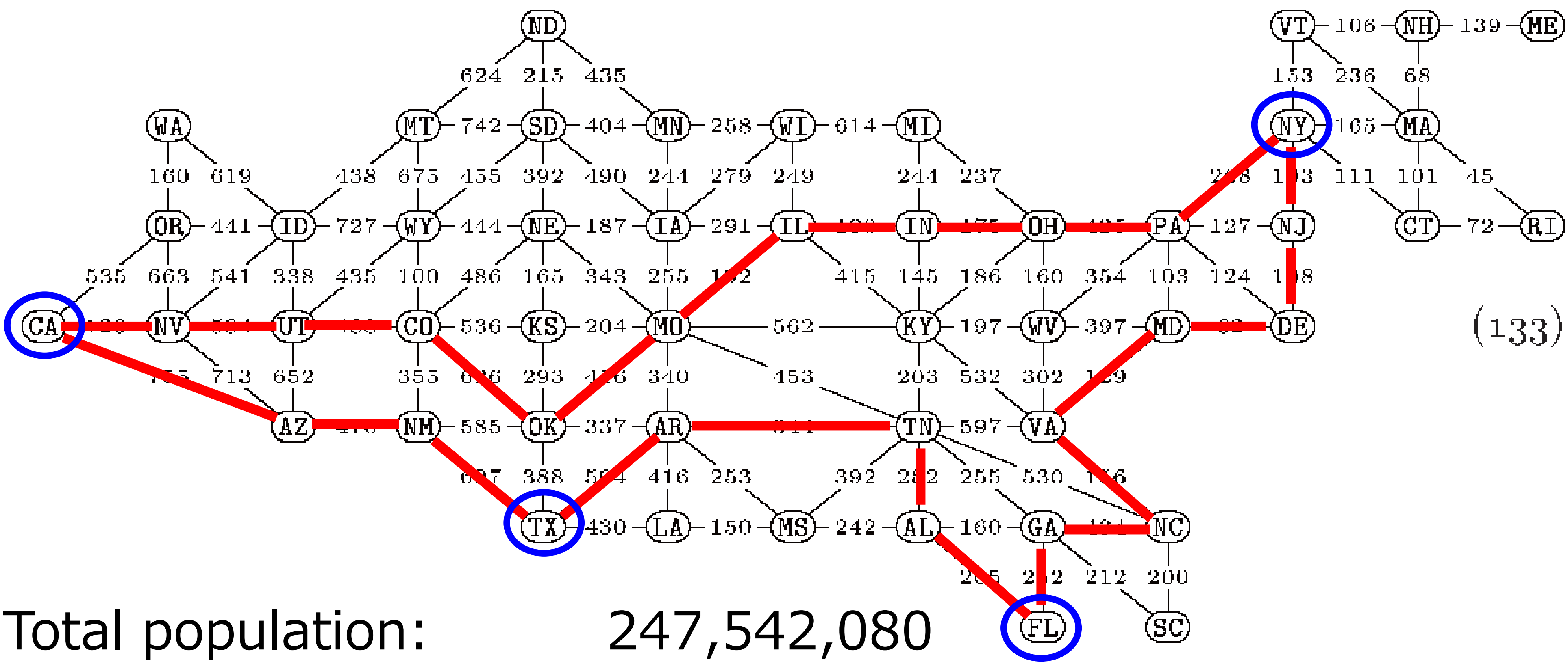
State	Code	Population
Alabama	AL	5,024,279
Arizona	AZ	7,151,502
Arkansas	AR	3,011,524
California	CA	39,538,223
Colorado	CO	5,773,714
Connecticut	CT	3,605,944
Delaware	DE	989,948
Florida	FL	21,538,187
Georgia	GA	10,711,908
Idaho	ID	1,839,106
Illinois	IL	12,812,508
Indiana	IN	6,785,528
Iowa	IA	3,190,369
Kansas	KS	2,937,880
Kentucky	KY	4,505,836
Louisiana	LA	4,657,757
Maine	ME	1,362,359
Maryland	MD	6,177,224
Massachusetts	MA	7,029,917
Michigan	MI	10,077,331
Minnesota	MN	5,706,494
Mississippi	MS	2,961,279
Missouri	MO	6,154,913
Montana	MT	1,084,225
Nebraska	NE	1,961,504
Nevada	NV	3,104,614
New Hampshire	NH	1,377,529
New Jersey	NJ	9,288,994
New Mexico	NM	2,117,522
New York	NY	20,201,249
North Carolina	NC	10,439,388
North Dakota	ND	779,094
Ohio	OH	11,799,448
Oklahoma	OK	3,959,353
Oregon	OR	4,237,256
Pennsylvania	PA	13,002,700
Rhode Island	RI	1,097,379
South Carolina	SC	5,118,425
South Dakota	SD	886,667
Tennessee	TN	6,910,840
Texas	TX	29,145,505
Utah	UT	3,271,616
Vermont	VT	643,077
Virginia	VA	8,631,393
Washington	WA	7,705,281
West Virginia	WV	1,793,716
Wisconsin	WI	5,893,718
Wyoming	WY	576,851
Total		328,571,074

