Adaptive Bulk Search: Solving Quadratic Unconstrained Binary Optimization Problems on Multiple GPUs

Ryota Yasudo, Koji Nakano, Yasuaki Ito (Hiroshima University, Japan)

Masaru Tatekawa, Ryota Katsuki, Takashi Yazane, Yoko Inaba (NTT DATA Corporation, Japan)

HIROSHIMA UNIVERSITY

NTT DATA

Presentation at ICPP2020

Quadratic Unconstrained Binary Optimization (QUBO)

An NP-hard problem of finding an n-bit vector $X = x_0 x_1 \cdots x_{n-1}$ with the minimum energy.

input: an $n \times n$ symmetric weight matrix $W = (W_{ij})$

minimize: energy $E(X) = X^{\mathrm{T}} W X = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} W_{ij} x_i x_j$

$x_i \in \{0, 1\}$

Weight matrix W $(n = 4)$

	0	1	2	3
0	W_{00}	W_{01}	W_{02}	W_{03}
1	W_{10}	W_{11}	W_{12}	W_{13}
2	W_{20}	W_{21}	W_{22}	W_{23}
3	W_{30}	W_{31}	W_{32}	W_{33}

- **Example**

$$E(X) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} W_{ij} x_i x_j$$

Weight matrix W

	0	1	2	3
0	1	−1	2	−2
1	−1	5	3	4
2	2	3	−3	−5
3	−2	4	−5	4

	0	1	2	3
0	x_0x_0	$-x_0x_1$	$2x_0x_2$	$-2x_0x_3$
1	$-x_1x_0$	$5x_1x_1$	$3x_1x_2$	$4x_1x_3$
2	$2x_2x_0$	$3x_2x_1$	$-3x_2x_2$	$-5x_2x_3$
3	$-2x_3x_0$	$4x_3x_1$	$-5x_3x_2$	$4x_3x_3$

	0	1	2	3
0	1	0	2	−2
1	0	0	0	0
2	2	0	−3	−5
3	−2	0	−5	4

$$\begin{aligned} E(1011) = {} & 1 - 0 + 2 - 2 \\ & -0 + 0 + 0 + 0 \\ & +2 + 0 - 3 - 5 \\ & -2 + 0 - 5 + 4 \\ = {} & -8 \end{aligned}$$

Why is QUBO important?

- Many NP problems can easily be converted to QUBO
  - a QUBO solver is a solver for a wide range of combinatorial optimization problems

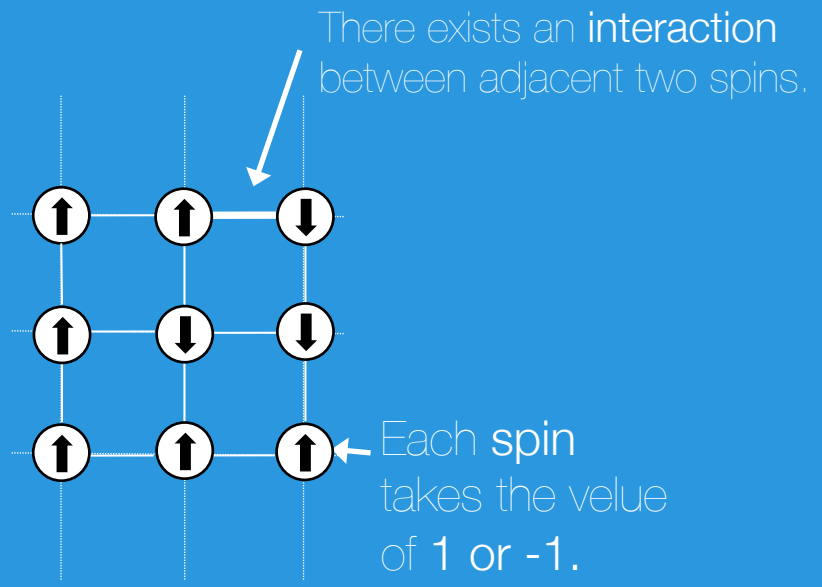
MAX-CUT, TSP, etc.

Weight matrix W

	0	1	2	3
0	W_{00}	W_{01}	W_{02}	W_{03}
1	W_{10}	W_{11}	W_{12}	W_{13}
2	W_{20}	W_{21}	W_{22}	W_{23}
3	W_{30}	W_{31}	W_{32}	W_{33}

QUBO instance

equivalent



Ising model

used in quantum annealing

- Solving QUBO can be an alternative method to quantum annealing (QA).

Our main contributions

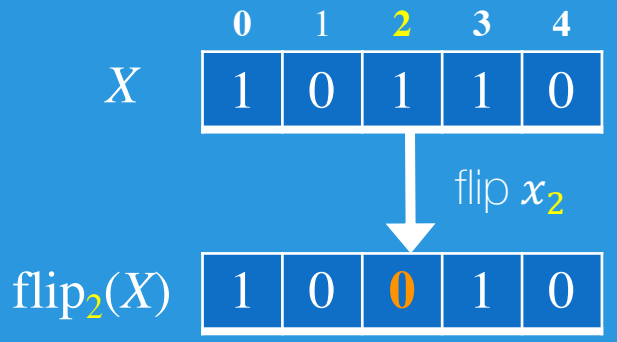
- Propose ***Adaptive Bulk Search*** (***ABS***), a new search method for solving QUBO
  - ***O*(1)** cost for evaluating energy per solution
- Present an implementation of ABS on **multiple GPUs** and show results for **MAX-CUT, TSP, and random** QUBO instances.
  - **1.2T search rate** (it can evaluate 1.2×10^{12} solutions per second).

Local search

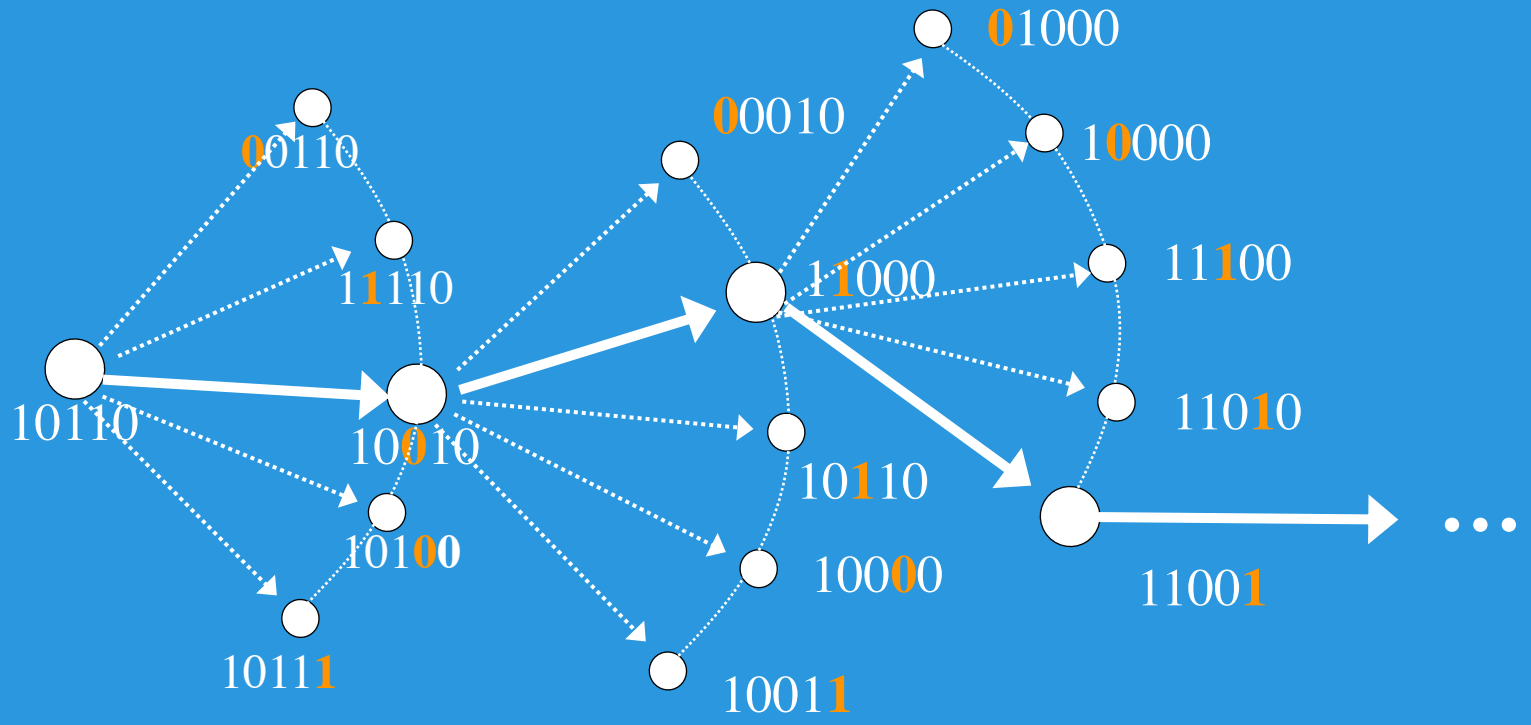
- For an n-bit vector $X = x_0 x_1 \cdots x_{n-1}$, let us define

