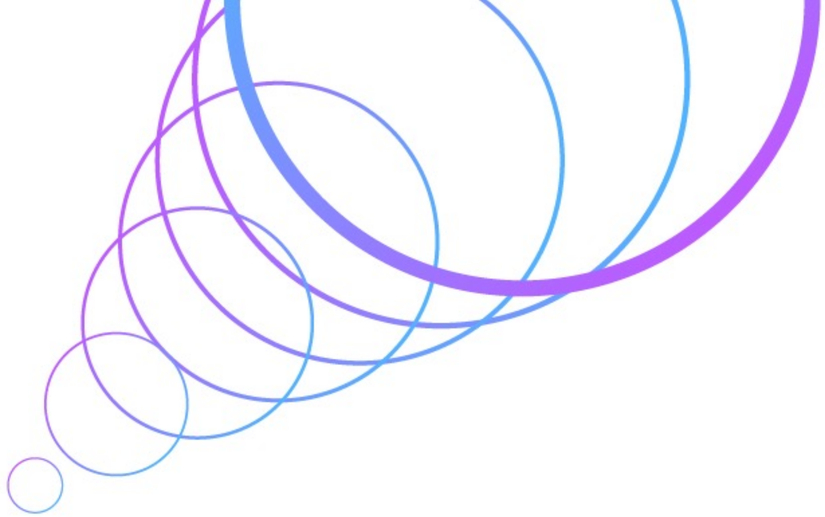


강화학습, 산업의 난제에 도전하다!

ASIC 반도체 설계(Floorplan) 자동화

박진우 Makinarocks

PEOPLE

MakinaRocks

우경민

임지윤

명우식

+

COOPERATORS

CONTENTS

1. ASIC 반도체 설계 문제

1.1 ASIC Design Flow

ASIC (Application-Specific Integrated Circuit)?

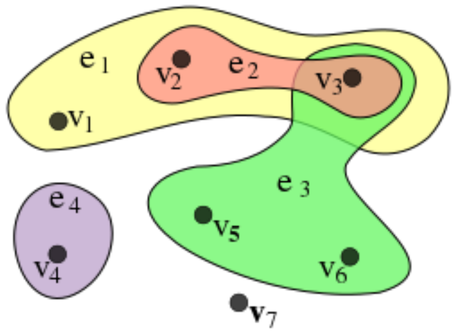
- 특정 용도용 집적 회로
- 범용이 아닌 특정 용도에 맞게 맞춤 제작된 집적 회로
- 주문형 반도체라고도 칭함

1.1 ASIC Design Flow

Netlist

- 반도체의 **논리적 설계도**
- 각 **소자들의 특성과 연결관계**를 정의하고 있는 데이터 (*Hypergraph)
- 소자는 상대적으로 크기가 큰 **Macro**와 크기가 작은 **Standard Cell**로 구분
- Netlist는 몇 백여개의 Macro와 수십만 ~ 수백만 개의 Standard Cell로 구성

* Hypergraph: 하나의 Edge가 복수의 Vertices에 대한 연결관계를 표현할 수 있는 그래프

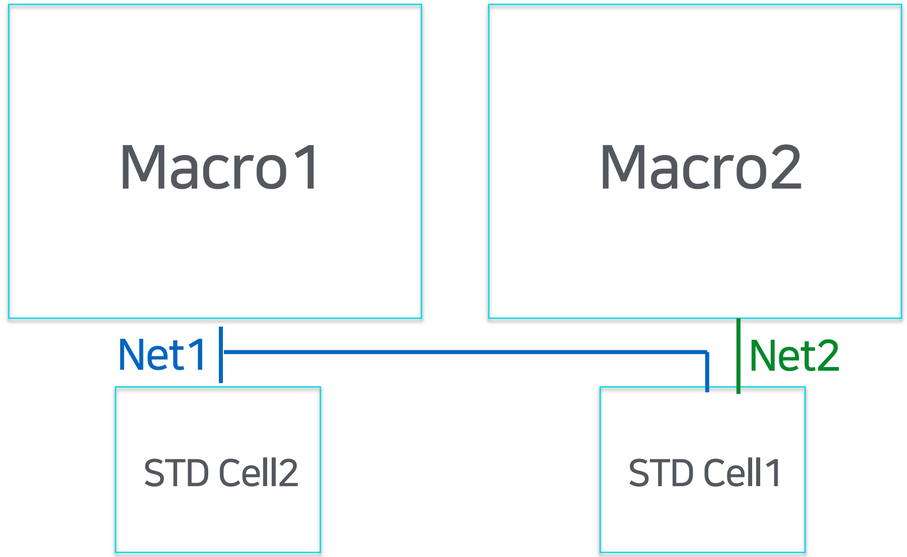


삽화 출처: https://en.wikipedia.org/wiki/Hypergraph

1.1 ASIC Design Flow

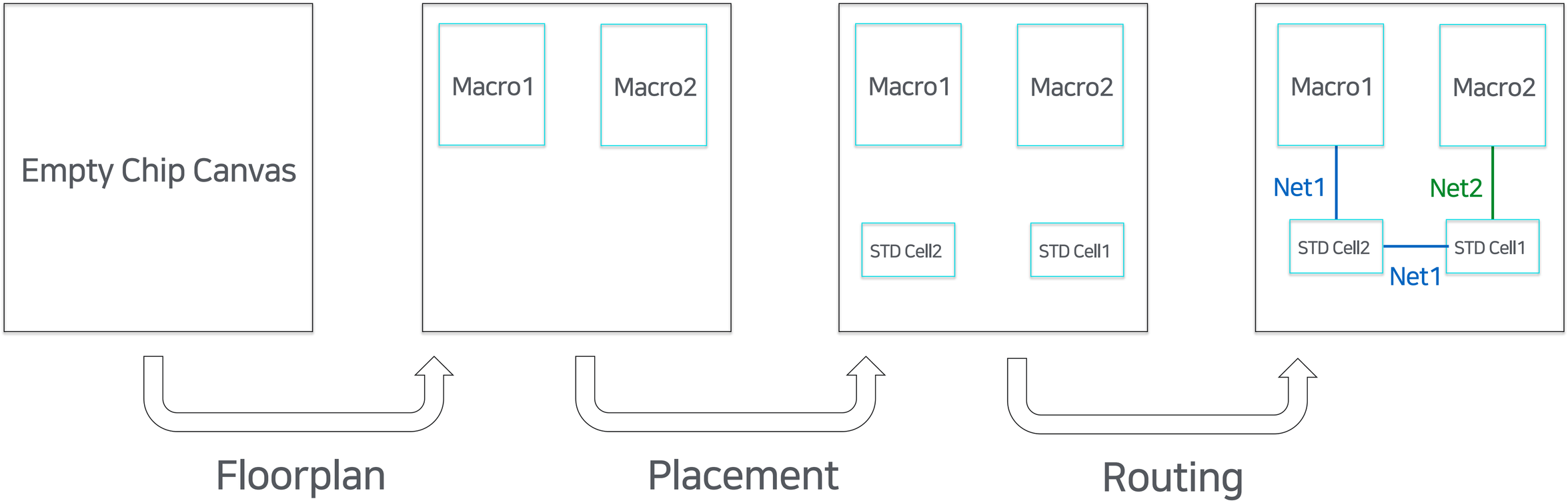
Netlist

	Net1	Net2
Macro1	1	
Macro2		1
STD Cell1	1	1
STD Cell2	1	



1.1 ASIC Design Flow

Physical Design Flow (논리적 설계도 ▶ 물리적 설계도)