The most populated 24 states self-avoiding tour



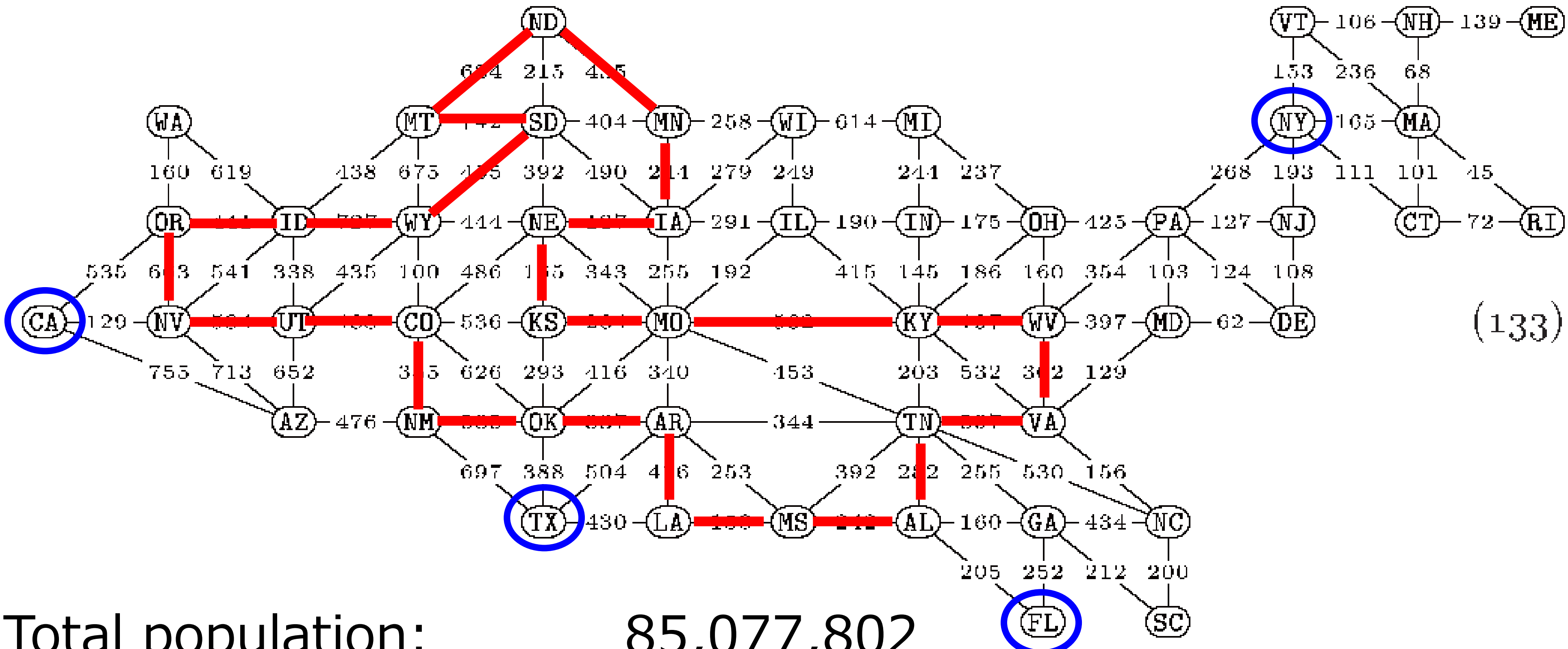
Total population: 247,542,080
(Exact half population: 164,285,537)

The most populated 24 states self-avoiding tour



Total population: 247,542,080
(Exact half population: 164,285,537)

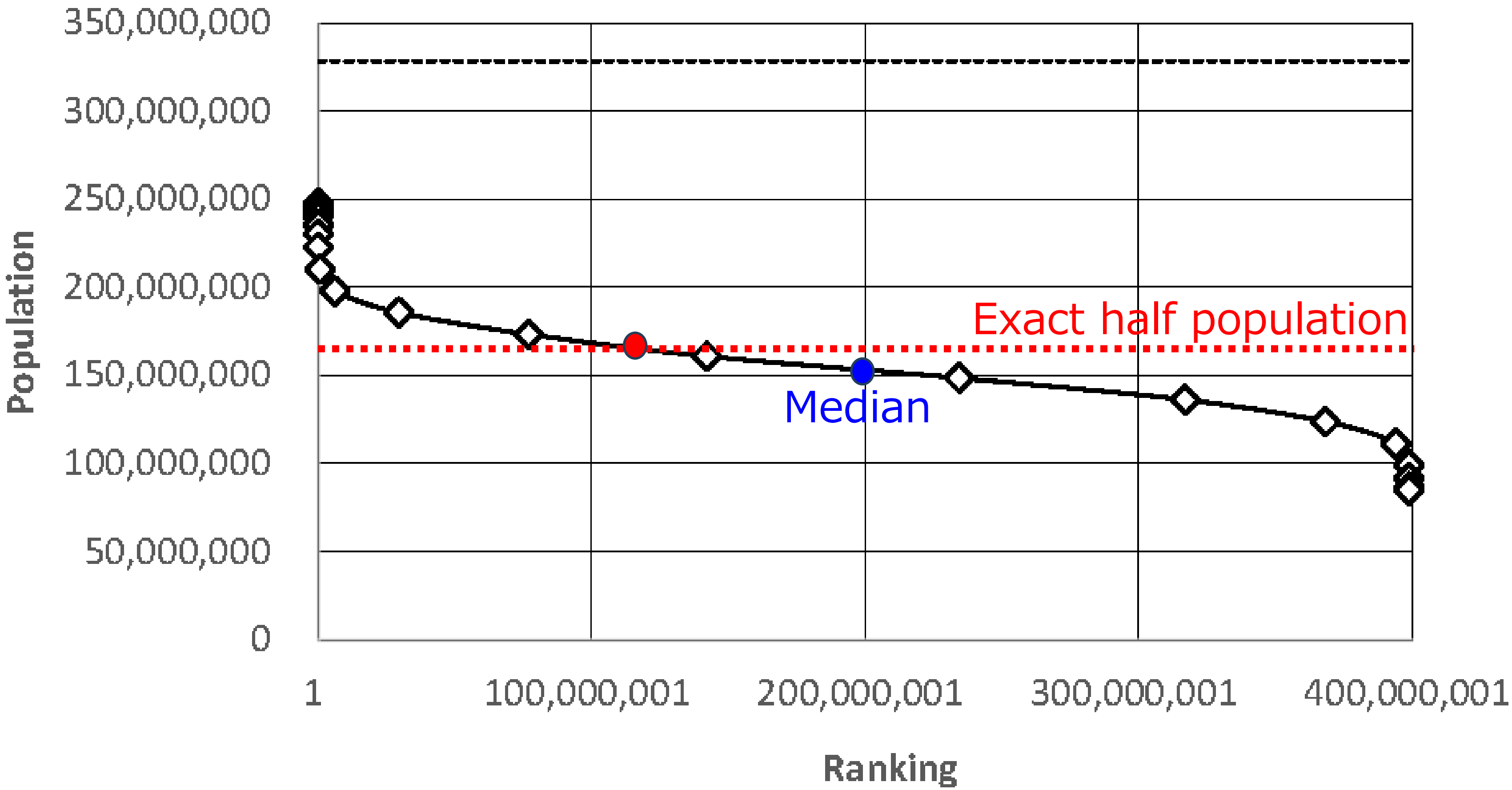
The least populated 24 states self-avoiding tour