$$\mathrm{flip}_k(X) = x_0 x_1 \cdots x_{k-1} \overline{x_k} x_{k+1} \cdots x_{n-1}$$

k-th bit is flipped



We can use this function for obtaining neighbor solutions.



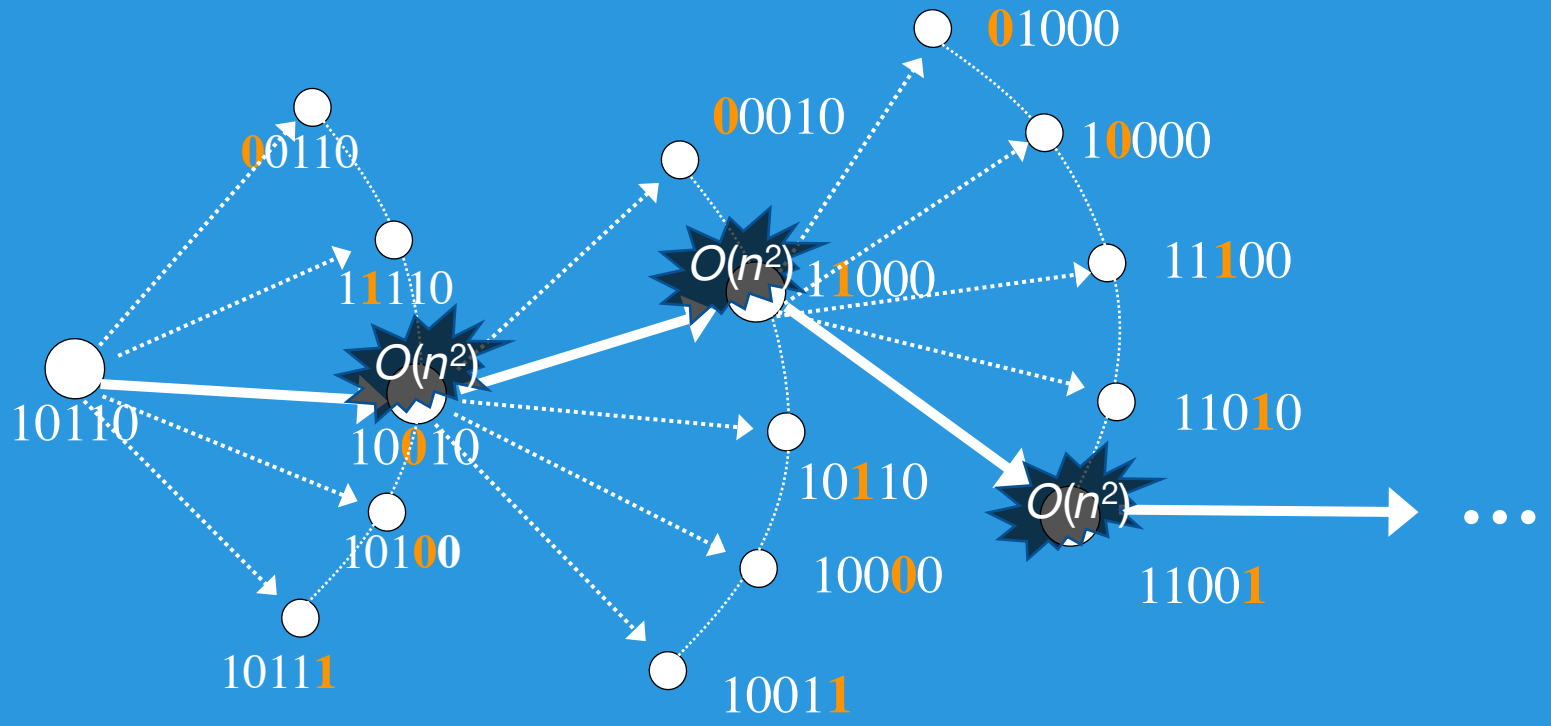
Naive Evaluation of Energy $E(X)$

$$E(X) = X^{\mathrm{T}} W X = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} W_{ij} x_i x_j$$

- Evaluating $E(X)$ requires $O(n^2)$ cost.

Weight matrix W

	0	1	2	3	4
0	W_{00}	W_{01}	W_{02}	W_{03}	W_{04}
1	W_{10}	W_{11}	W_{12}	W_{13}	W_{14}
2	W_{20}	W_{21}	W_{22}	W_{23}	W_{24}
3	W_{30}	W_{31}	W_{32}	W_{33}	W_{34}
4	W_{40}	W_{41}	W_{42}	W_{43}	W_{44}



Incremental computation for $E(X)$

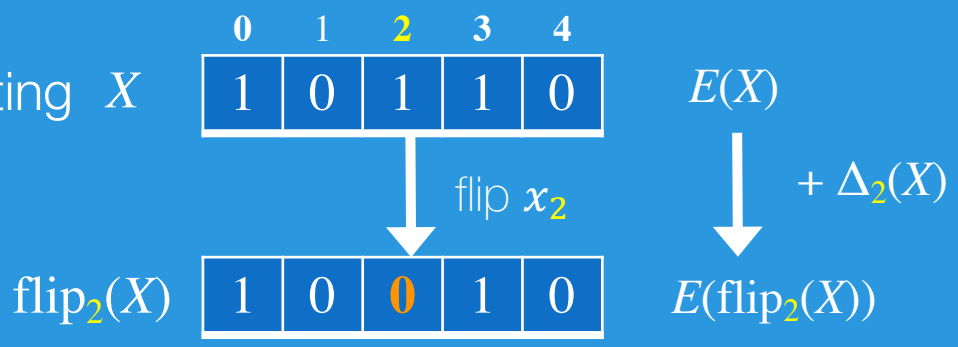
- Incremental computation decreases the cost for evaluating energy.

