Graph optimization algorithm with symmetry and biased host density for Order/Radix Problem

Masahiro Nakao,[†] Masaki Tsukamoto,[‡] Yoshiko Hanada[‡]

†RIKEN Center for Computational Science

‡Kansai University

Order Degree/Problem

- Our team also won awards in Order/ Degree Problem
- The algorithm we used is the same as the one in GraphGolf 2018
- Please refer to the following papers
- Masahiro Nakao et al. ``Graph optimization algorithm for low-latency interconnection networks'', Parallel Computing, July 2021
- 中尾昌広 他、最適化アルゴリズムによる低遅延相互結合網のためのグラフ構成法、情報処理学会HPC研究会、2020年3月

Final rankings

🏆 : Awarded

General Graph Widest Improvement Ranking

	Rank	Author	Number of best solutions
🏆	1	Team RK	8
	2	EvbCFfp1XB	2
	3	YK-NITTYM	1
	3	Yoshiki Satotani, Norikazu Takahashi	1
	3	Ryo Ashida	1
	3	MODAL-Team	1

General Graph Deepest Improvement Ranking

	Rank	Author	ASPL gap
🏆	1	MODAL-Team	0.0
🏆	1	Ryo Ashida	0.0
🏆	1	Yoshiki Satotani, Norikazu Takahashi	0.0
🏆	1	YK-NITTYM	0.0
🏆	1	Team RK	0.0
🏆	1	EvbCFfp1XB	0.0

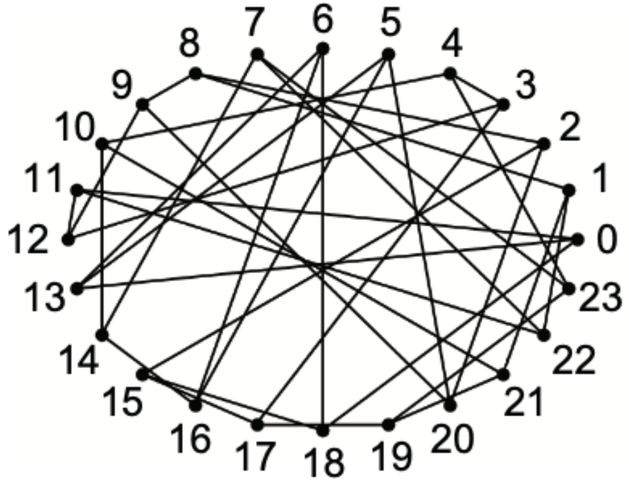
Library for Order/Degree Problem

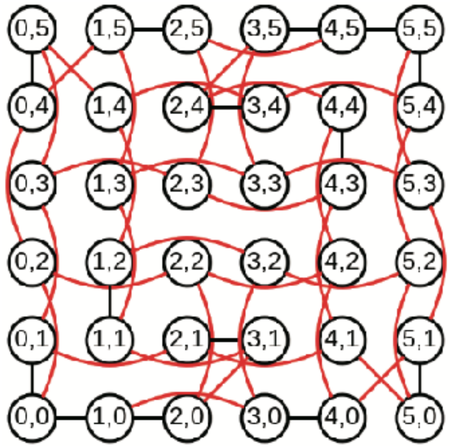
General Graph

```
#include "odp.h"
…
ODP_Init_aspl_general(…);
for(int i=0;i<ITERATIONS;i++){
  /* Optimization */
  ODP_Set_aspl(...);
}
ODP_Finalize_aspl();
```

Grid Graph

```
#include "odp.h"
…
ODP_Init_aspl_grid(…);
for(int i=0;i<ITERATIONS;i++){
  /* Optimization */
  ODP_Set_aspl(...);
}
ODP_Finalize_aspl();
```





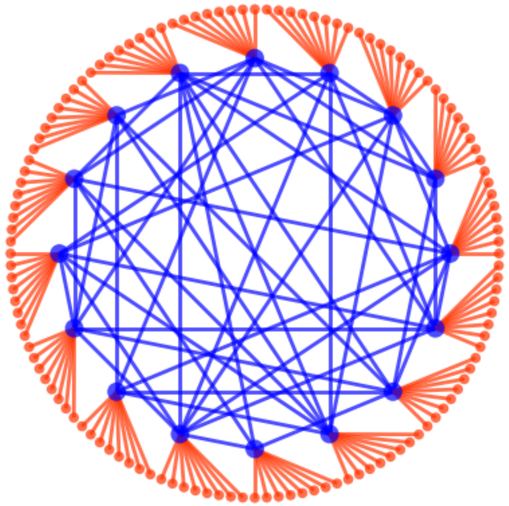
- Calculate ASPL
- Support both graphs
- C language
- Threads/MPI/CUDA

https://github.com/mnakao/ODP

Order/Radix Problem

- Biased host density and Symmetry (**by Masahiro Nakao**)
  - 中尾昌広 他、Order/Radix Problemにおける対称性とホストの偏りを利用した最適化アルゴリズムの提案、情報処理学会HPC研究会、2021年12月
  - https://mnakao.net/data/2021/HPC182.pdf
- Automatic determination of the number of switches (**by Masaki Tsukamoto**)

```
#include "orp.h"
…
ORP_Init_aspl(…);
for(int i=0;i<ITERATIONS;i++){
  /* Optimization */
  ORP_Set_aspl(...);
}
ORP_Finalize_aspl();
```