Total population: 85,077,802
(Exact half population: 164,285,537)

AFSA

Our recent ZDD-based algorithm shows the distribution of 398,924,116 feasible solutions.



Why easy and not easy?

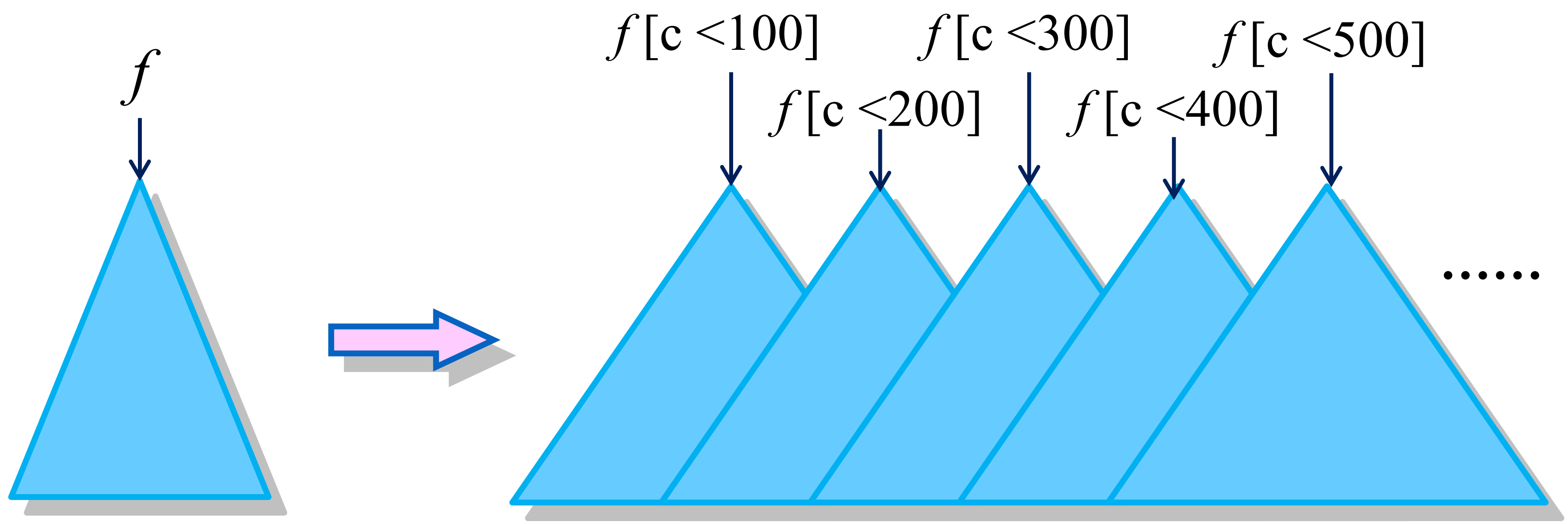
AFSA

- **Easy tasks by using ZDDs:** **(Linear-time for ZDD size)**
  - Counting **number of solutions**. (6,876,928 ways)
  - Finding **shortest/longest paths**. (11,698 / 18,040 miles)
  - Computing the **average length** of all feasible solutions.
- **Seems easy but still not easy tasks:**
  - Counting all paths **less than the average length**.
  - Finding the **median of** all feasible solutions.
  - Show **ranking** of a given solution.
  - Constructing ZDDs for **all paths no more than 10% increase** from the shortest path.
  - Constructing ZDDs enumerating the **top 5% solutions**.

Because ZDDs are indexed in a lexicographical order, but not indexed in a cost-oriented order.

"ZDD-based histogram" for cost-oriented indexing

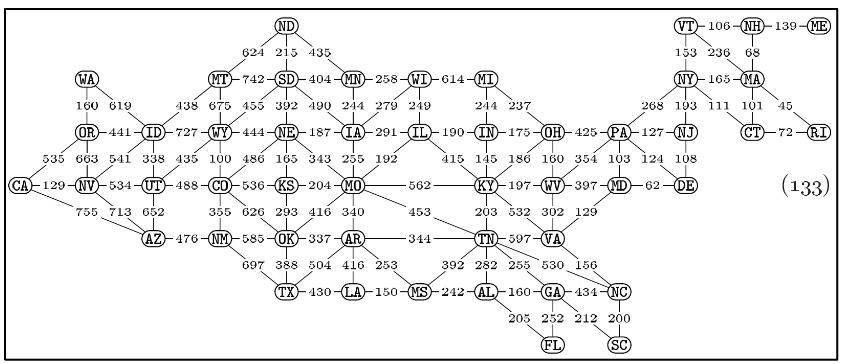
- **If we can efficiently generate ZDDs of cost-bounded solutions from the ZDD of all feasible solutions, then we may construct a "ZDD-based histogram".**
- **This is a kind of "cost-oriented index" for all feasible solutions of a combinatorial optimization problem.**