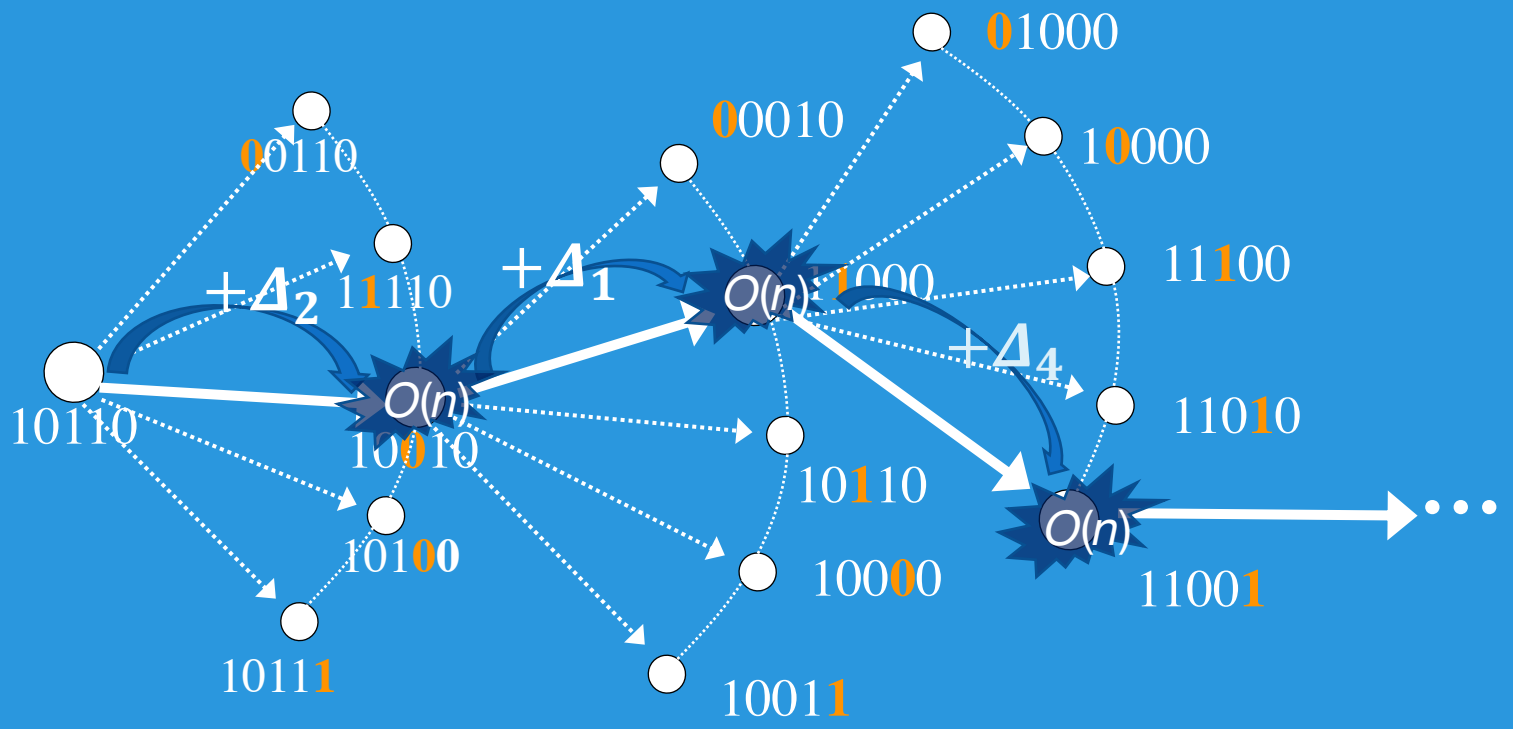
$$E(X) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} W_{ij}x_i x_j$$

$$\Delta_k(X) = E(\text{flip}_k(X)) - E(X)$$
$$= \pm\,(2\sum_{j\neq k} W_{kj}x_j + W_{kk})$$

Weight matrix W

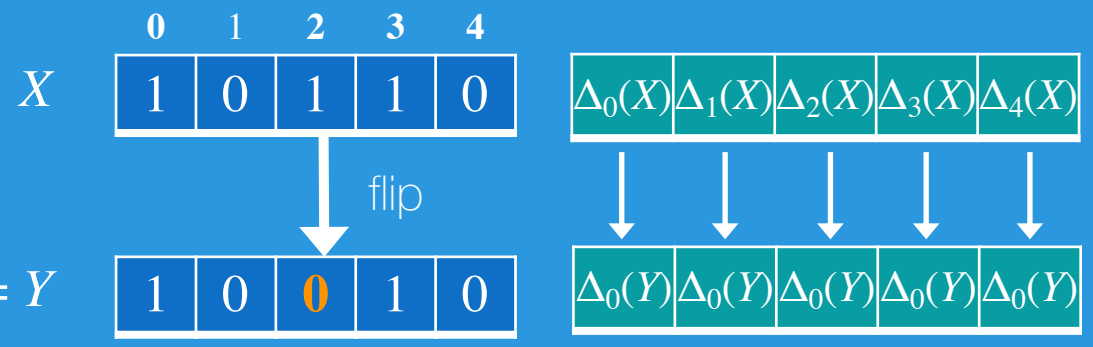
	0	1	k	3
0	W_{00}	W_{01}	W_{02}	W_{03}
1	W_{10}	W_{11}	W_{12}	W_{13}
k	W_{20}	W_{21}	W_{22}	W_{23}
3	W_{30}	W_{31}	W_{32}	W_{33}



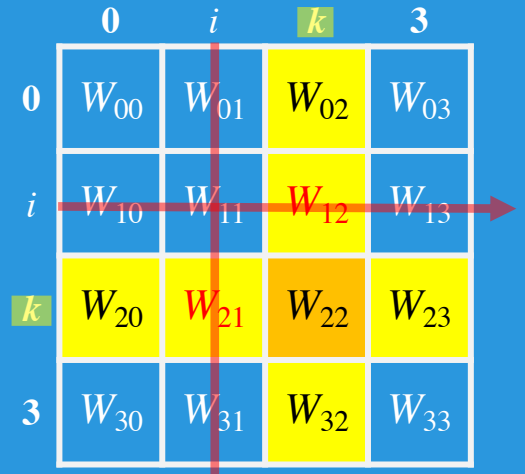


Incremental computation for $\Delta(X)$

- Let us consider the difference of Δ_k after x_i is flipped.