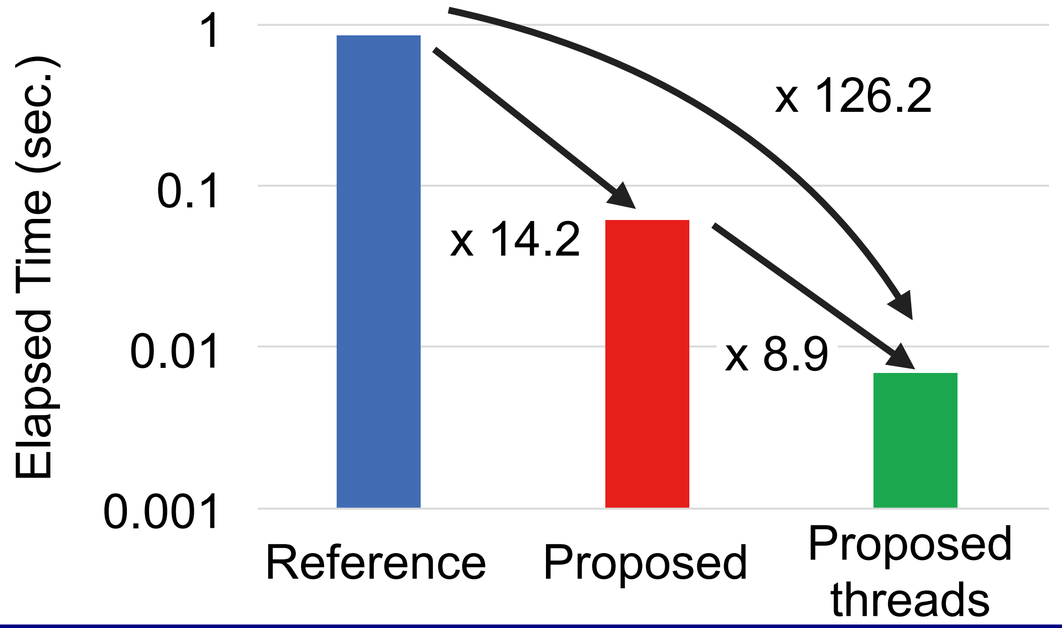
https://github.com/mnakao/ORP

- Calculate h-ASPL
- C language
- Threads

Order/Radix Problem

- Biased host density and Symmetry (**by Masahiro Nakao**)
  - 中尾昌広 他、Order/Radix Problemにおける対称性とホストの偏りを利用した最適化アルゴリズムの提案、情報処理学会HPC研究会、2021年12月
  - https://mnakao.net/data/2021/HPC182.pdf
- Automatic determination of the number of switches (**by Masaki Tsukamoto**)

```
#include "orp.h"
…
ORP_Init_aspl(…);
for(int i=0;i<ITERATIONS;i++){
  /* Optimization */
  ORP_Set_aspl(...);
}
ORP_Finalize_aspl();
```