Generating ZDDs for cost-bounded solutions

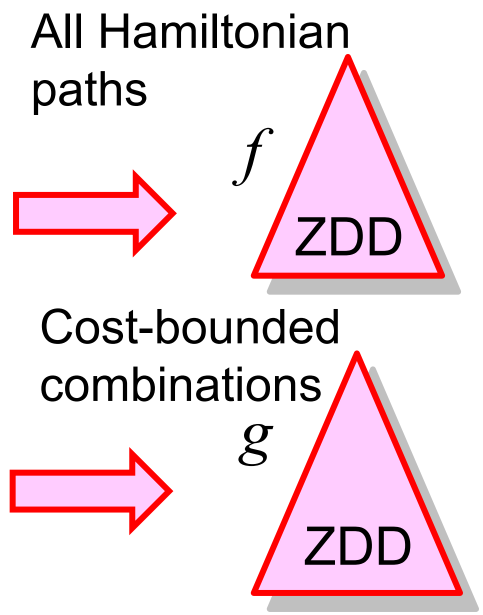
AFSA

- We can very efficiently construct ZDD f of all Hamiltonian paths (without costs) by using Knuth's (frontier-based) algorithm. (→ for the US map instance, only 0.03 sec to generate ZDD)
- We may construct another ZDD g for the cost constraint, and apply intersection between the two ZDDs to generate output ZDD h.



(PB-constraint)

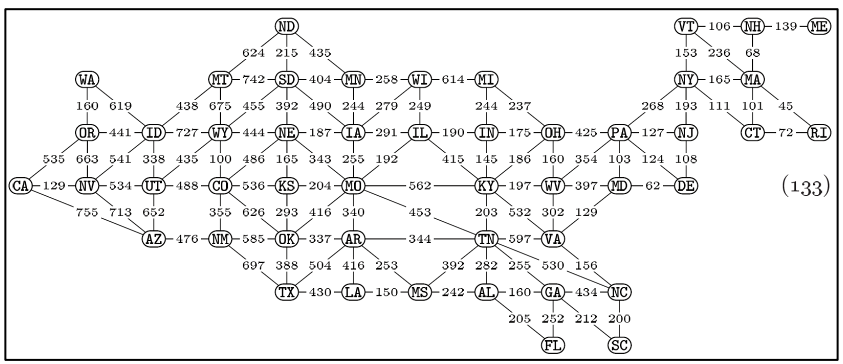
$$\sum c_i x_i \leq b$$



Generating ZDDs for cost-bounded solutions

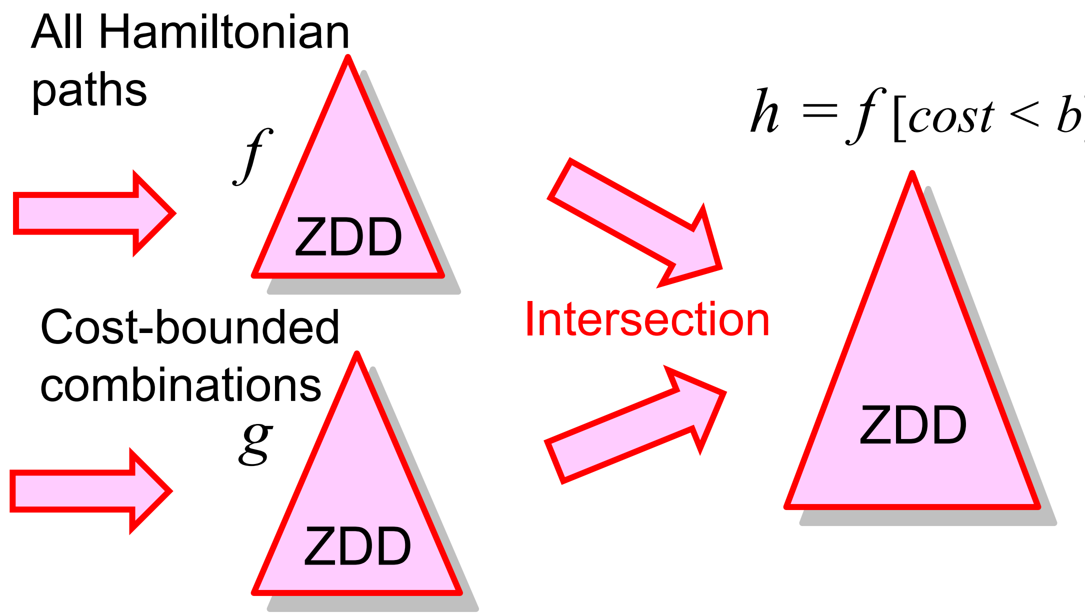
- We can very efficiently construct ZDD f of all Hamiltonian paths (without costs) by using Knuth's (frontier-based) algorithm.
  (→ for the US map instance, only 0.03 sec to generate ZDD)
- We may construct another ZDD g for the cost constraint, and apply intersection between the two ZDDs to generate output ZDD h.

It seems easy but ...



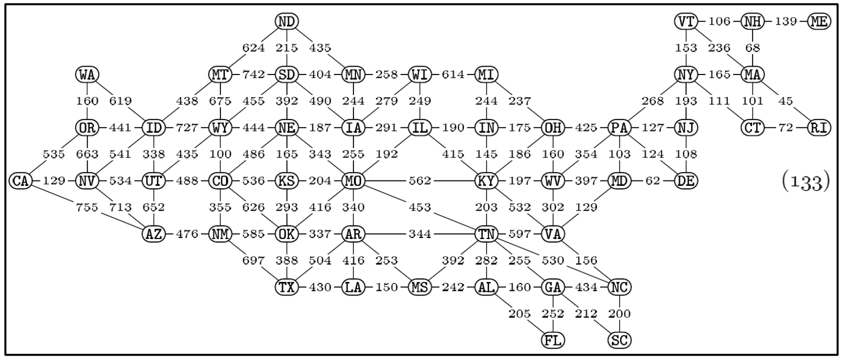
(PB-constraint)