e.g., $n = 4, k = 2, i = 1$



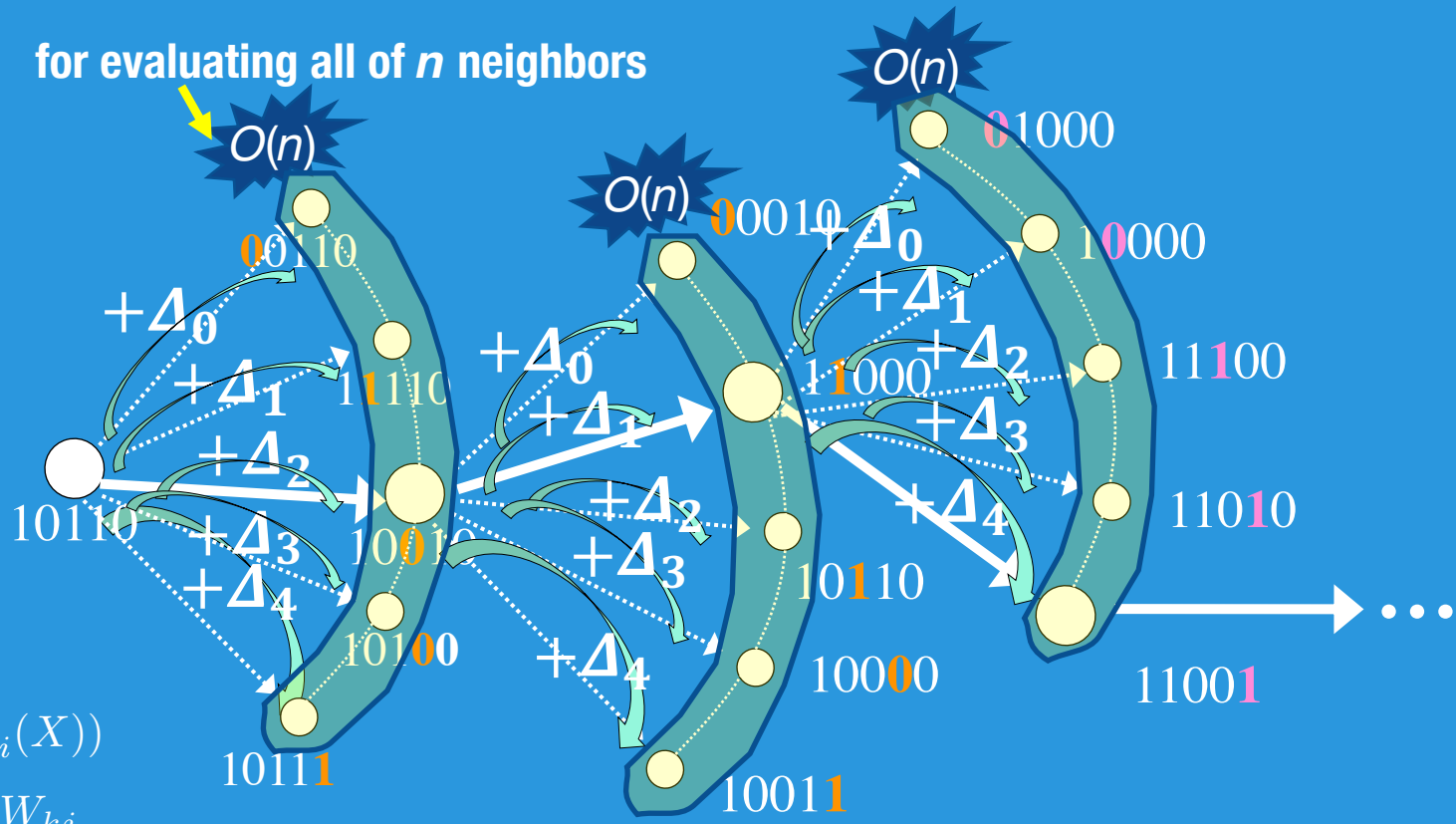
After x_i is flipped, $W_{kj}x_i$ is changed.

if $i = k$ $\quad \Delta_k(\text{flip}_i(X)) = -\Delta_k(X)$

else $\quad \Delta_k(\text{flip}_i(X)) = E(\text{flip}_k(\text{flip}_i(X))) - E(\text{flip}_i(X))$

$$= \pm(2\sum_{j\neq k} W_{kj}x_j + W_{kk}) \pm 2W_{kj}$$

$$= \Delta_k(X) \pm 2W_{kj}$$



- The cost is reduced to **O(1) per solution** by storing Δ_k for all n bits in a memory.

Selection policy

Store Δ_i for all n bits

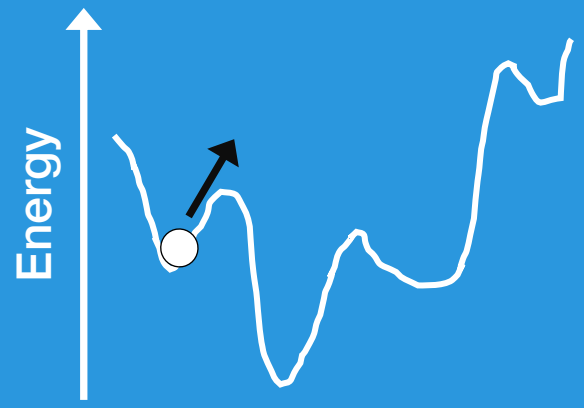
$\Delta_0(X)$	$\Delta_1(X)$	$\Delta_2(X)$	$\Delta_3(X)$	$\cdots$	$\Delta_{n-2}(X)$	$\Delta_{n-1}(X)$

We can select a bit from all the *n* bits based on Δs.

Hill Climbing

Select best one from all the *n* bits.

☹ We cannot escape from a local minimum.



Our proposed policy

Select best one from a segment of the bits.

0110110110011100

segment

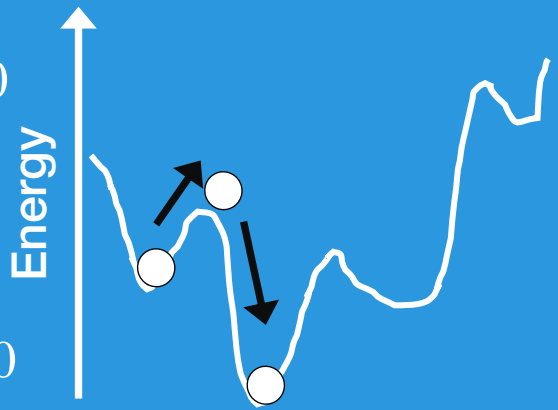
Analogous to simulated annealing (SA).

00101101100111000100011110101100 10

sometimes bad solution is selected
(high temperature)

00101101100111000100011110101100 10

always good solution is selected
(low temperature)