1.1 ASIC Design Flow

DEVIEW 2021

좋은 설계란?

- 낮은 전력(Power) 소모
- 높은 성능(Performace) e. g. Frequency
- 작은 영역(Area) 사용

► a.k.a. PPA

1.1 ASIC Design Flow

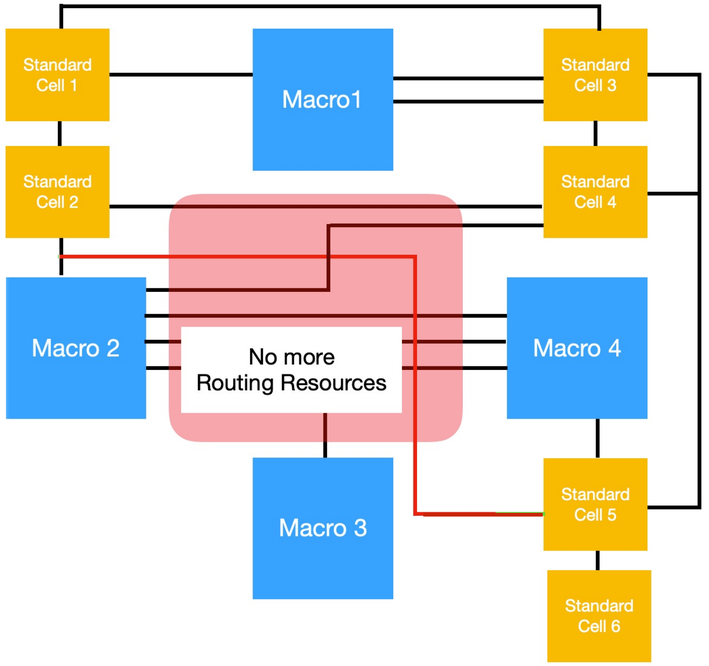
PPA를 최적화 하려면?

- 소자들을 연결하는 Wire의 길이가 짧아지면 전기신호의 도달이 빨라짐
- 전기 신호의 도달이 빨라지면 성능이 올라감
- 짧은 시간 안에 전기 신호를 전달함으로써 전력 사용이 줄어듦
- 전반적인 Wire 사용이 줄어들면 집적도가 올라가며 차지하는 영역이 작아짐

1.1 ASIC Design Flow

모든 소자들을 가깝게 배치하면 될까?

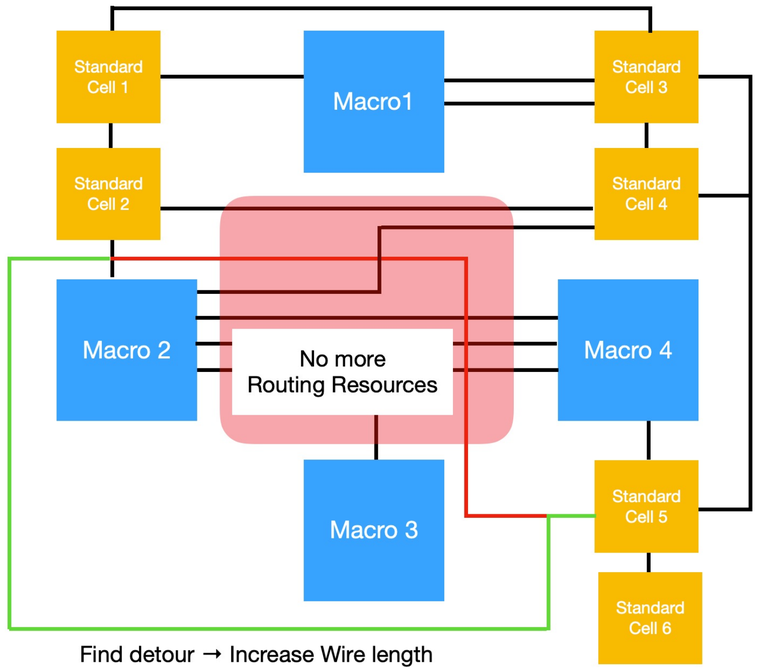
- NO!
- Chip Canvas의 영역마다 Wire에 할당 가능한 **자원(Routing Resource)이 한정적**
- Routing Resource를 초과하면 결국 Wire가 우회하여 연결됨



1.1 ASIC Design Flow

모든 소자들을 가깝게 배치하면 될까?