$$\sum c_i x_i \leq b$$



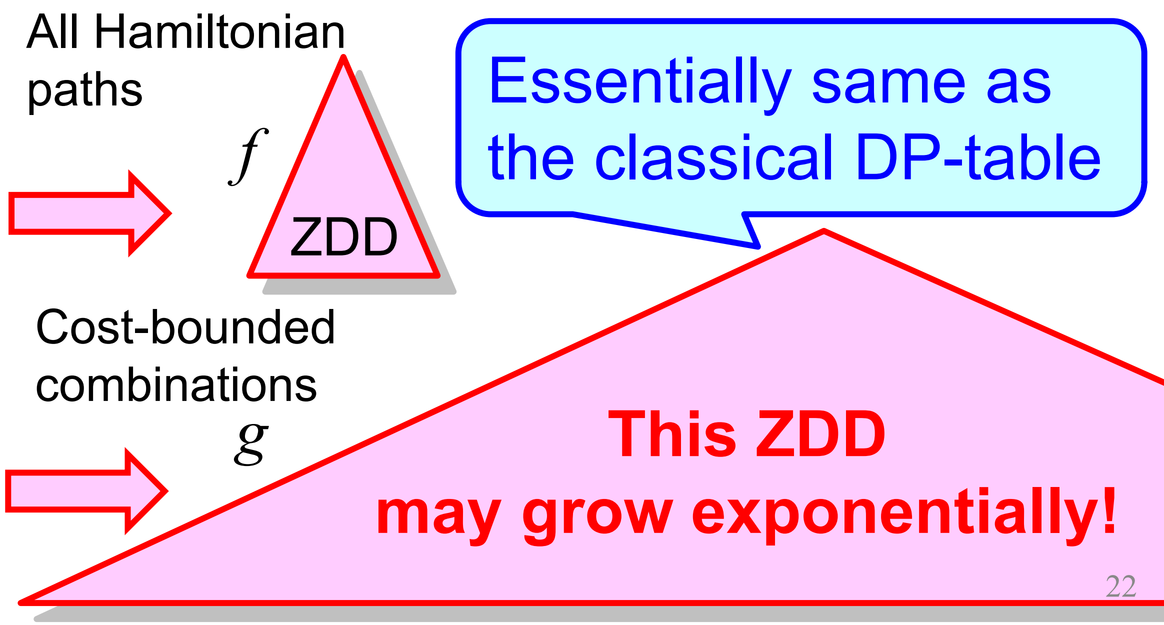
Generating ZDDs for cost-bounded solutions

- We can very efficiently construct ZDD f of all Hamiltonian paths (without costs) by using Knuth's (frontier-based) algorithm.
(→ for the US map instance, only 0.03 sec to generate ZDD)
- We may construct another ZDD g for the cost constraint, and apply intersection between the two ZDDs to generate output ZDD h.



(PB-constraint)