Our selection policy

1. **Compare Δ of bits in a segment.**

2. **Flip a bit with minimum Δ in a segment.**

3. **Repeat the same operations in the following segment.**

⋮

iteration 1

0010110110011100

segment

0110110110011100

iteration 2

0110110110011100

segment

0110110010011100

iteration 3

0110110010011100

segment

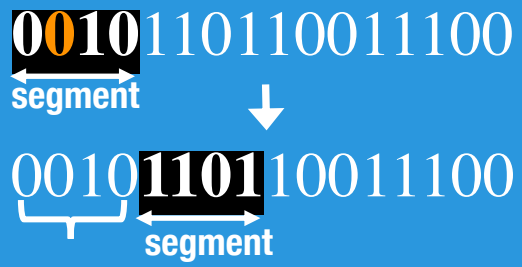
0110110000111100

⋮

Advantages

1. No random number generation (e.g., cuRAND)

2. A flipped bit is not re-flipped in the following several iterations.

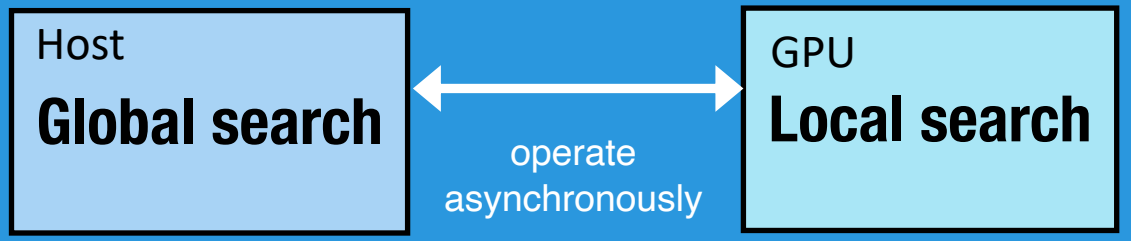


3. A bit-flipping operation is always performed in every iteration.

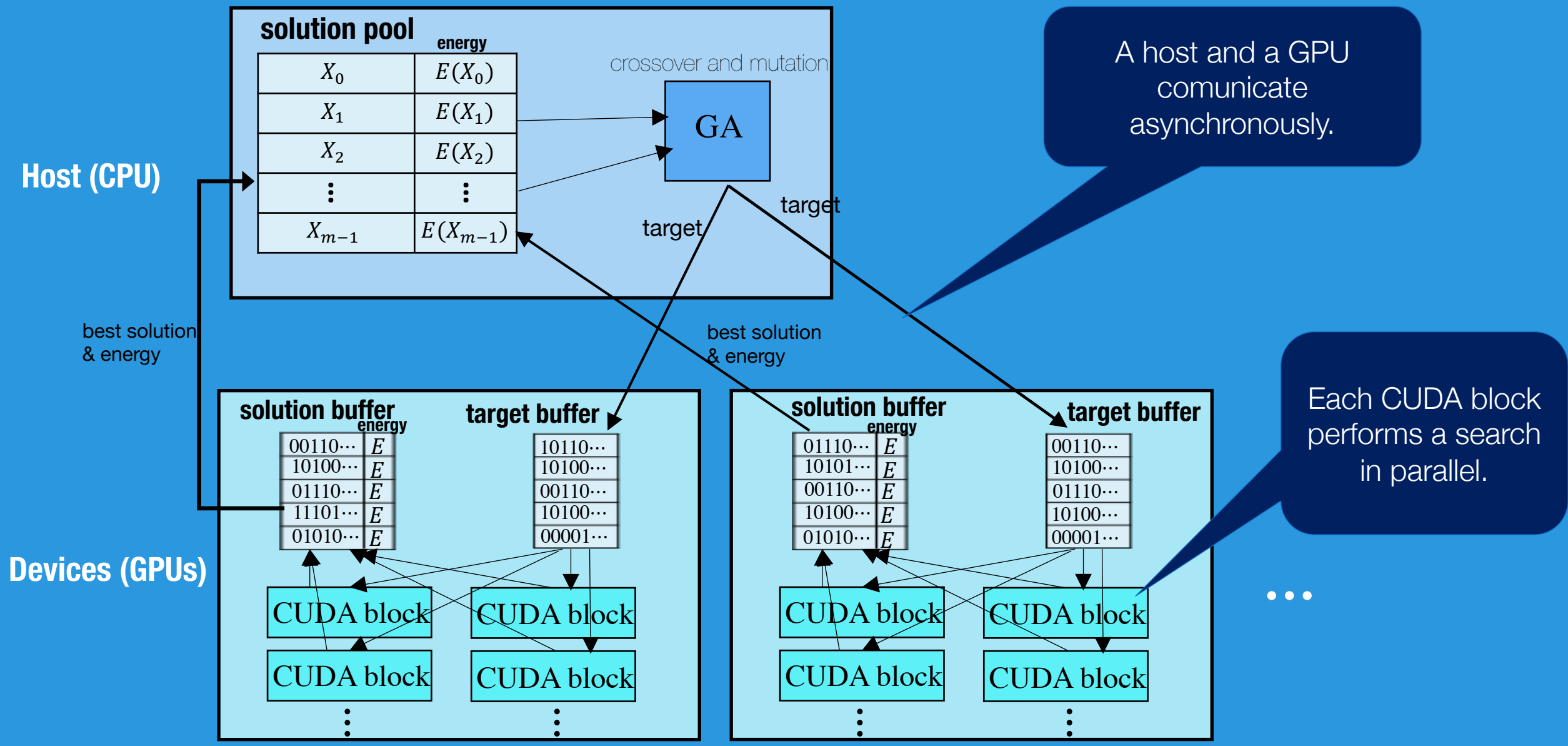
In SA, a bit may not be flipped.

Concept of Adaptive Bulk Search (ABS)

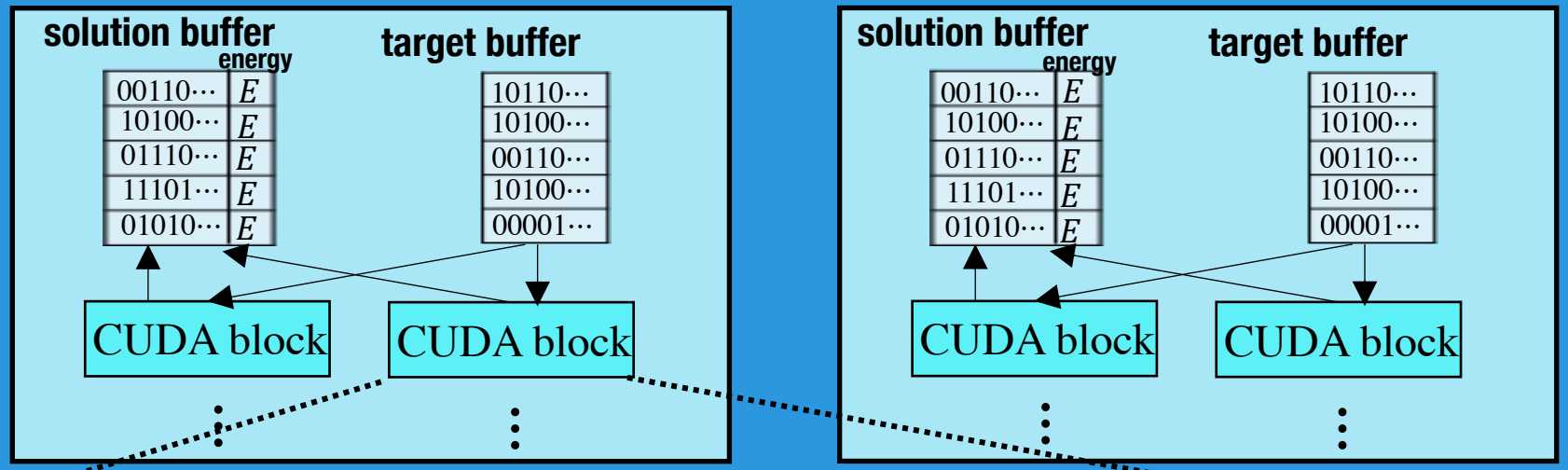
- Perform a significant number of local searches
  - cost efficient (O(1) algorithm described before)
  - full utilization of GPU computation resources
  - adaptive local search (a search algorithm can be changed)
  - combination with a global search (genetic algorithm)