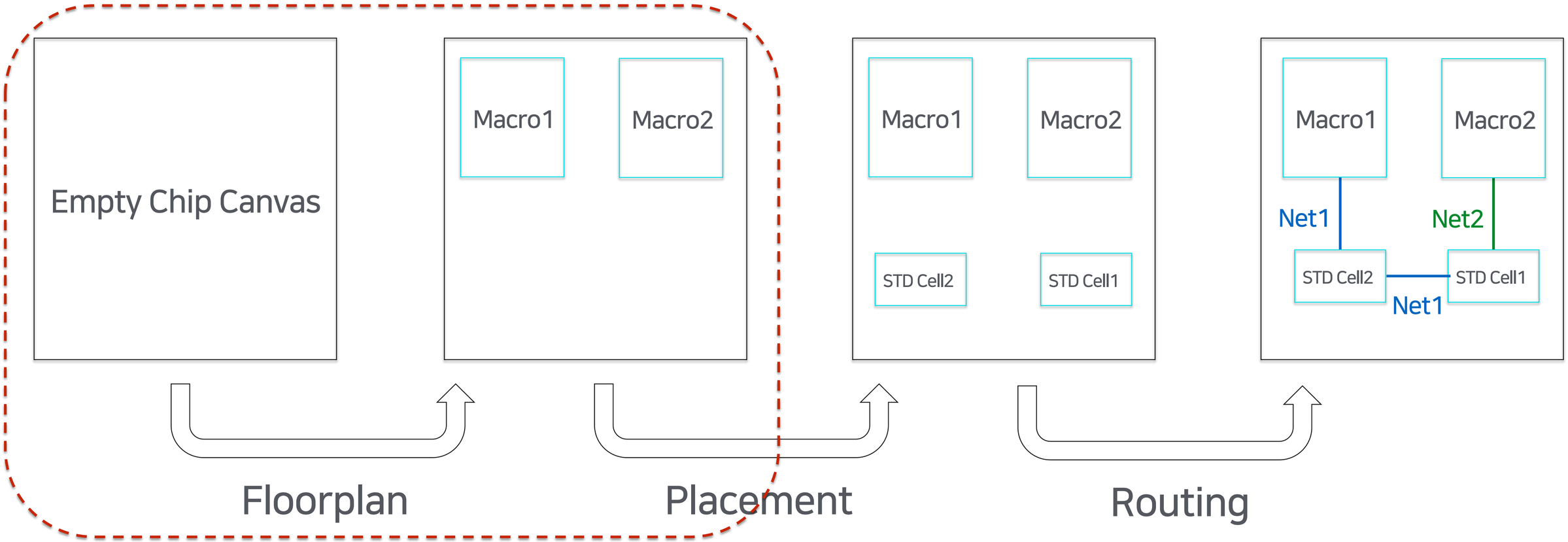
- NO!
- Chip Canvas의 영역마다 Wire에 할당 가능한 **자원(Routing Resource)이 한정적**
- Routing Resource를 초과하면 결국 Wire가 우회하여 연결됨



1.1 ASIC Design Flow

DEVIEW 2021

Physical Design Flow (논리적 설계도 ▶ 물리적 설계도)

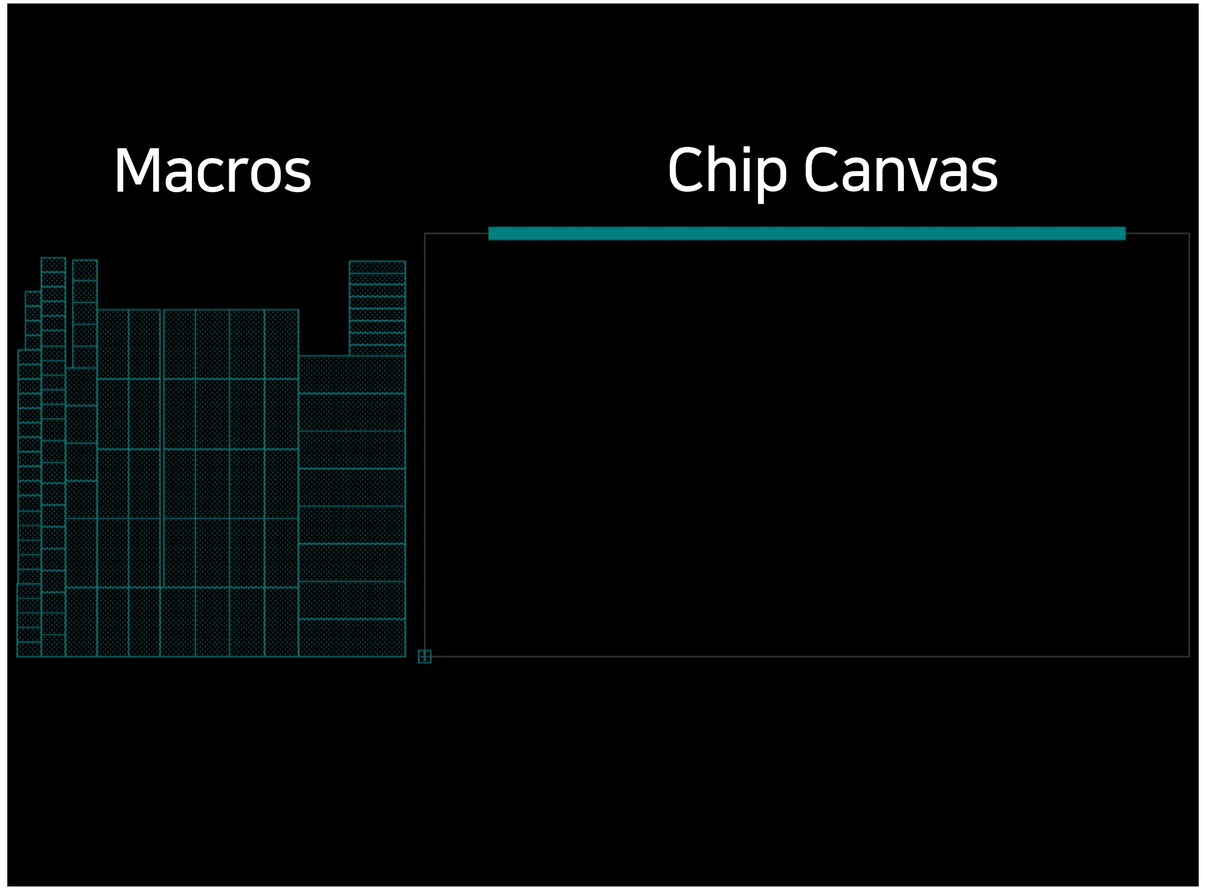


▶ 좋은 설계는 Floorplan에서부터!