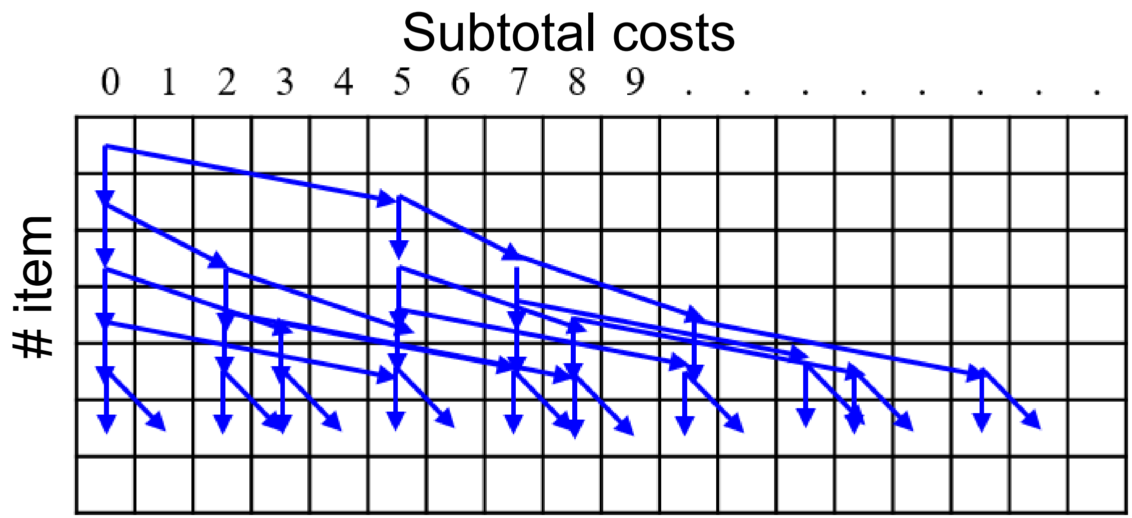
$$\sum c_i x_i \leq b$$



Classical DP Method for PB constraint problem

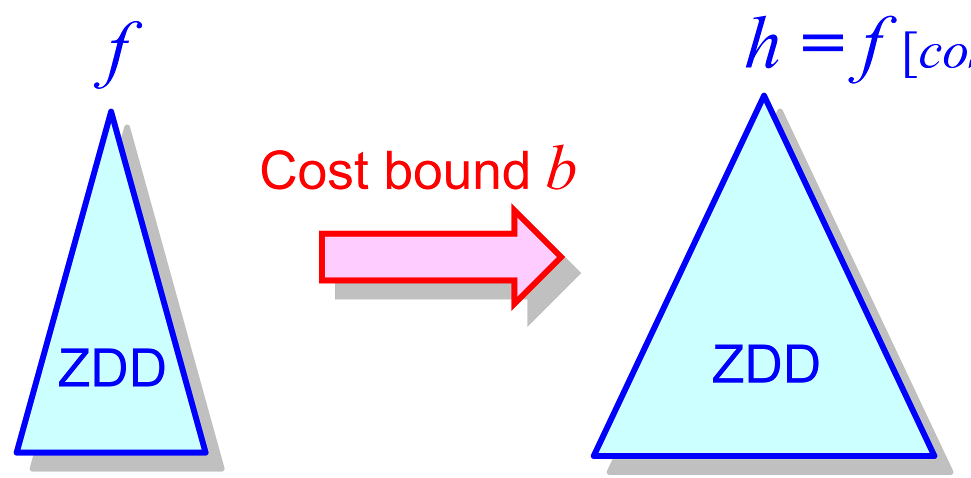
AFSA

- A classical method with dynamic programming using a DP table to store the subtotal costs for each decision.
  - Pseudo-polynomial time (with the total cost values)
  - Table becomes too large in practical applications:
    - Mileage
    - Financial incomes
    - Populations
  - For the US map with **"mileage cost"**, the total cost value becomes **35,461** (miles), and the **DP table may have 3,000,000 cells.**
  - **Too difficult** for the problem with "**population cost**".



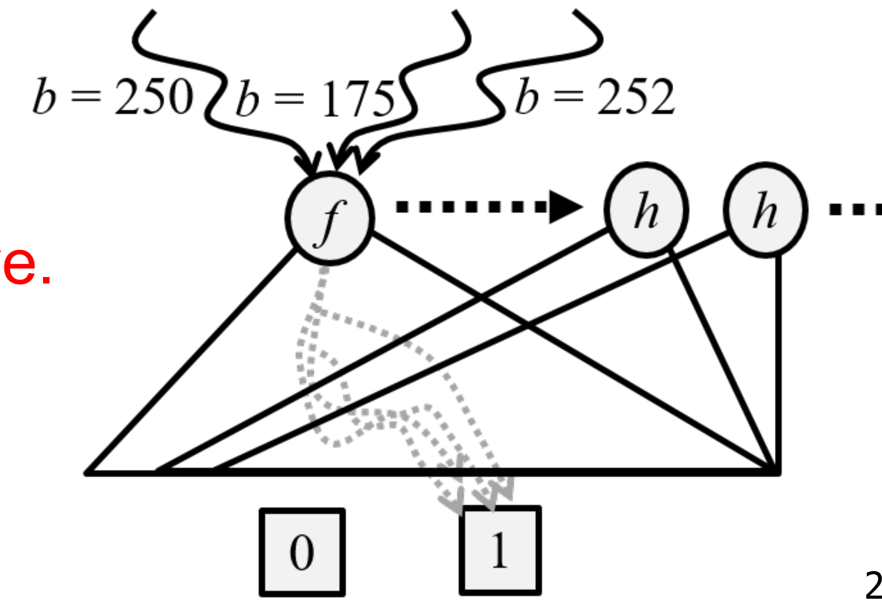
Direct ZDD construction without ZDD g

- Recursively performs a simple backtracking on the input ZDD f in a depth-first manner, and output a ZDD h.
  - On each recursive step, the problem (f, b) is divided into the two sub-problems (f_0, b) and $(f_1, b - \mathrm{cost}(x))$.
  - When reaching 1-terminal with the cost bound $b \geqq 0$, then we accept it and return 1-terminal. Otherwise, we reject it and return 0-terminal.



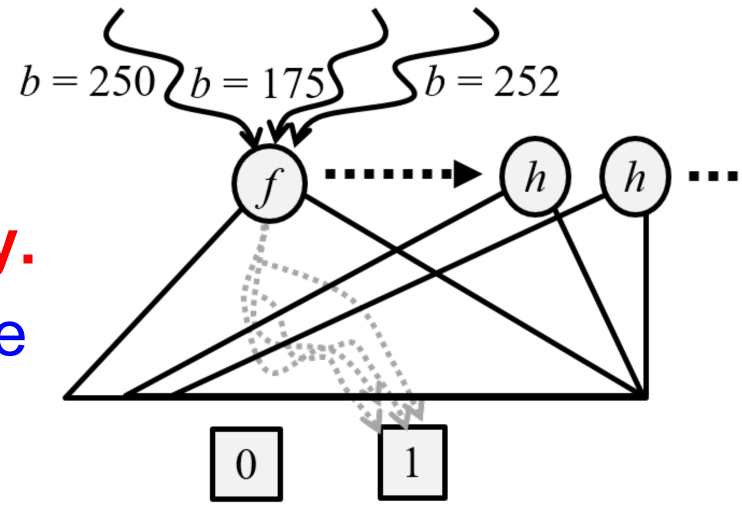
Limitation of conventional memoizing

- Conventional memoizing is not very effective for the cost-bounded cases, because the subtotal cost of used items may be different from one at the first visit.
  - In such cases, the result may not be the same, and thus we should check a pair of (f, b) as a key to the memo.
  - When cost values are large and have wide distributions, the probability of memo-hitting is significantly low, and this method is not very effective.
  - Essentially same as using the classical DP table.



Key idea of our proposed method

- If we revisit a same ZDD node f with a cost bound ***b'*** different from the first bound ***b***, the result ZDD node h may not be the same.
- **but if *b* and *b'* are very close, the result *h* becomes the same with a high possibility.**
- More formally, the result ***h*** must be the same if there is no solution with a cost between ***b*** and ***b'***.