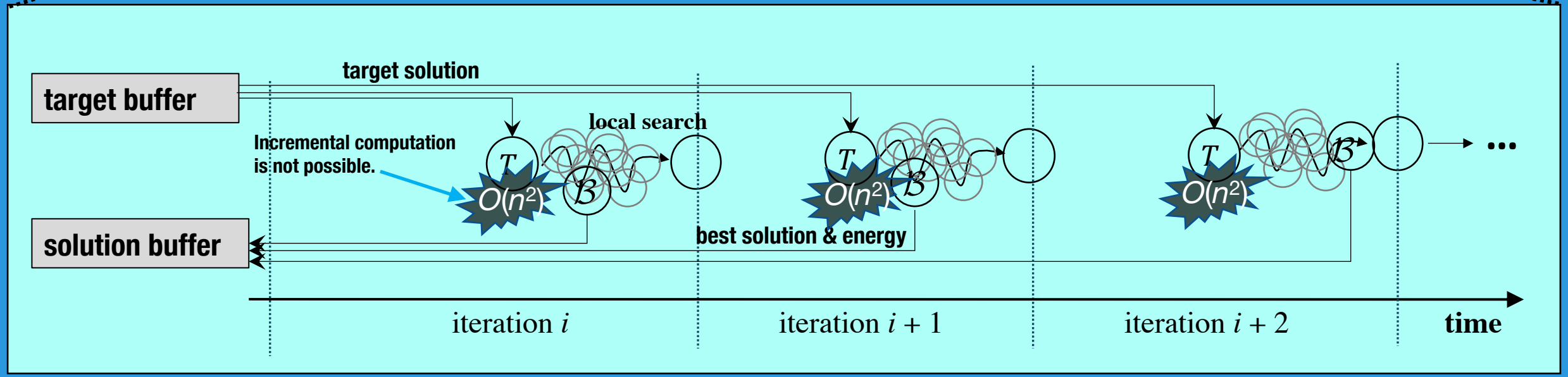


Overview of ABS

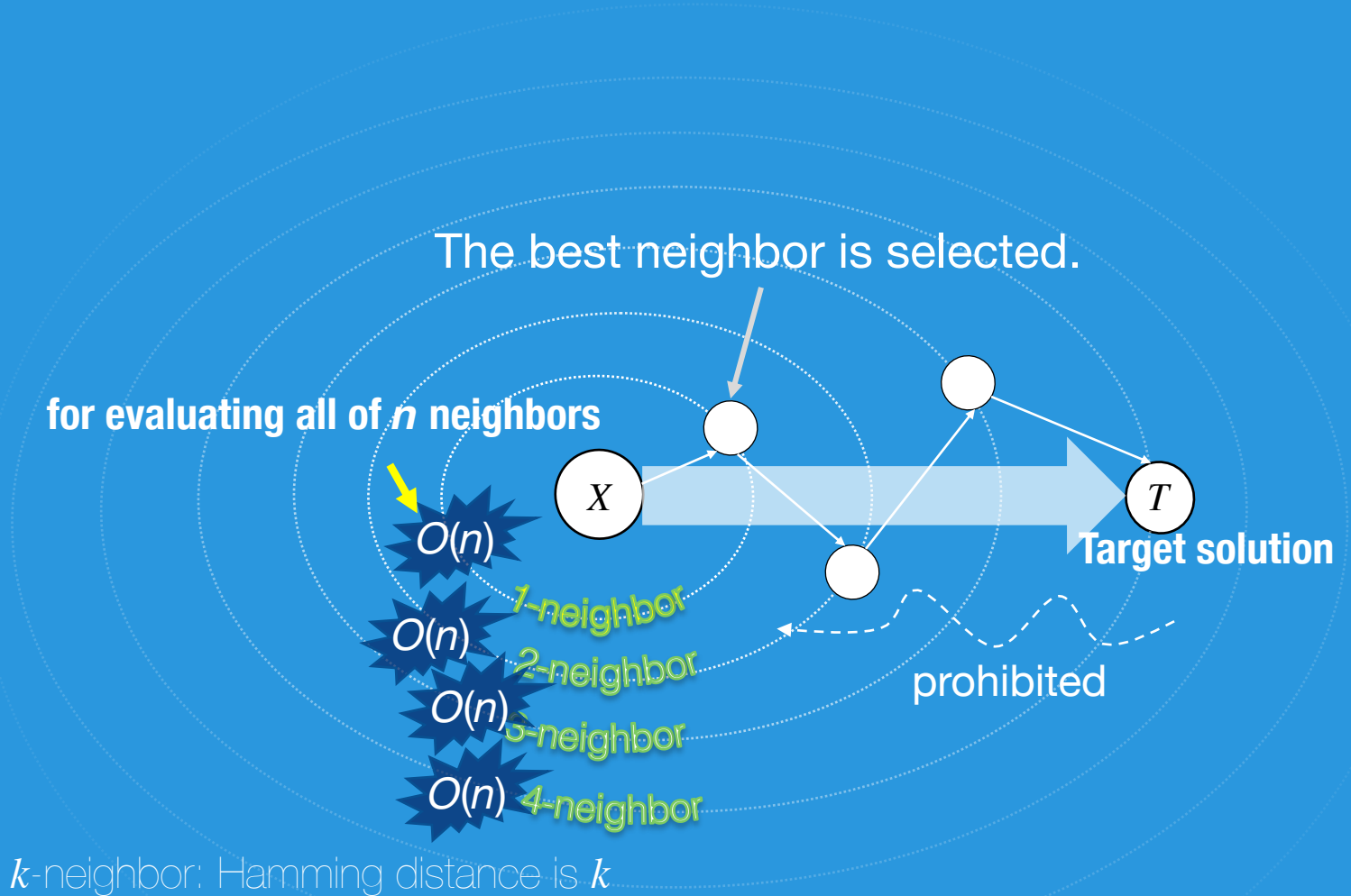


Devices (GPUs)