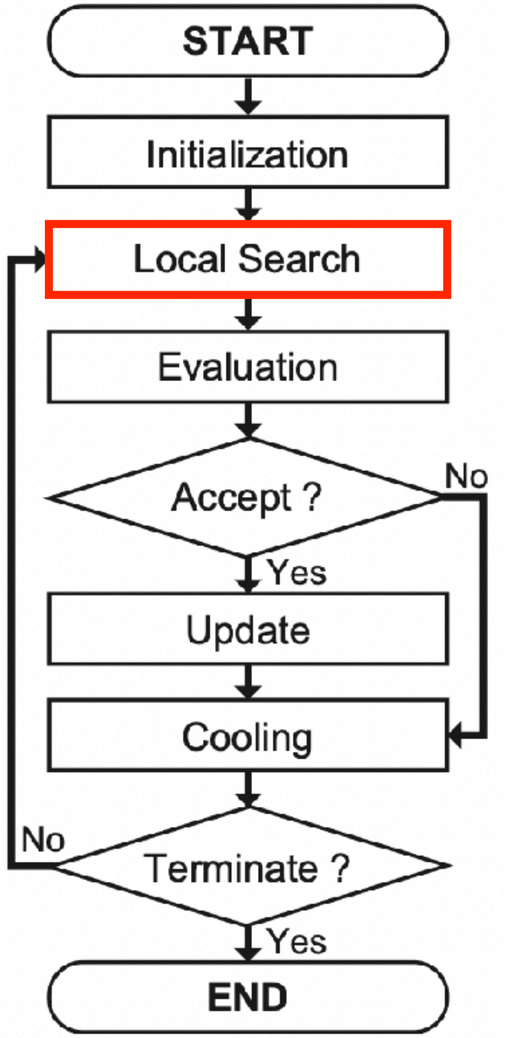


hosts = 65536
radix = 64
switches = 4096

Intel Xeon Gold 6126 (2.6GHz, 12 Cores) x 2

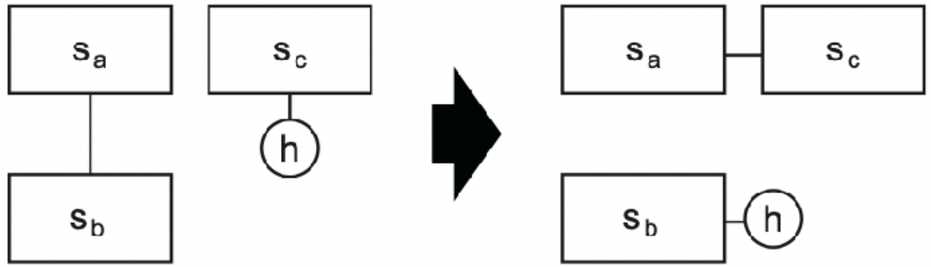
Cygnus system in University of Tsukuba

SWING and SWAP for Local Search

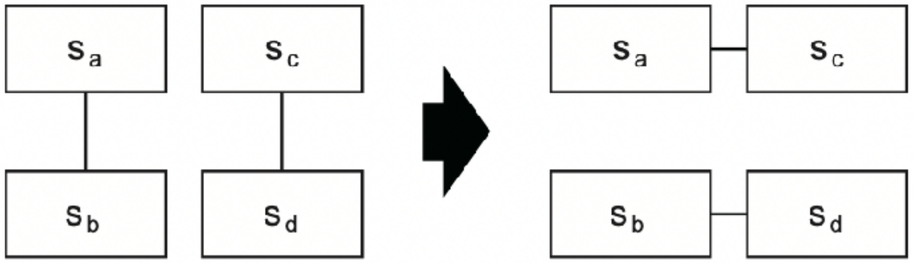


- **SWING**
  - A host and an edge are changed

s : Switch, h: Host

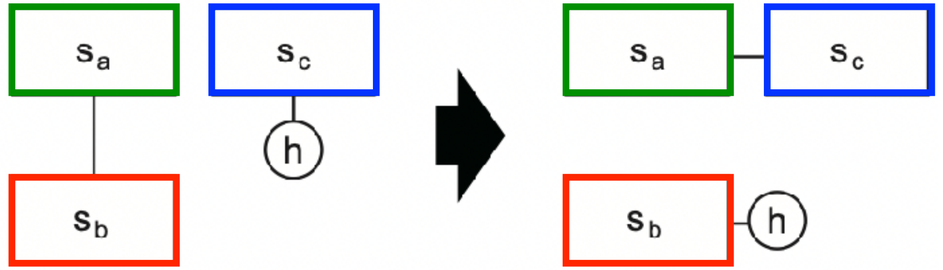


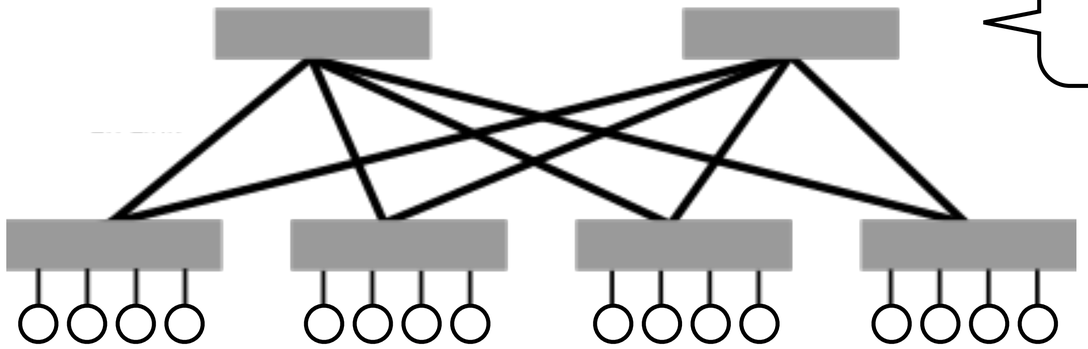
- **SWAP**
  - Two edges between switches are changed



Biased host density in SWING

- In SWING, a host in s_b is added,
  while a host in s_c is removed
- When the edge s_a - s_b is selected in SWING,
  which one becomes s_b is random

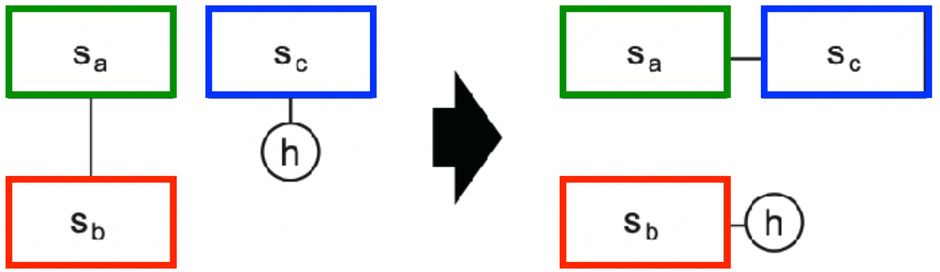




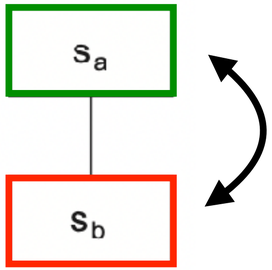
It may be a good network topology
where some switches have no hosts

Biased host density in SWING

- In SWING, a host in s_b is added, while a host in s_c is removed
- When the edge s_a - s_b is selected in SWING, which one becomes s_b is random



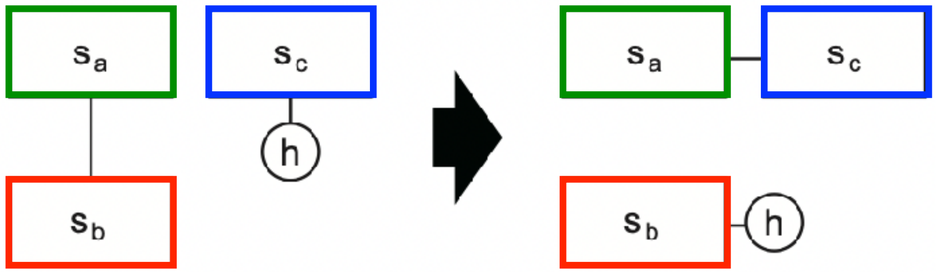
- Replace s_a and s_b so that there are more hosts adjacent to s_b before moving a host
- This operation means that a switch with many hosts will tend to have more hosts



If #switches adjacent to **sa** and **sb** are **A** and **B**, the replacement is performed with the probability of **B**/(**A**+**B**)

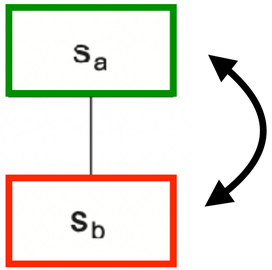
Biased host density in SWING

- In SWING, a host in s_b is added, while a host in s_c is removed
- When the edge s_a - s_b is selected in SWING, which one becomes s_b is random



called “Random Selection”

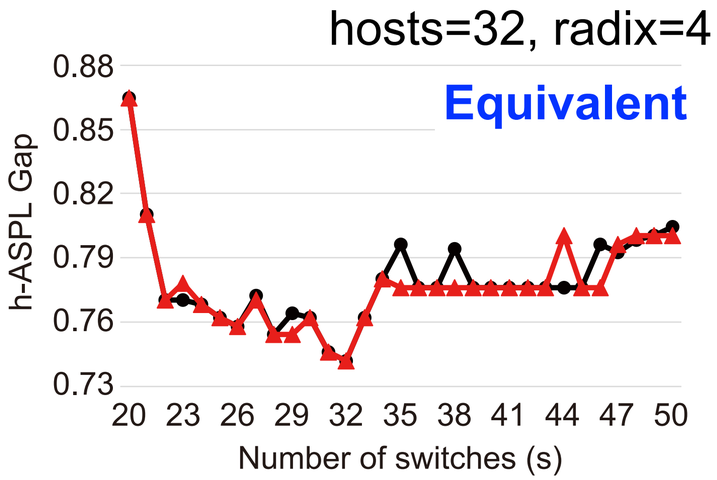
- Replace s_a and s_b so that there are more hosts adjacent to s_b before moving a host
- This operation means that a switch with many hosts will tend to have more hosts