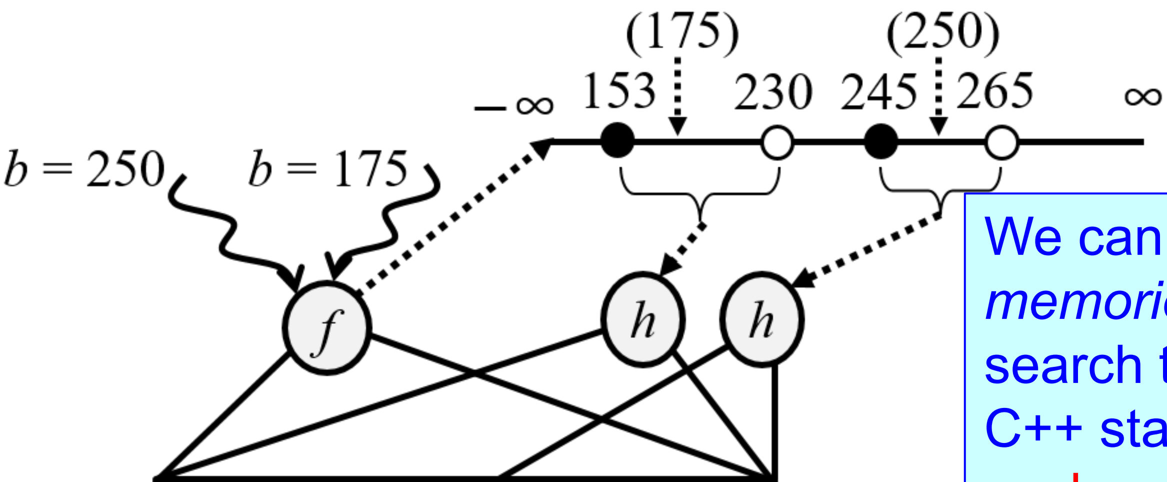
accept_worst(f, b) : the worst (highest) cost of an accepted solution in ***h***.
reject_best(f, b) : the best (lowest) cost of one rejected for ***h*** but in ***f***.

We can guarantee the same result ***h*** for ***b*** and ***b'*** if and only if :

$$accept\_worst(f, b) \leqq b' < reject\_best(f, b)$$

Interval-memoized backtracking

- For each ZDD node f, we prepare a **numerical-ordered memory** to store the intervals of the two cost bounds.
  - *accept_worst* → black dot ●, *reject_best* → white dot ○.
  - if we revisit f with b in the interval [●, ○) , then we avoid new recursive call and immediately return the result at the first visit.



We can implement it as *numerical-ordered memories* using self-balancing binary search trees, available in std::map of GNU C++ standard library. → $O(\log m)$ time for each read/write in average.

Another problem: how to know the interval (***accept_worst, reject_best***) ?
→ We can easily compute it in the recursive process.

Algorithm with interval-memoizing

```
BacktrackIntMemo(ZDD f, int b)
// returns a triple (ZDD h, int accept_worst, reject_best)
{
    if f = [0] return ([0], −∞, ∞)
    if f = [1] then
        if b ≥ 0 return ([1], 0, ∞)
        else return ([0], −∞, 0)
    (h, aw, rb) ← memo[f, b]; if exists return (h, aw, rb)
    // f consists of (x, f0, f1)
    (h0, aw0, rb0) ← BacktrackIntMemo(f0, b)
    (h1, aw1, rb1) ← BacktrackIntMemo(f1, b − cost(x))
    h ← ZDD(x, h0, h1) // applying ZDD reduction
    aw ← max(aw0, aw1 + cost(x))
    rb ← min(rb0, rb1 + cost(x))
    memo[f, [aw, rb)] ← h
    return (h, aw, rb)
}
```

Returns not only the ZDD but also the interval $[aw, rb)$

If b in the interval $[aw, rb)$, reuse the last result h.

We can compute aw, rb in a constant steps from the two children's results aw_0, rb_0, and aw_1, rb_1

Memoize the computation result h.

For $b = -\infty$: it returns empty set, and ***reject_best*** shows the min cost.
For $b = +\infty$: it returns f, and ***accept_worst*** shows the max cost.

→ Our algorithm integrates the two classical methods: BB and DP.

Hamiltonian paths for US mileage map

- Knuth's US 48 state adjacent graph (from ME to WA)
  - Exactly enumerated millions of lower-cost solutions in 0.1 sec.
  - 10 to 600 times faster than using conventional memoizing.
  - 100 times faster than existing ASP solver "clingo" [Gebster2012].

Contiguous US map graph ($|V|$: 48, $|E|$: 105) with mileage costs

cost bound (ratio)	proposed method (IntervalMemo) #solutions	$\lvert ZDD \rvert$	time(sec)	(NaiveMemo) time(sec)	clingo time(sec)
11,698 (+0%)	1	47	0.029	1.083	10.784
11,814 (+1%)	8	99	0.029	1.077	5.243
11,931 (+2%)	28	152	0.033	1.086	7.028
12,282 (+5%)	388	1,001	0.031	1.115	8.783
12,867 (+10%)	16,180	9,679	0.035	1.179	12.080
14,037 (+20%)	939,209	72,808	0.098	1.431	26.276
15,207 (+30%)	4,525,541	99,759	0.126	1.719	40.463
16,377 (+40%)	6,702,964	38,548	0.055	1.901	39.015
17,547 (+50%)	6,876,526	4,934	0.029	1.828	36.879
18,040 (+54%)	(*) 6,876,928	3,616	0.029	1.836	37.031

(*): contains all feasible solutions. ($S_f = S_h$)

Hamiltonian paths for 10x10 grid graph

- 10×10 grid graph with uniform-random cost in [1000, 2000).
  - Exactly enumerated quadrillions of lower-cost solutions in an hour.
  - **Extracted top-10Tera solutions from 1.4Peta feasible ones.**

10×10 grid graph (V: 121, E: 220)

bound b (ratio)	#solutions	ZDD $\|h\|$	proposed method	
			time(sec)	#calls
170,010 (1.00)	1	120	0.588	997,797
171,710 (1.01)	416,589	276,180	0.896	1,641,231
173,410 (1.02)	270,414,340	10,388,829	20.667	23,437,909
175,110 (1.03)	26,560,896,936	89,730,352	219.796	186,280,687
178,511 (1.05)	10,319,390,767,690	586,360,102	1,684.215	1,183,335,939
183,611 (1.08)	623,456,177,103,148	1,154,540,999	3,411.512	2,318,089,817
187,011 (1.10)	1,311,263,635,264,660	1,002,804,299	2,980.704	2,009,425,775
190,411 (1.12)	1,442,845,484,382,530	460,708,572	1,255.781	923,313,563
195,512 (1.15)	1,445,778,909,234,550	3,599,172	5.565	7,224,627
(*) 198,385 (1.17)	1,445,778,936,756,068	498,417	0.664	996,835

(*): maximum cost. (here $h = f$)

Self-avoiding 24 States tour to cover population

AFSA

Our ZDD-based algorithm could get the distribution of all 398,924,116 feasible solutions.