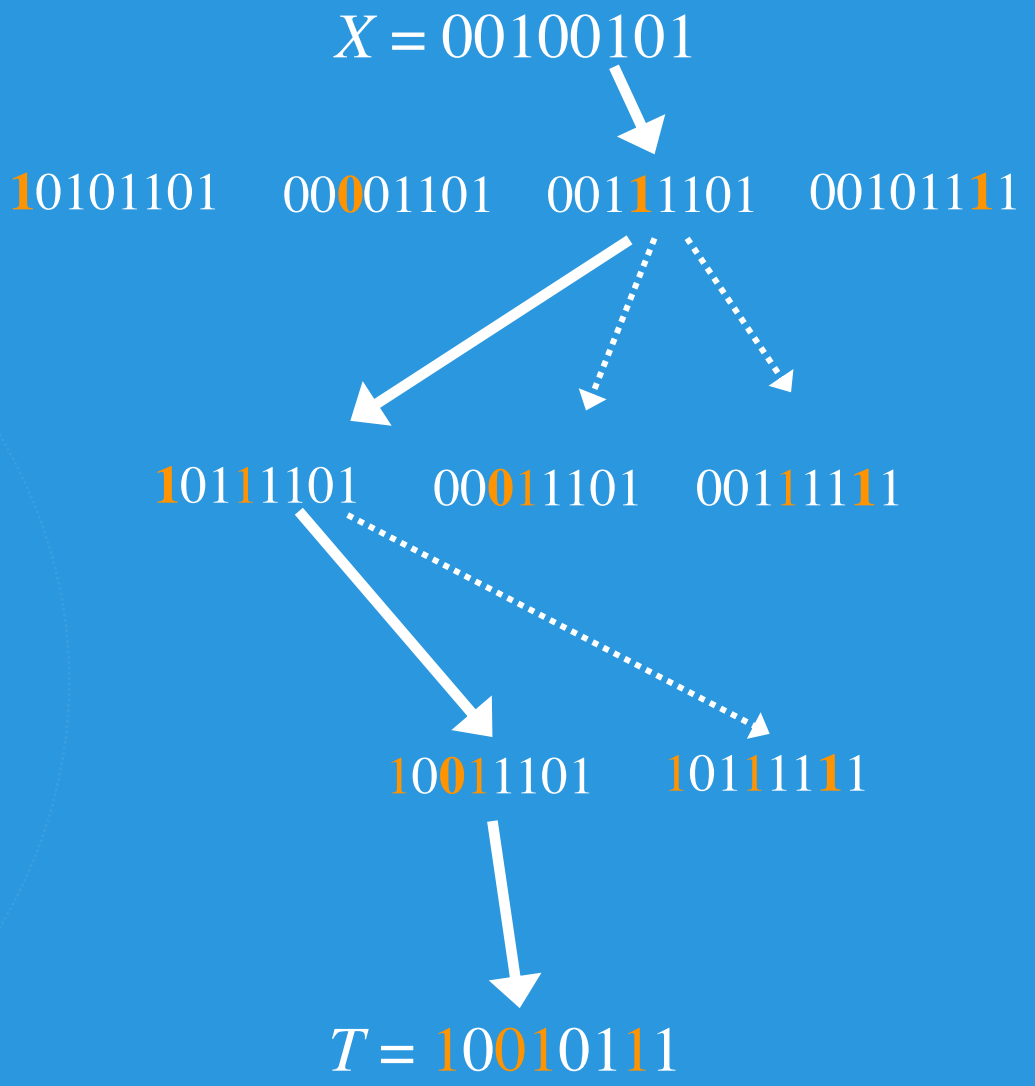




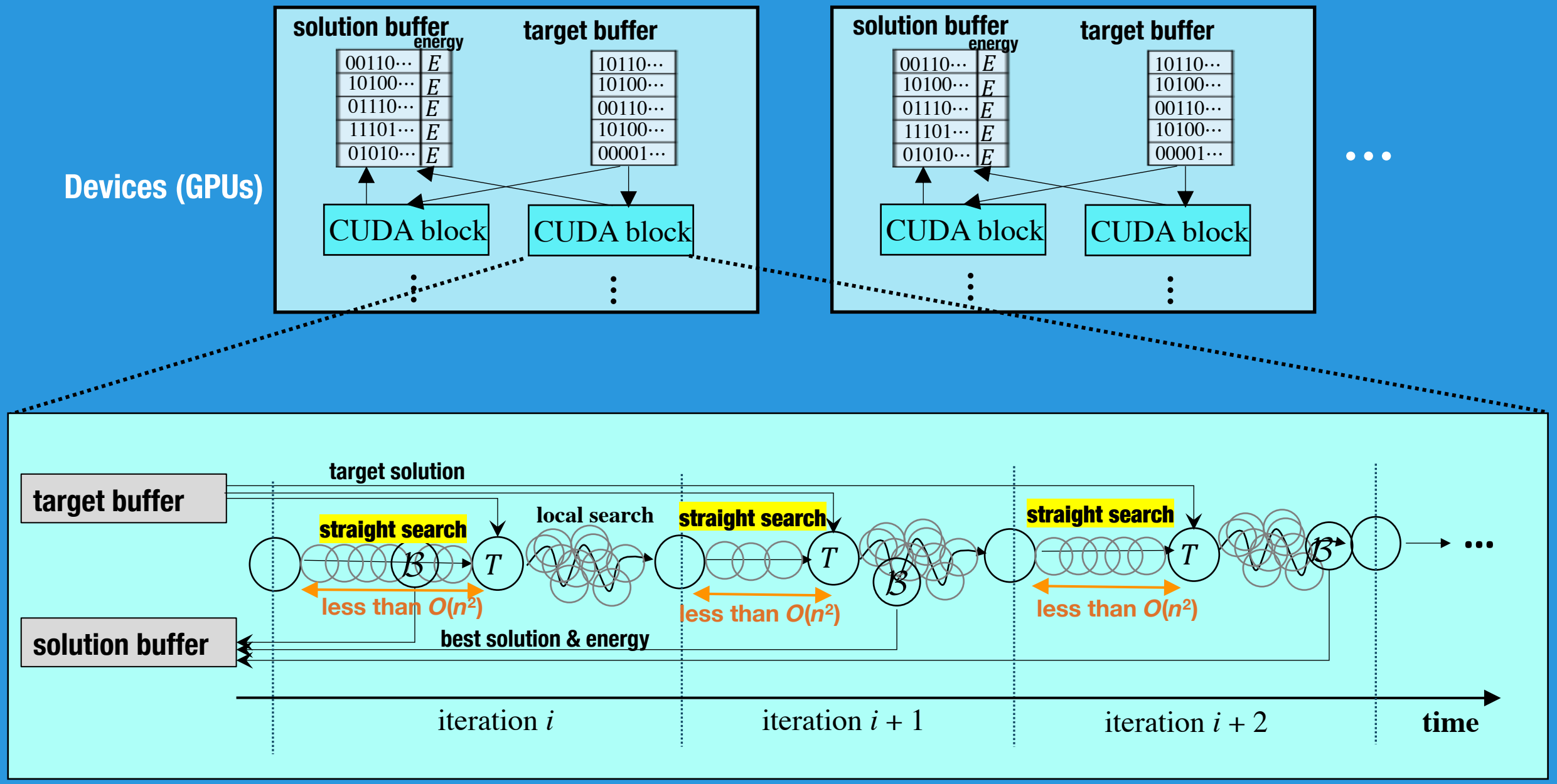
Straight Search



k-neighbor: Hamming distance is *k*



Total computational cost are less than $O(n^2)$.


Devices (GPUs)
solution buffer
energy
target buffer
00110⋯ E
10100⋯ E
01110⋯ E
11101⋯ E
01010⋯ E
10110⋯
10100⋯
00110⋯
10100⋯
00001⋯
CUDA block
CUDA block
target buffer
solution buffer
target solution
straight search
local search
straight search
straight search
less than O(n²)
less than O(n²)
less than O(n²)
best solution & energy
iteration i
iteration i + 1
iteration i + 2
time

System configuration

- 18-core Intel Core i9-9980XE CPU (3.00 GHz)
- NVIDIA GeForce RTX 2080 Ti GPU ×4 (Turing architecture, Compute capability 7.5)
  - 11GB global memory (GDDR6 SDRAM), 64-KB shared memory, 64K 32-bit registers per multiprocessor, 1024 threads per block, and 68 multiprocessors.

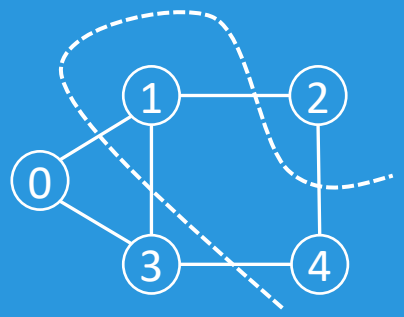
- CUDA (v10.0)

Evaluation methods

- Evaluate two metrics
  - Time-to-solution
    - Time to obtain good solutions (e.g., best-known solutions)
  - Search rate (throughput)
    - The number of energy evaluations per second

- Three benchmarks for up to 32k-bit QUBO problems.
  - MAX-CUT (maximum cut problem)
  - TSP (travelling salesman problem)
  - Randomly generated matrix instances

Time-to-solution (MAX-CUT)



weight matrix W

	0	1	2	3	4
0	–2	1	0	1	0
1	1	–3	1	1	0
2	0	1	–2	0	1
3	1	1	0	–3	1
4	0	0	1	1	–2

Graph	*n*	Type	Edge weight	Target value	Time (s)
G1	800	random	+1	Best-known	0.0723
G6	800	random	±1	Best-known	0.106
G22	2000	random	+1	99% of best-known	0.110
G27	2000	random	±1	99% of best-known	0.721
G35	2000	planar	+1	99% of best-known	0.208
G39	2000	planar	±1	99% of best-known	1.89
G55	5000	random	+1	95% of best-known	0.150
G70	10000	random	+1	95% of best-known	0.360

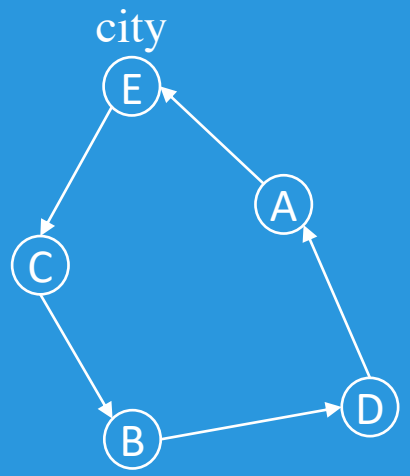
- G-set
  - https://web.stanford.edu/~yyye/yyye/Gset/