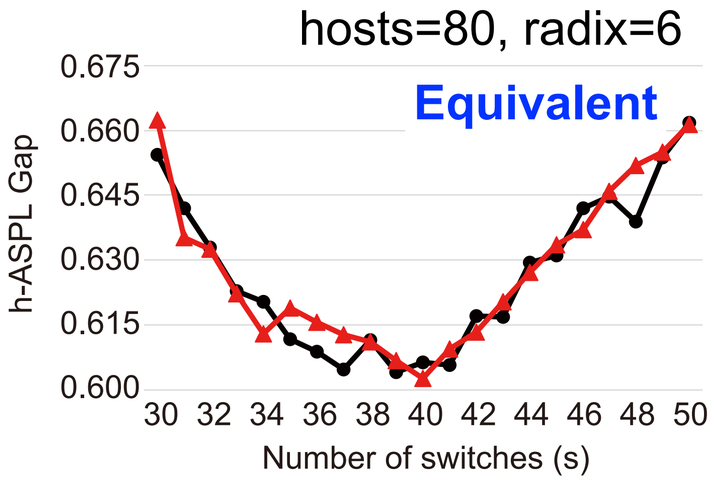


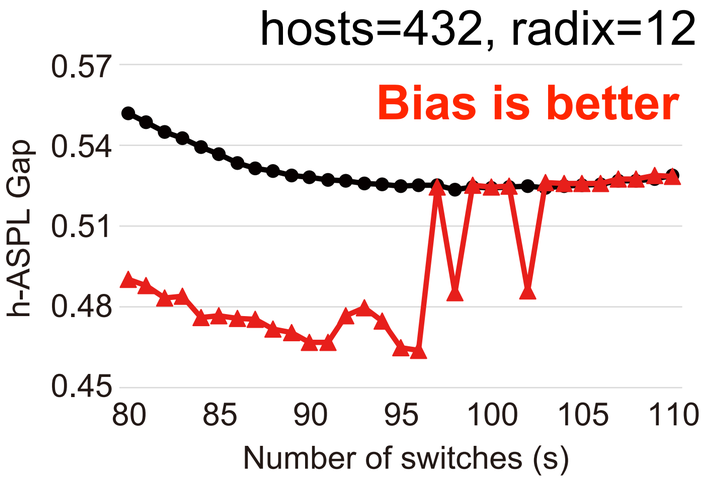
If #switches adjacent to **sa** and **sb** are **A** and **B**, the replacement is performed with the probability of **B**/(**A**+**B**)

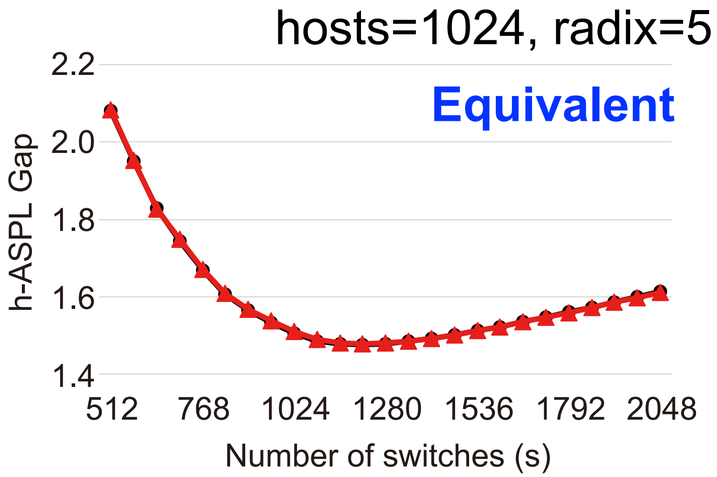
called “Bias Selection”