Table 2. Results for 24 states self-avoiding tours with population costs

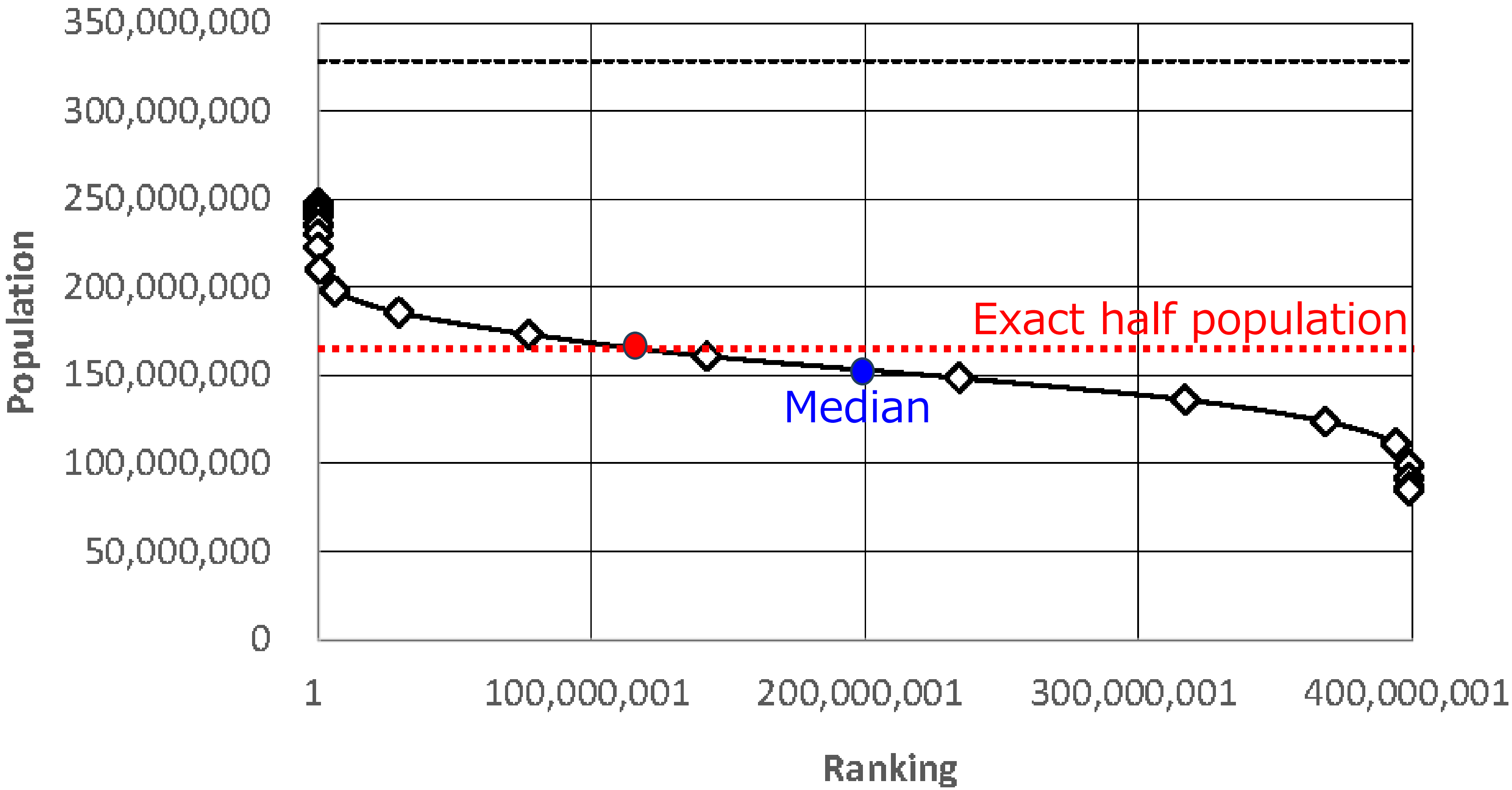
Contiguous US map graph ($|V|$: 48, $|E|$: 105) with population costs

lower cost bound (ratio)	proposed method (IntervalMemo) #solutions	|ZDD|	time(sec)	(NaiveMemo) time(sec)
247,542,080 (100%)	1	24	0.085	78.861
242,591,238 (98%)	11	46	0.085	77.516
235,164,976 (95%)	223	545	0.087	76.789
222,787,872 (90%)	36,438	8,421	0.092	79.056
210,410,768 (85%)	747,341	39,260	0.126	82.907
198,033,664 (80%)	6,151,634	117,160	0.222	87.126
185,656,560 (75%)	29,613,872	238,176	0.410	143.170
160,902,352 (65%)	142,020,633	399,070	0.612	289.366
136,148,144 (55%)	317,105,606	330,516	0.463	467.883
123,771,040 (50%)	368,379,152	201,716	0.275	516.095
111,393,936 (45%)	394,219,874	103,542	0.153	526.322
99,016,832 (40%)	398,776,535	43,577	0.099	522.995
91,590,569 (37%)	398,919,281	29,560	0.089	524.485
85,077,802 (34.37%)	(*) 398,924,116	26,798	0.088	520.014

(*): contains all feasible solutions. ($S_f = S_h$)

Distribution of the solutions in terms of population

AFSA



Future direction of my interests

Integration of
"Enumeration, Optimization, and Satisfiability" techniques.