1.2 AS-IS

DEVIEW 2021

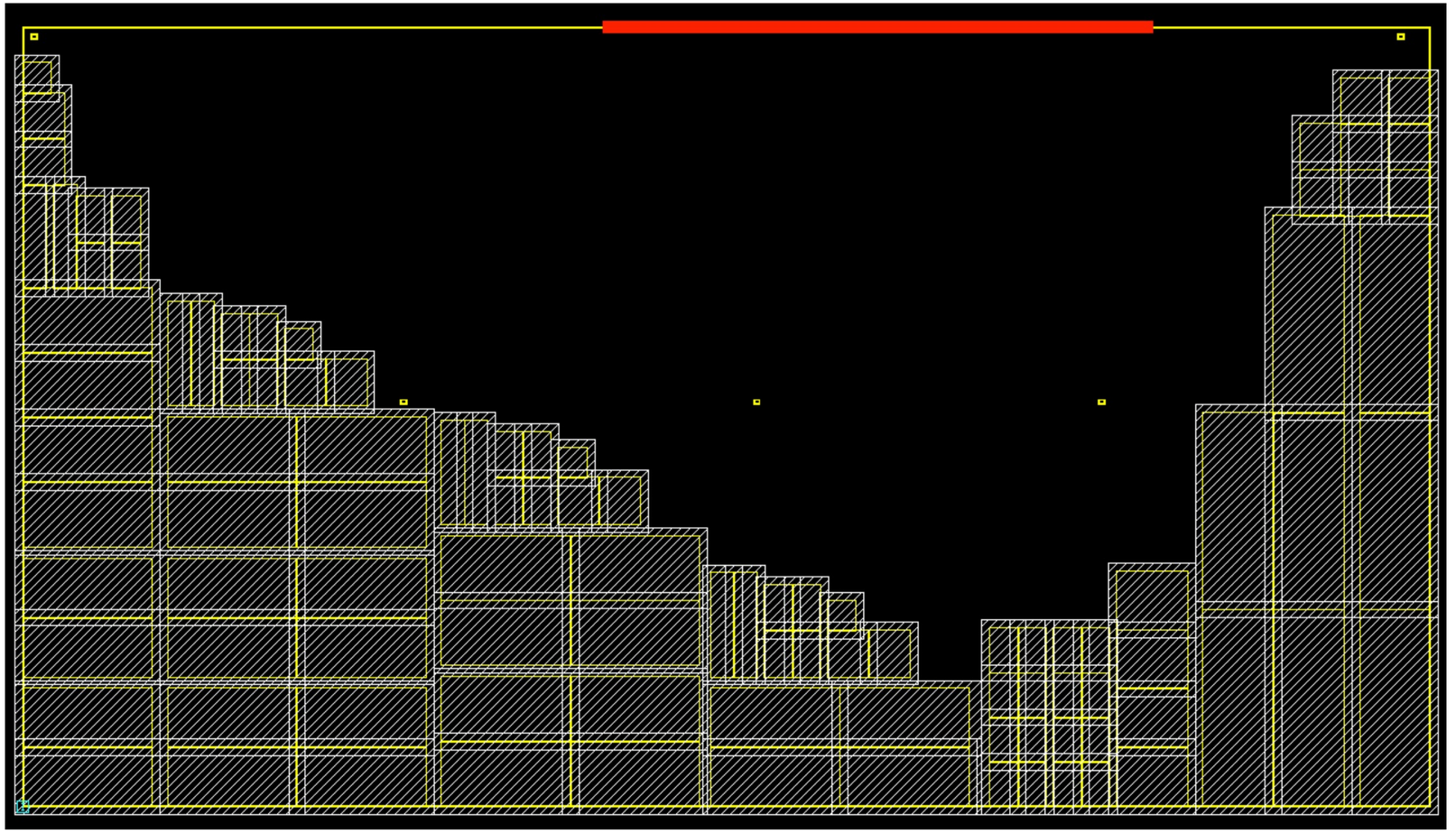
Example (Synopsys ICC2): Initial State



1.2 AS-IS

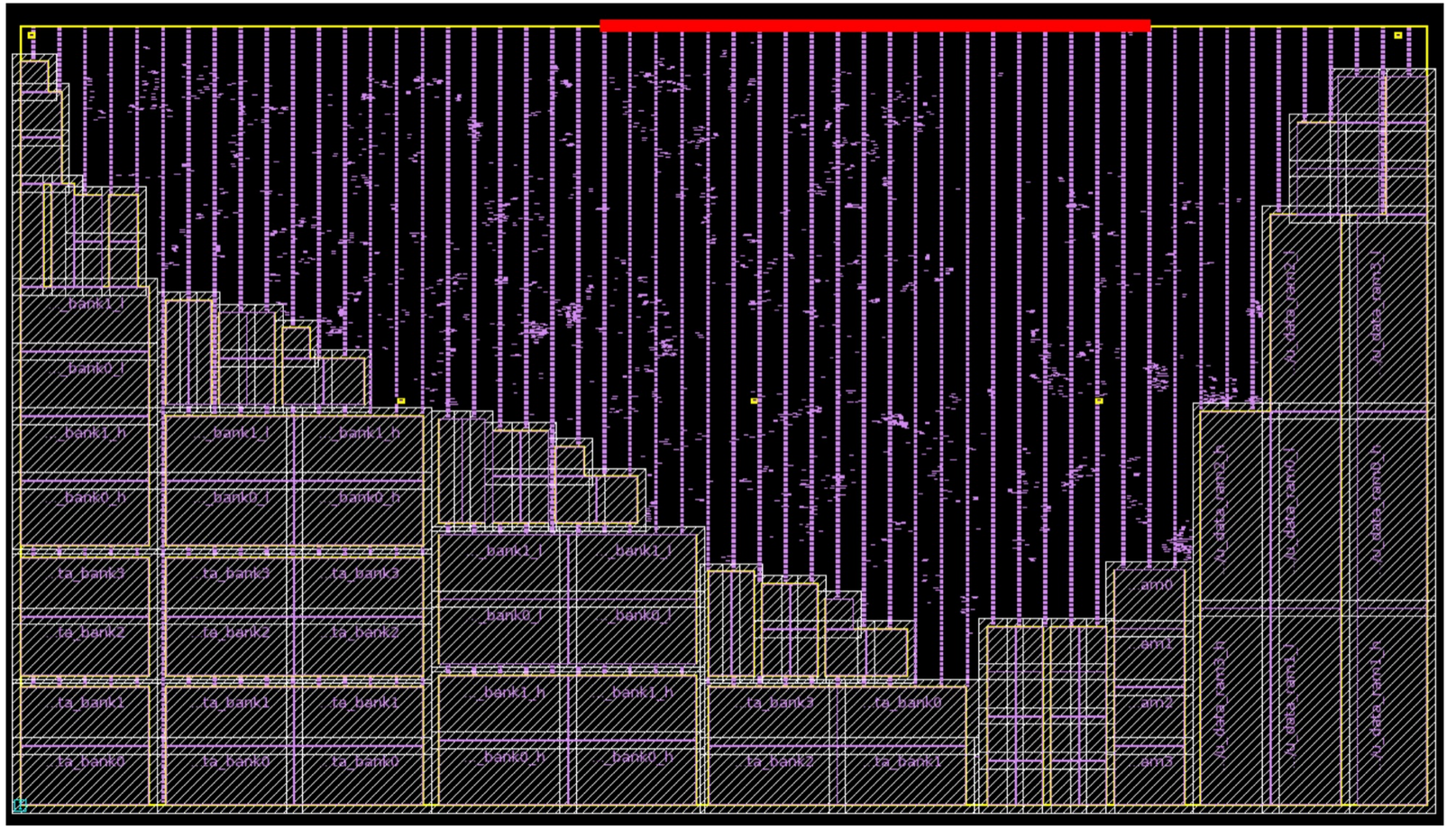
DEVIEW 2021

Example (Synopsys ICC2): After Floorplan



1.2 AS-IS

Example (Synopsys ICC2): After Placement



1.2 AS-IS

DEVIEW 2021

However,

- Floorplan은 주로 엔지니어의 경험 기반으로 수행
- 엔지니어의 숙련도에 따라 업무 수행속도가 천차만별
- 수십 ~ 수백만 Cell의 연결관계를 염두하며 효율적인 배치를 하기 어려움
- Floorplan의 뒤이은 설계 과정은 경우에 따라 며칠 이상의 시간이 소요됨
- 최종 설계 결과의 품질이 좋지 않을 때, Floorplan 과정부터 다시 수행