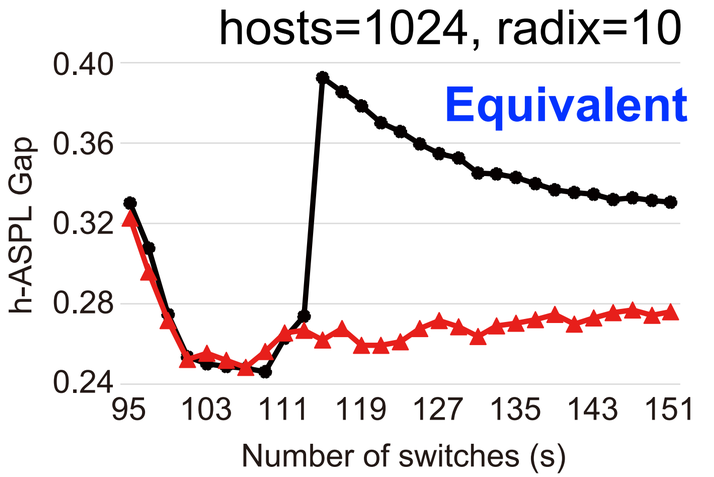
Result with biased host density

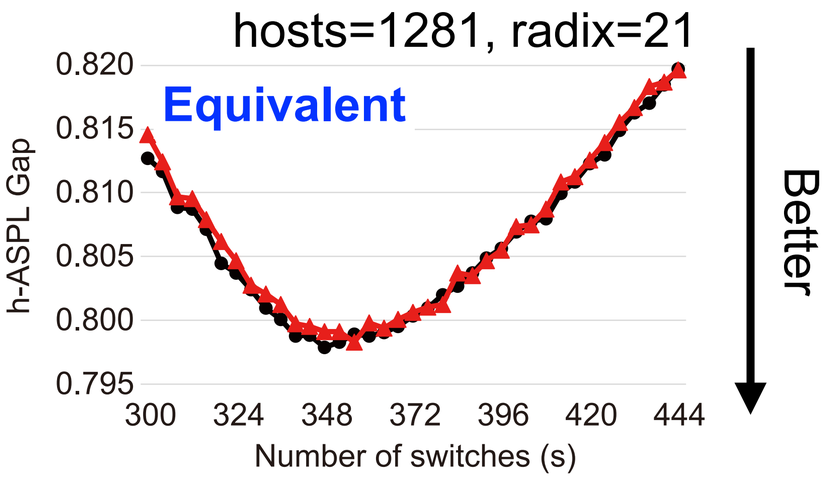




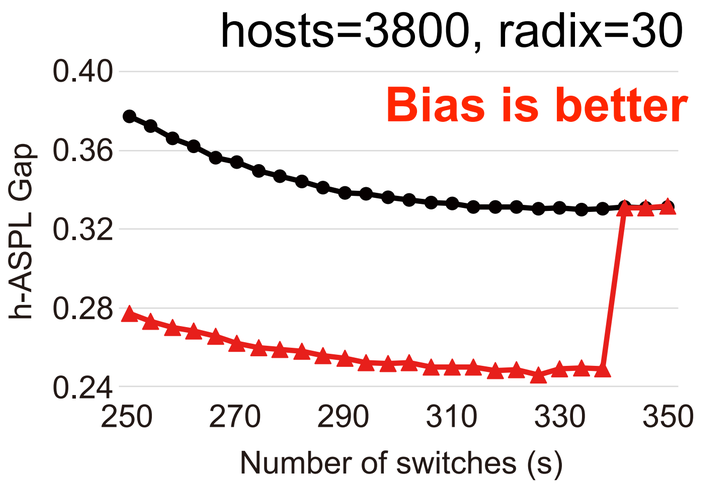


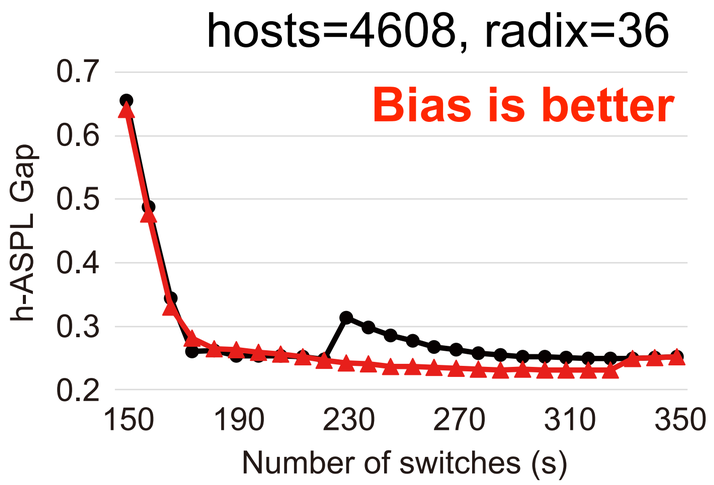


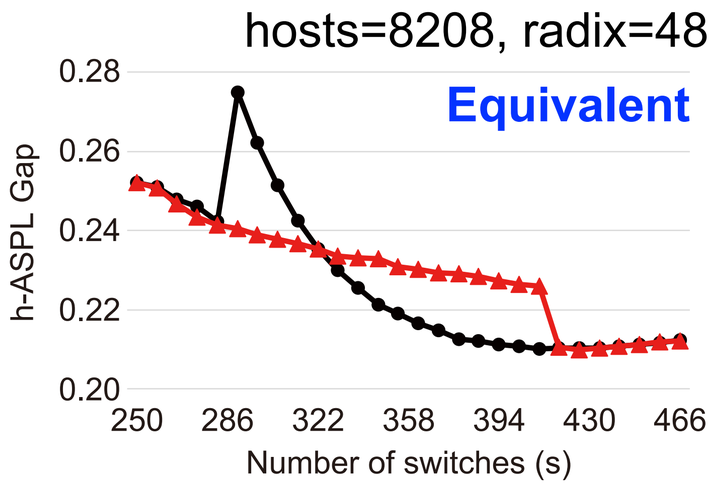


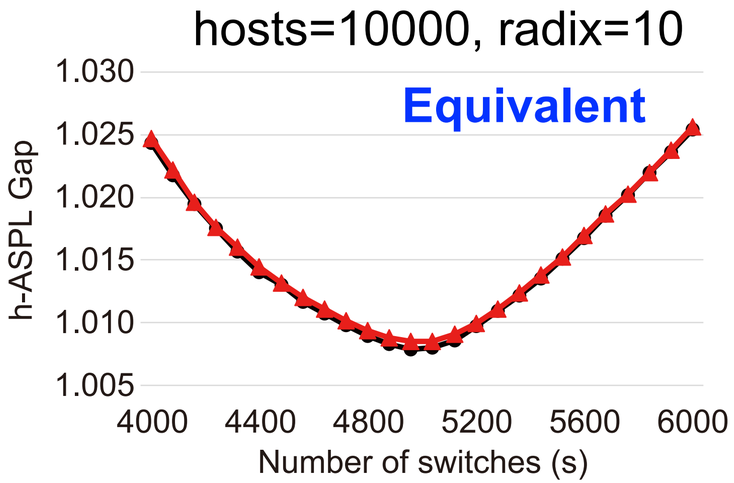


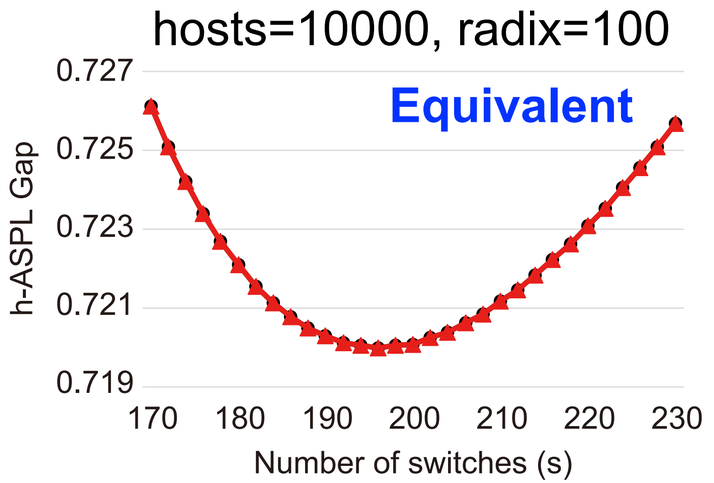
Result with biased host density

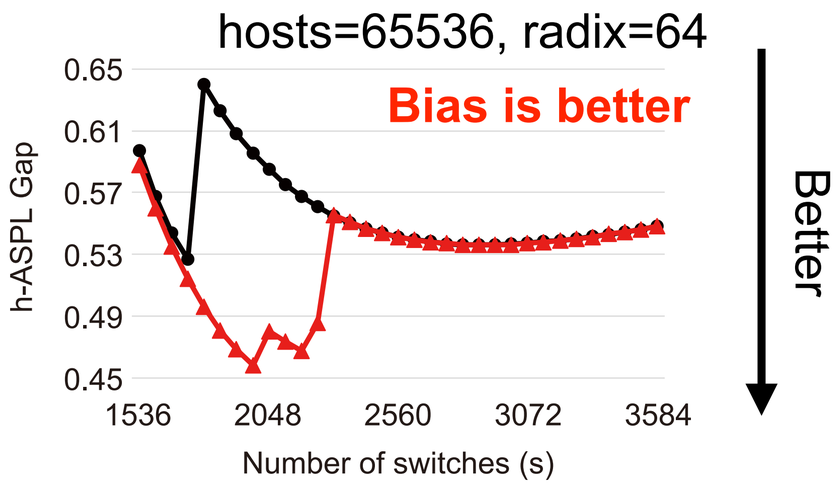






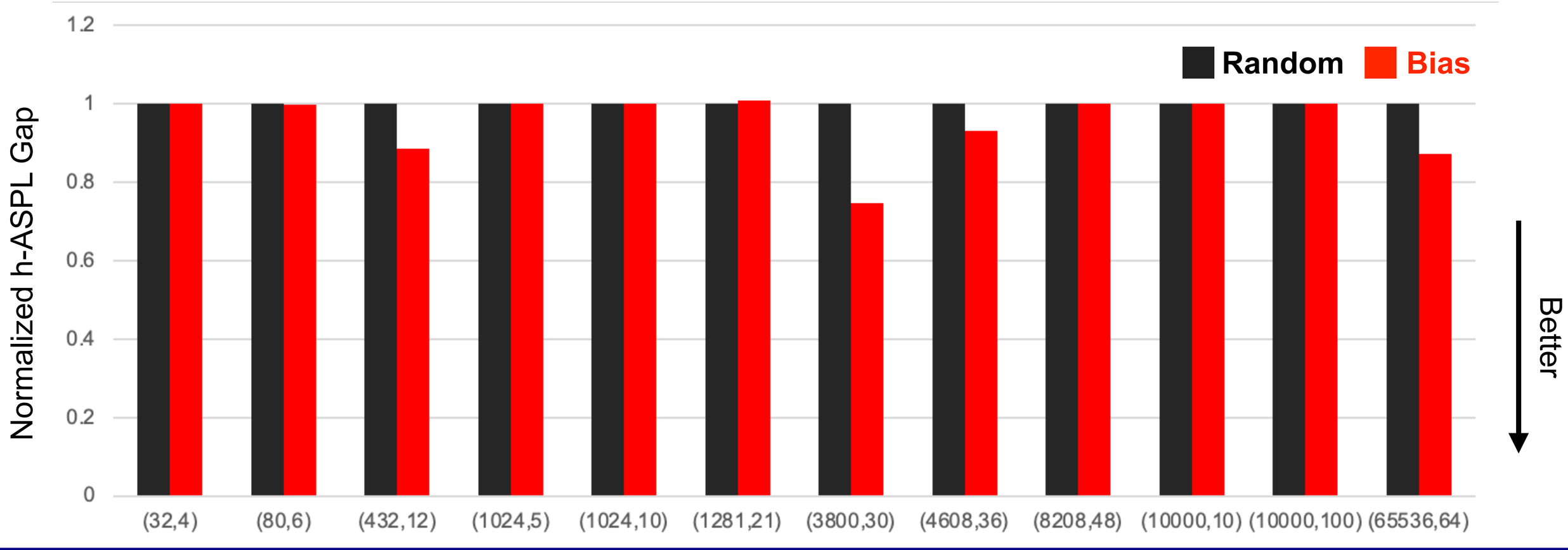






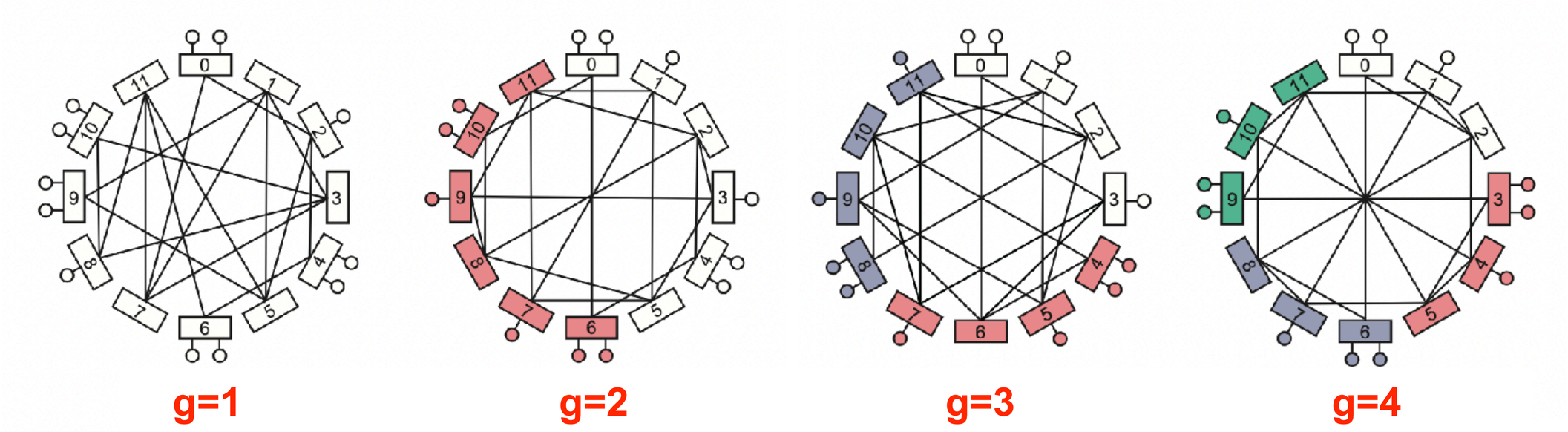
Result with biased host density