Time-to-solution (TSP)

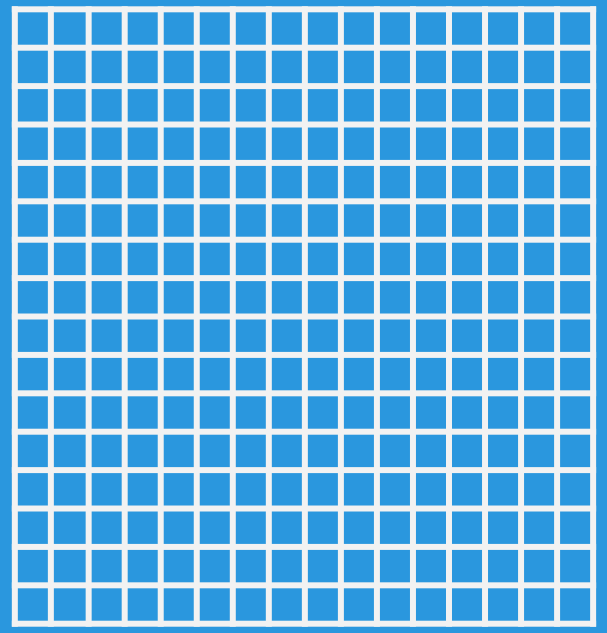
Problem	#city	*n*	Target	Time(s)
ulysses16	16	225	Best-known	0.11
bayg29	29	784	Best-known	0.69
dantzig42	42	1681	Best-known+5%	1.25
berlin52	52	2601	Best-known+5%	1.79
st70	70	4621	Best-known+10%	4.19

An *N*-city problem can be converted into $(N-1)^2$-bit QUBO.

5-city problem



16 x 16 weight matrix W



- TSPLIB
  - http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Time-to-solution (random)

Weight matrix W

	0	1	2	3
0	W_{00}	W_{01}	W_{02}	W_{03}
1	W_{10}	W_{11}	W_{12}	W_{13}
2	W_{20}	W_{21}	W_{22}	W_{23}
3	W_{30}	W_{31}	W_{32}	W_{33}

Problem	*n*	Target	Time (s)
Random	1024	Best-known	0.0172
Random	2048	Best-known	0.0413
Random	4096	Best-known	1.04
Random	16384	99% of best-known	0.417
Random	32768	99% of best-known	1.79

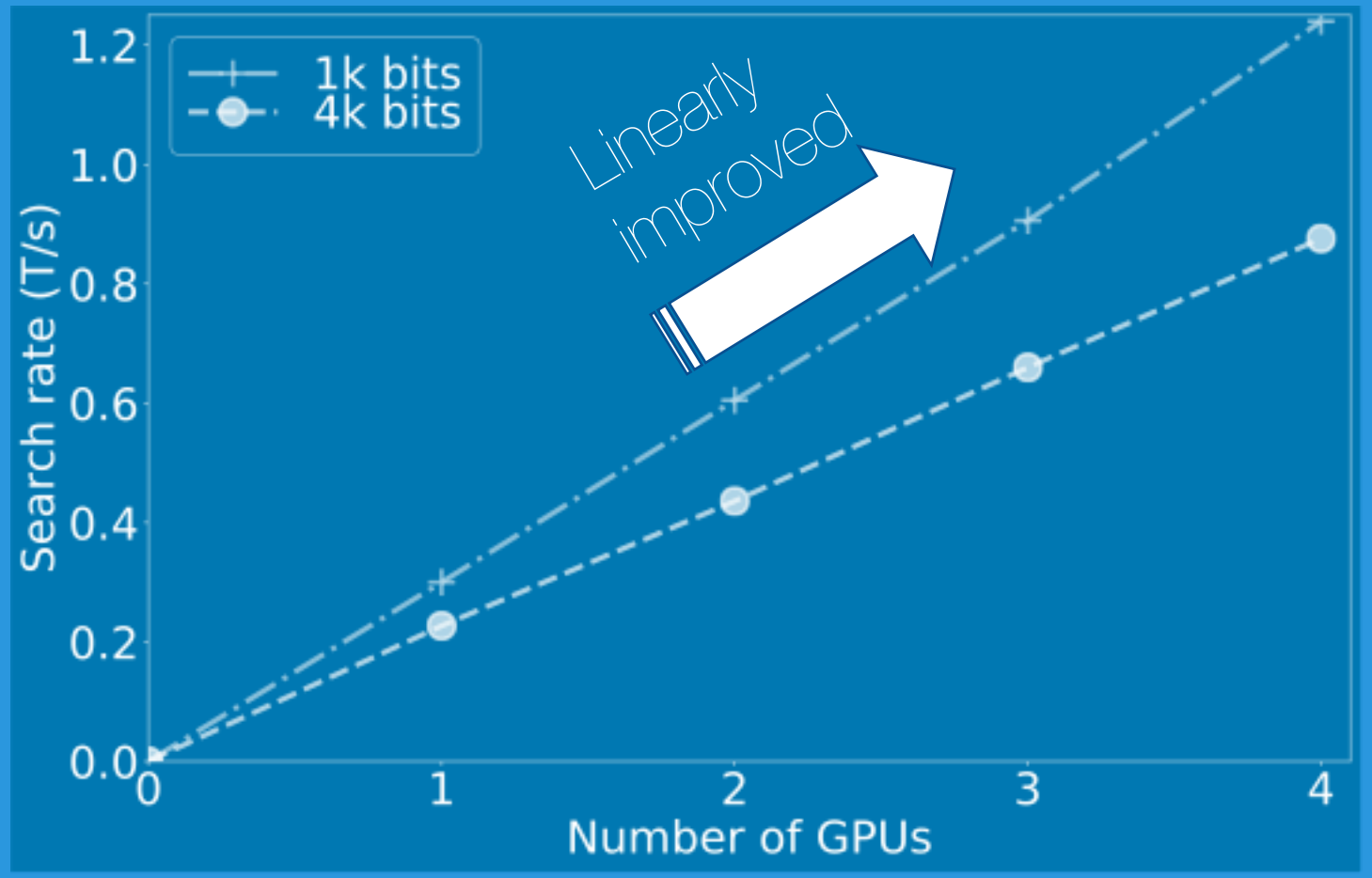
Every weight is selected from all 16-bit integers uniformly at random (–32768～32767).

Search rate (#evaluation/second)

n	Search rate [T/s]
1k	**1.24**
2k	1.01
4k	0.732
8k	0.537
16k	0.578
32k	0.439

60x faster than prior work

[Matsubara et al., "Ising-Model Optimizer with Parallel-Trial Bit-Sieve Engine", Int. Conf. on Complex, Intelligent, and Software Intensive Systems 2017]



Scalablity

Conclusions

- A QUBO solver is potentially a solver for a wide range of problems.
- We have proposed *Adaptive Bulk Search* (*ABS*), a framework for solving Quadratic Unconstrained Binary Optimization (QUBO) problems.
  - Efficient algorithm for evaluating energy
  - Local search, GA, and straight search
- We have implemented ABS on four GPUs, which supports 32k-bit QUBO problems.
  - ABS finds approximate solutions for MAX-CUT, TSP, and random QUBO problems.
  - ABS attains 1.2T search rate for 1k-bit QUBO problems.