1.3 IMPACT

반도체 산업에의 Impact

- 반도체 설계 시간 단축
- Designer 숙련도 차이에 따른 설계 편차 감축

타 산업에의 Impact

- 조합 최적화 문제 해결에 활용 가능한 공통 기술 확보
- 물류 거점 최적화 등 다양한 산업 문제에 강화학습 적용 기반 마련

2. 기계학습을 이용한 Floorplan 자동화 (Google)

2.1 Overview

2021년 6월, NATURE에 게재

- Google의 TPU 설계에 실제 적용

Article

A graph placement methodology for fast chip design

https://doi.org/10.1038/s41586-021-03544-w

Received: 3 November 2020

Accepted: 13 April 2021

Published online: 9 June 2021

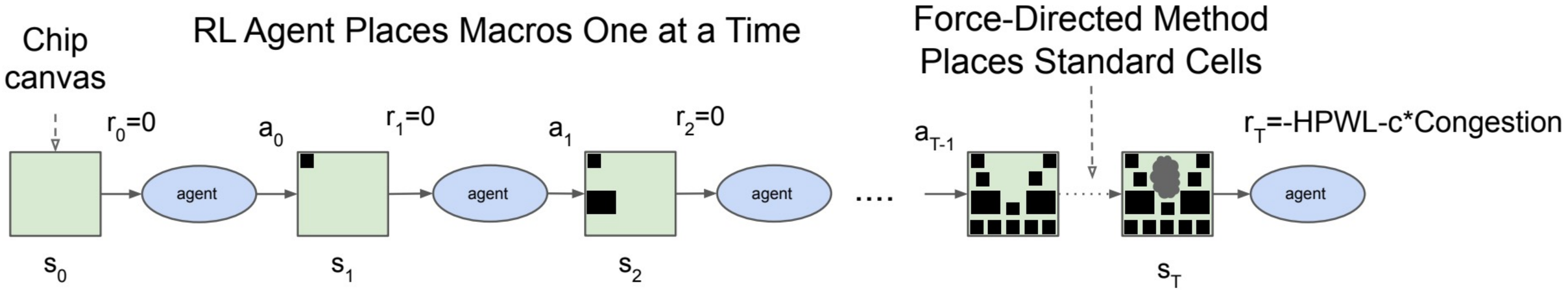
Azalia Mirhoseini[1,4]✉, Anna Goldie[1,3,4]✉, Mustafa Yazgan[2], Joe Wenjie Jiang[1], Ebrahim Songhori[1], Shen Wang[1], Young-Joon Lee[2], Eric Johnson[1], Omkar Pathak[2], Azade Nazi[1], Jiwoo Pak[2], Andy Tong[2], Kavya Srinivasa[2], William Hang[3], Emre Tuncer[2], Quoc V. Le[1], James Laudon[1], Richard Ho[2], Roger Carpenter[2] & Jeff Dean[1]

A. Mirhoseini et al. A graph placement methodology for fast chip design, Nature 2021

2.1 Overview

RL Process

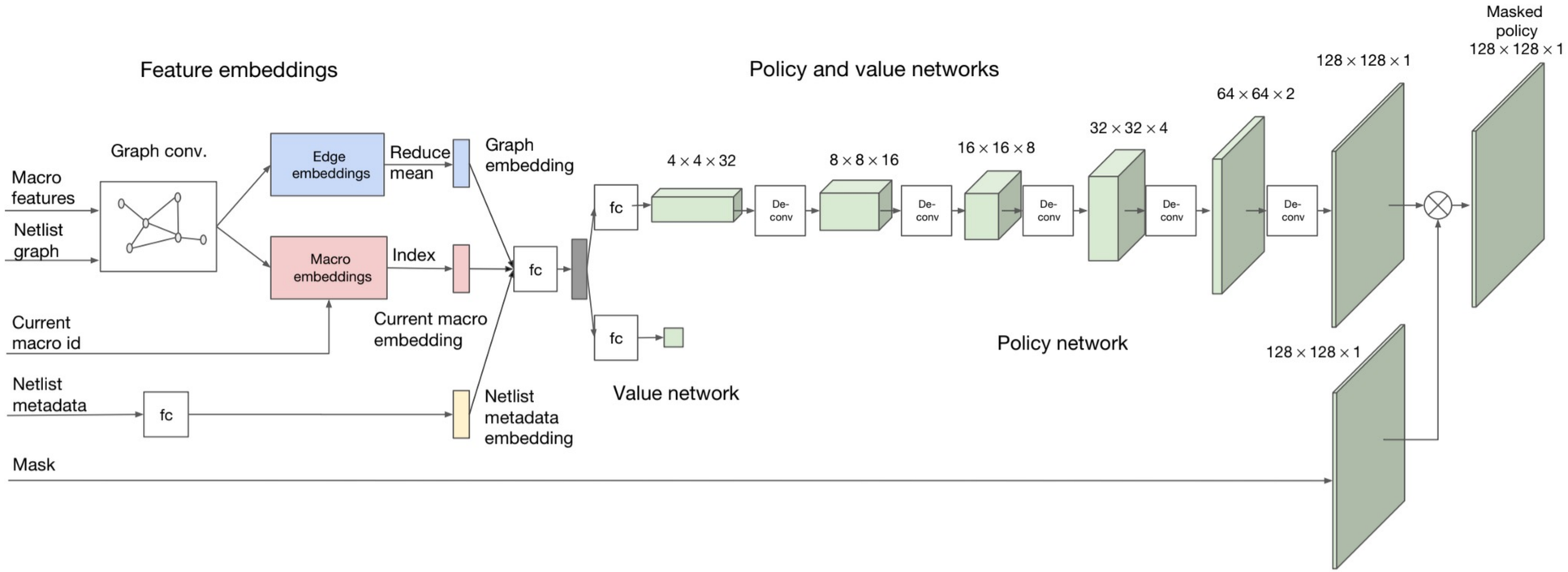
- 강화학습 에이전트로 Macro Cell을 배치
- Macro Cell 배치 완료 후 Force-Directed Method로 Standard Cell 배치
- Wire Length와 Routing Congestion의 추정값을 이용해 Reward를 계산