- Normalize the best h-ASPL Gaps of **Random** to 1.0
- From this result, it is said that **Bias** has the same or better performance as **Random**



Symmetry

- Examples of the graph symmetry with (hosts=12, radix=4, switches=12)



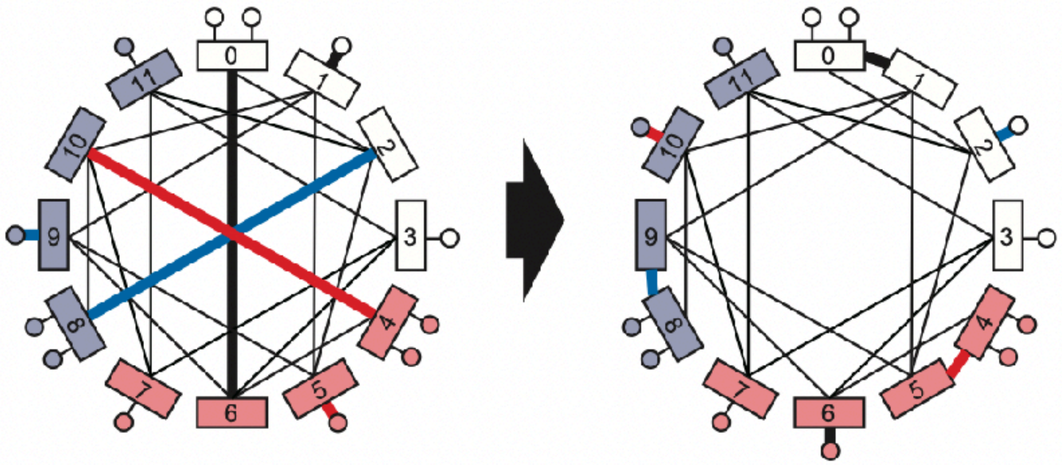
g=1 g=2 g=3 g=4

- The variable g must be a common divisor of #hosts and #switches
- When a graph is viewed as a plane, if it is rotated by 360/g degrees, the connection relationship between the edge and the vertex becomes the same graph