2.1 Overview

Network Architectures

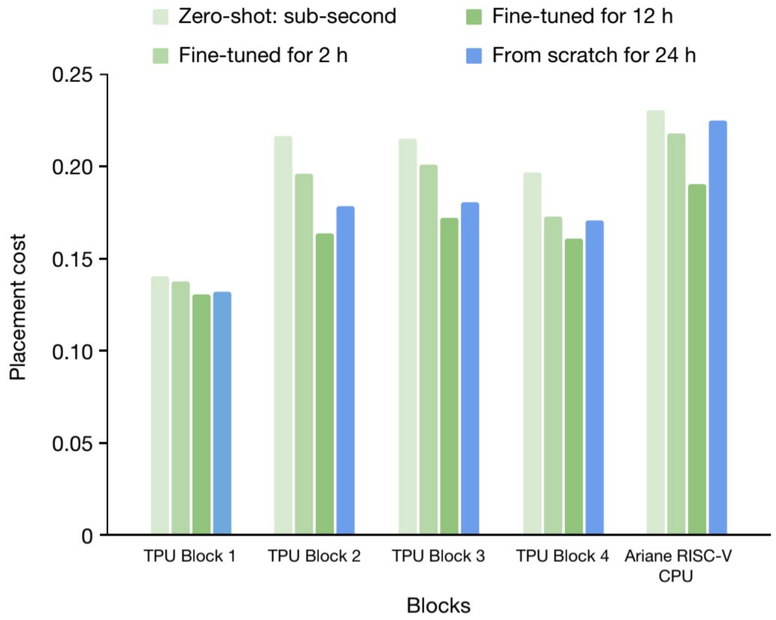
- Macro Feature(x, y, w, h, type)와 Netlist Graph를 GNN에 입력
- Netlist Metadata와 Macro / Graph Embedding을 입력으로 Chip Canvas의 각 배치 영역에 대한 확률분포를 출력

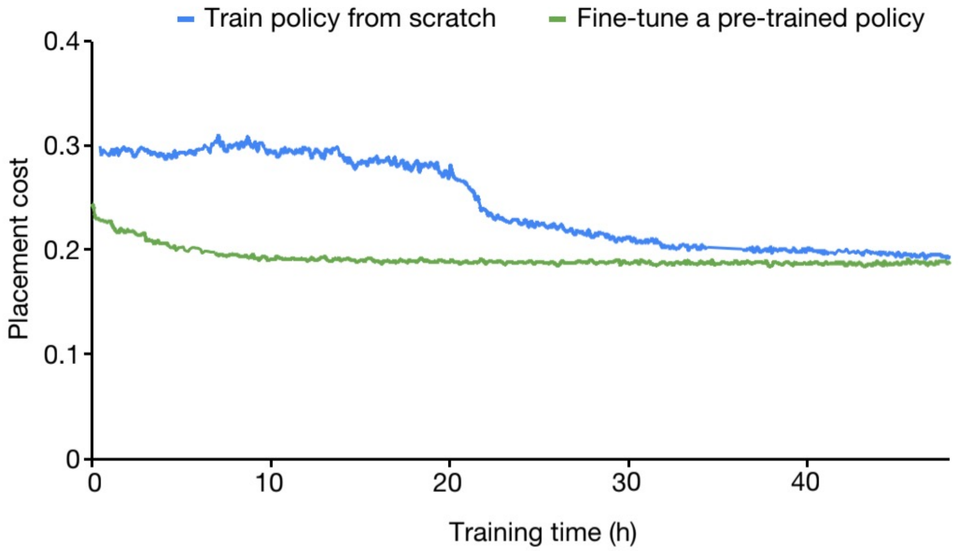


2.1 Overview

Training Process

- 최대 20개 Block(Netlist)으로 Pretraining (입력: 배치결과, 출력: Reward)
- 최종 학습 대상의 Netlist에 대해 Fine-Tuning
- Pretraining을 했을 때, 훨씬 더 빠르게 학습되는 경향성이 관측됨





2.2 Details

Standard Cell Clustering

- hMETIS 기법을 이용하여 Netlist의 Standard Cell을 Clustering
- hMETIS는 Multilevel의 Hypergraph Bisection Clustering 방법
- Standard Cell Clustering을 통해 GNN, FDM의 계산량을 완화

G. Karypis et al. Multilevel Hypergraph Partitioning, DAC 1998

2.2 Details

Force-Directed Method (FDM)

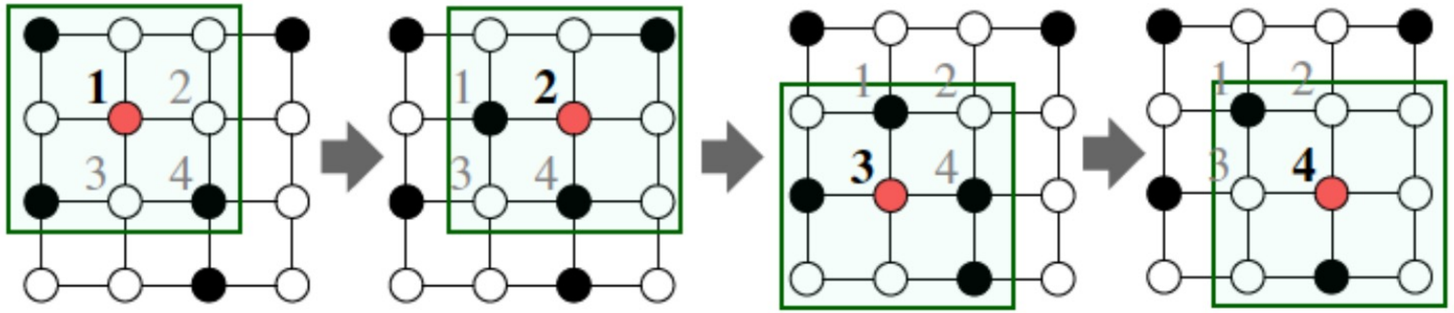
- Standard Cell을 배치하는 전통적인 기법
- 각각의 노드 사이에 **Weight x Distance의 장력**이 작용한다고 가정
  (노드 사이의 연결성이 높을수록 가까이에 위치)
- Standard Cell을 FDM으로 배치함으로써 RL의 Sparse Reward 문제를 완화