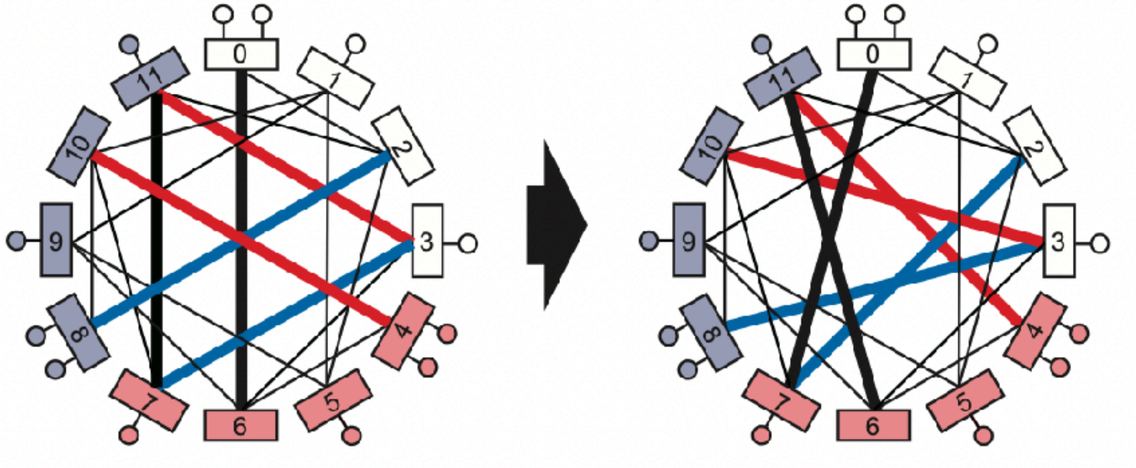
SWING and SWAP with symmetry

- Examples of the graph symmetry with (hosts=12, radix=4, switches=12, g=3)
- **SWING** and **SWAP** are performed while maintaining symmetry
- The combination of multiple edges changes in the same way

- **SWING**

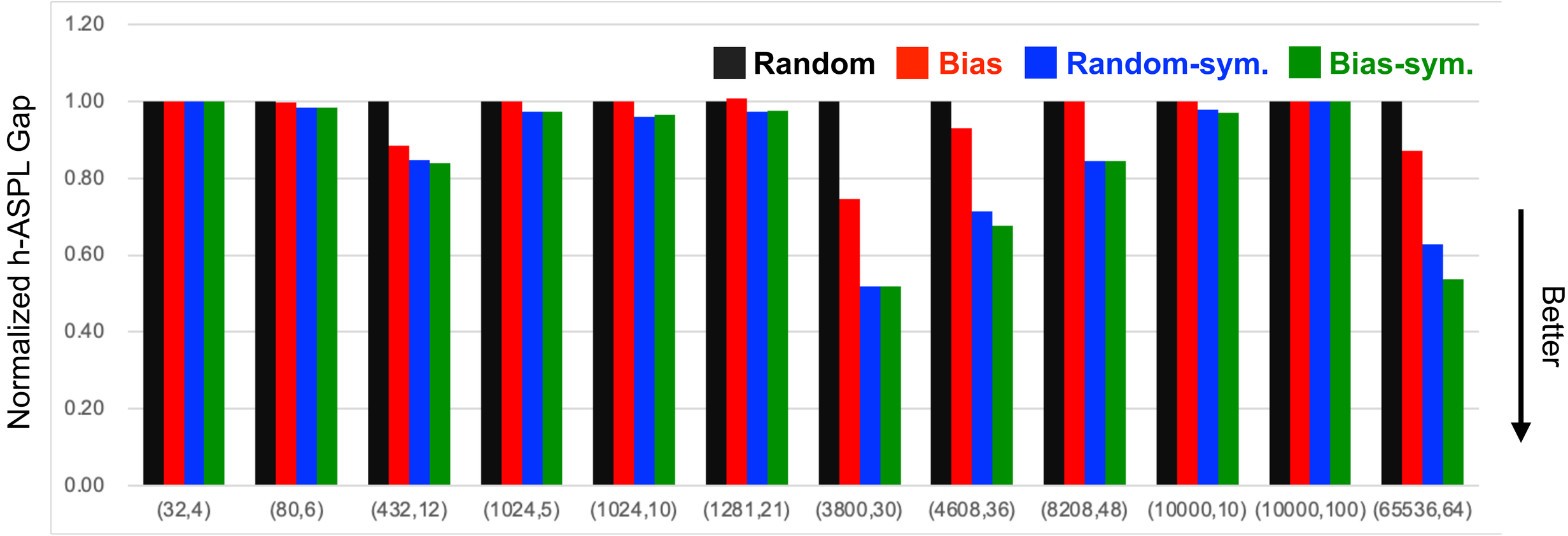


- **SWAP**

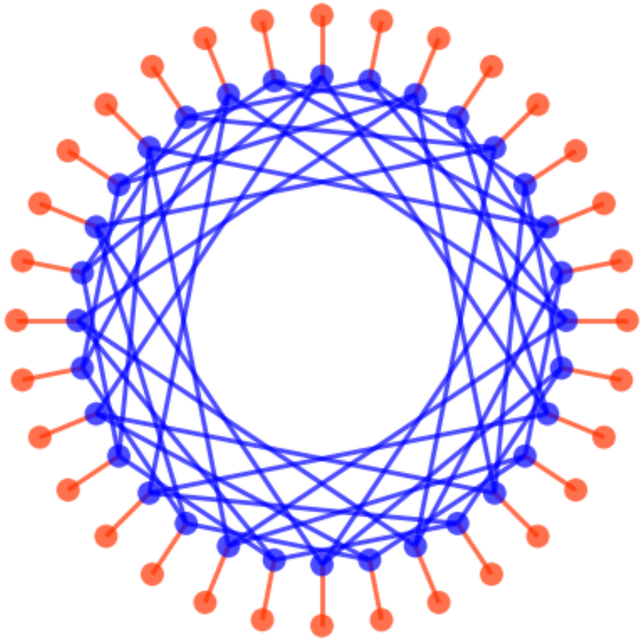


Result with symmetry

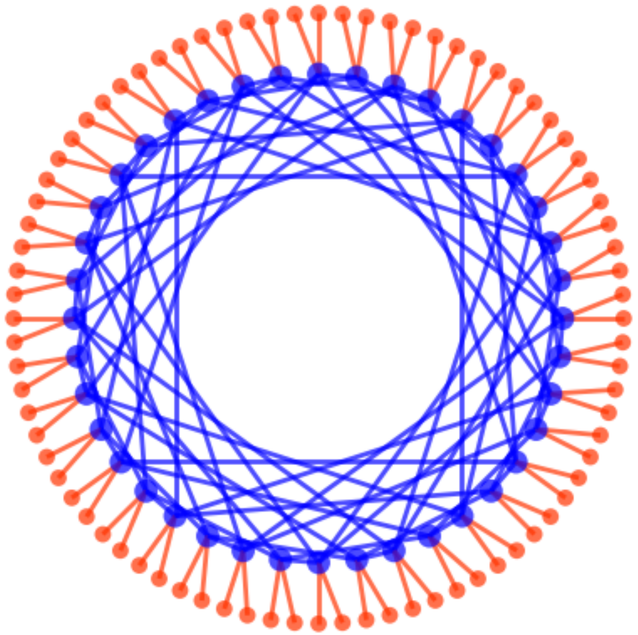
- Normalize the best h-ASPL Gaps of **Random** to 1.0
- From this result, it is said that **Bias-sym.** has the best performance of all



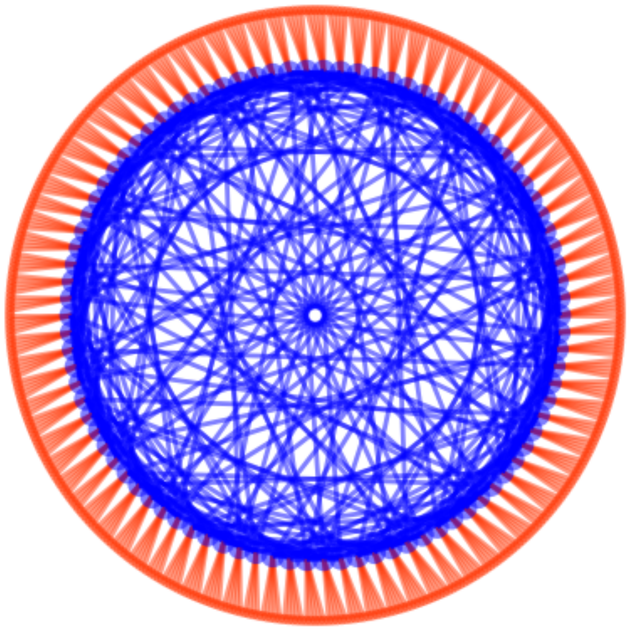
Examples of results



(32, 4)



(80, 6)



(432, 12)

Conclusion

- We develop a graph optimization algorithm for Order/Radix Problem
- The algorithm creates a graph with a small h-ASPL by biasing the host density and giving the graph symmetry
- The calculation speed of h-ASPL is 126.2 times that of the reference implementation

- State-of-the-art future
  - Automatic determination of the number of switches (by **Masaki Tsukamoto**)

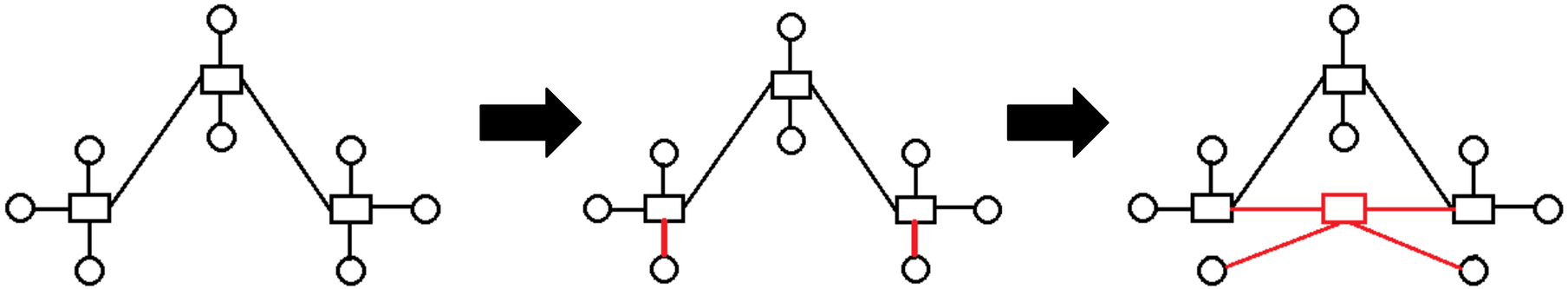
Automatic determination of the number of switches

- To find the optimal number of switches through the search, we introduce the operations to increase or to decrease the number of switches.
- 4 operations to generate neighborhood solutions
  - Swing
  - Swap
  - *Add switch*
  - *Reduce switch*

Add switch

Put one more switch to the graph keeping the constraints about radix

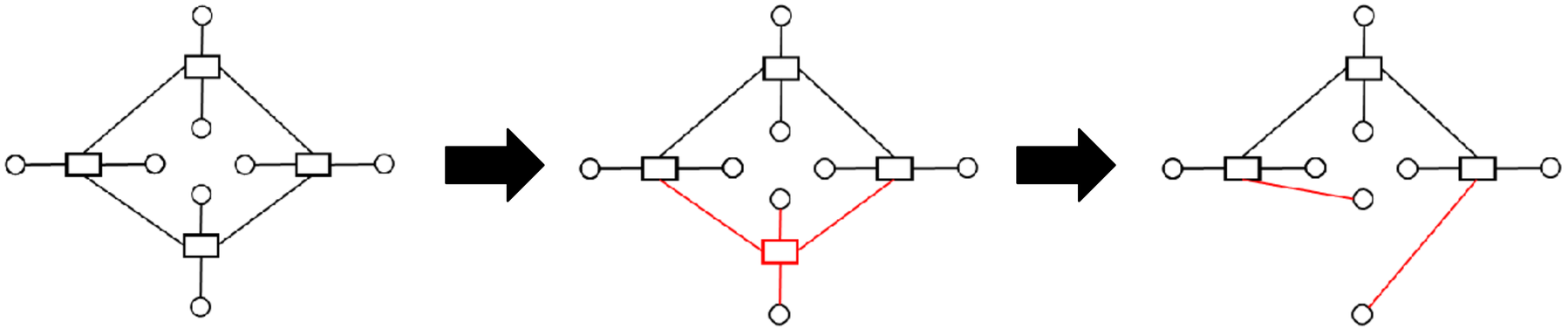
1. Select radix/2 edges
2. Connect the nodes at both ends to the new switch



Reduce switch

Delete one switch with the number of connecting hosts is small.

1. Select a switch where the number of hosts is less than 2/radix.
2. Reconnect the disconnected nodes to other switches.



Results

	original SA (Best)		Adaptive SA	
Instance	ASPL	Switches	ASPL	Switches
(32, 4)	4.93548	32	4.95161	26
(80, 6)	4.45570	40	4.475	35
(128, 24)	2.87845	8	2.88164	8
(432, 12)	4.15816	103	4.3212	91

- The search performance becomes worse compared to normal SA, due to the large search space
- Converges in a small number of switches compared to the optimal solution