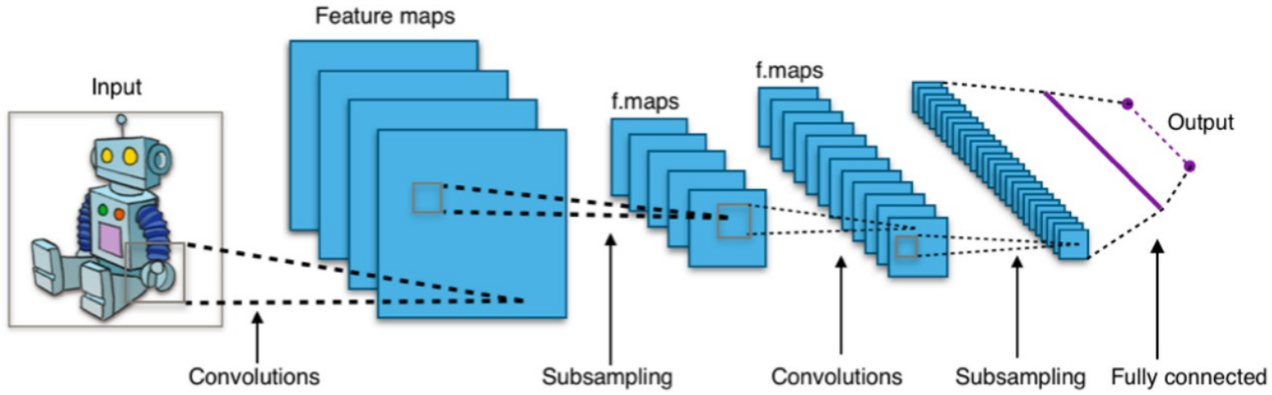
K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques, ACM Computing Surveys 1991

2.2 Details

Graph Neural Network (Basics)

CNN on an image:





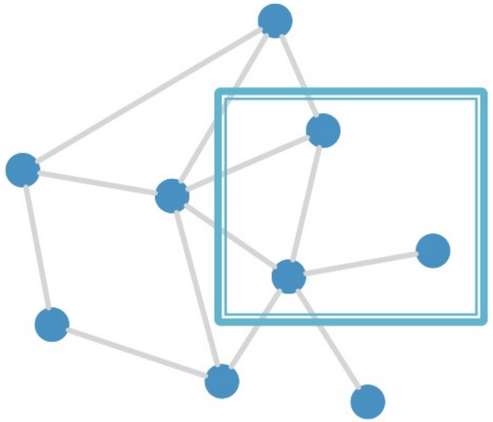
Jure Leskovec. Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu

2.2 Details

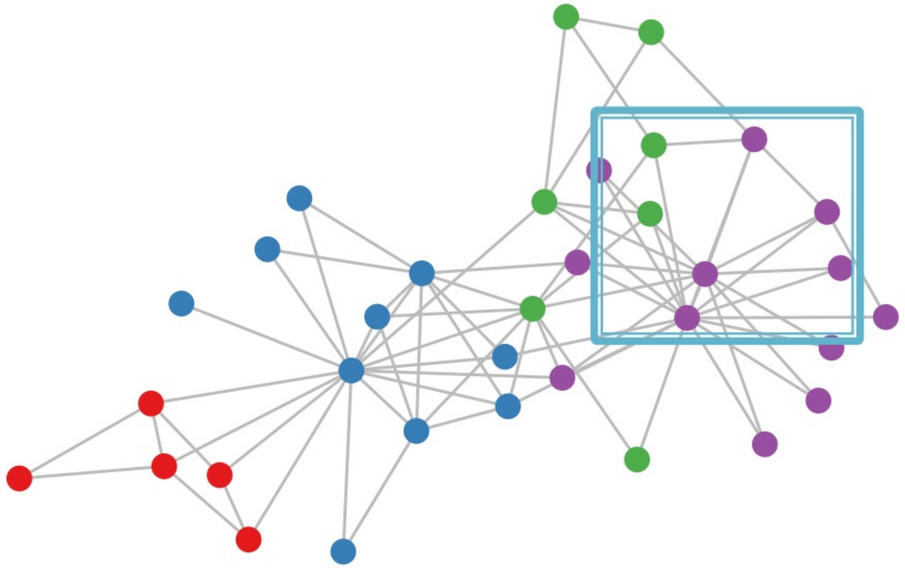
DEVIEW 2021

Graph Neural Network (Basics)

But our graphs look like this:



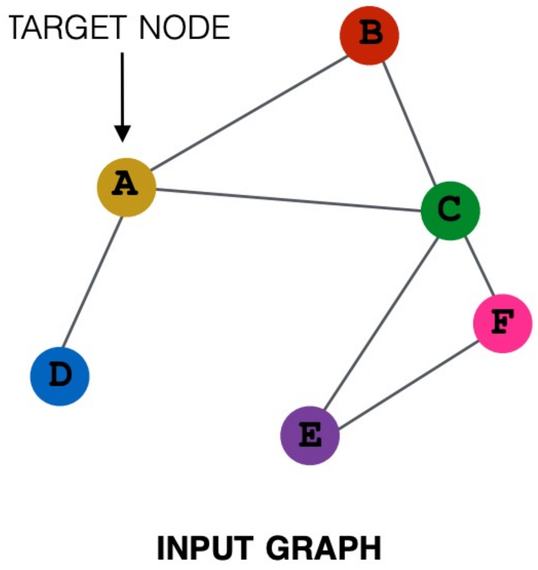
or this:



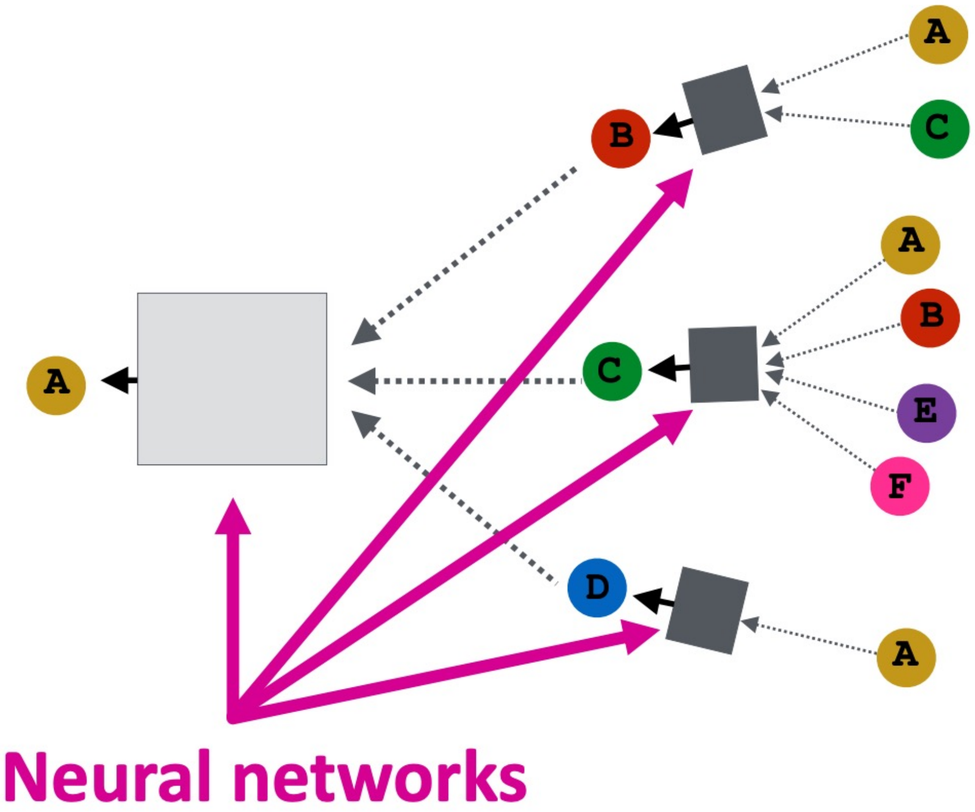
Jure Leskovec. Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu

2.2 Details

Graph Neural Network (Basics)



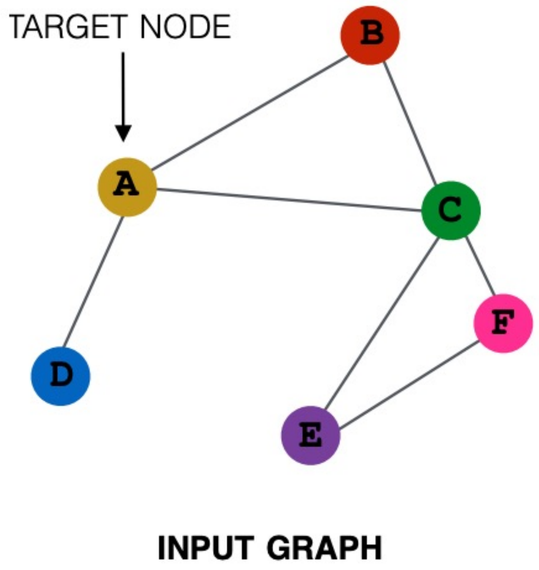
INPUT GRAPH

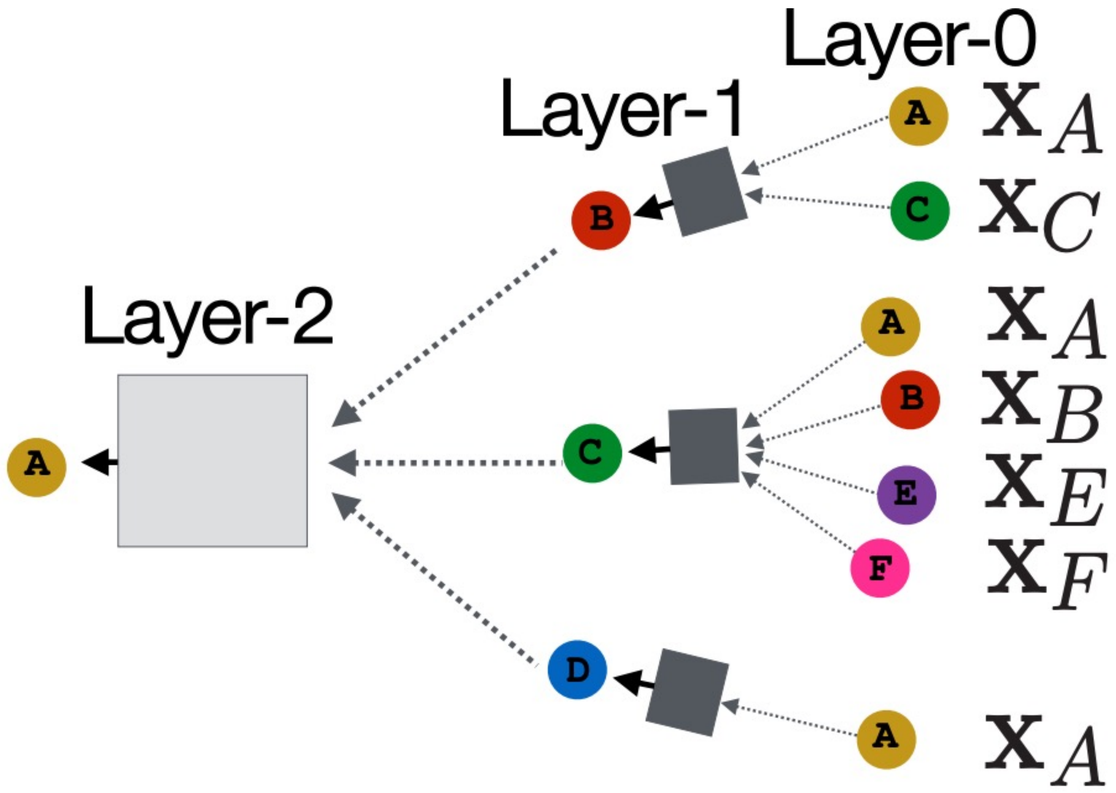


Jure Leskovec. Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu

2.2 Details

Graph Neural Network (Basics)

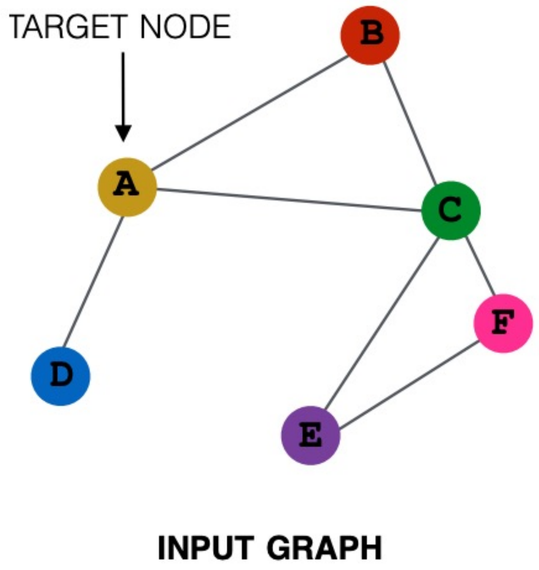


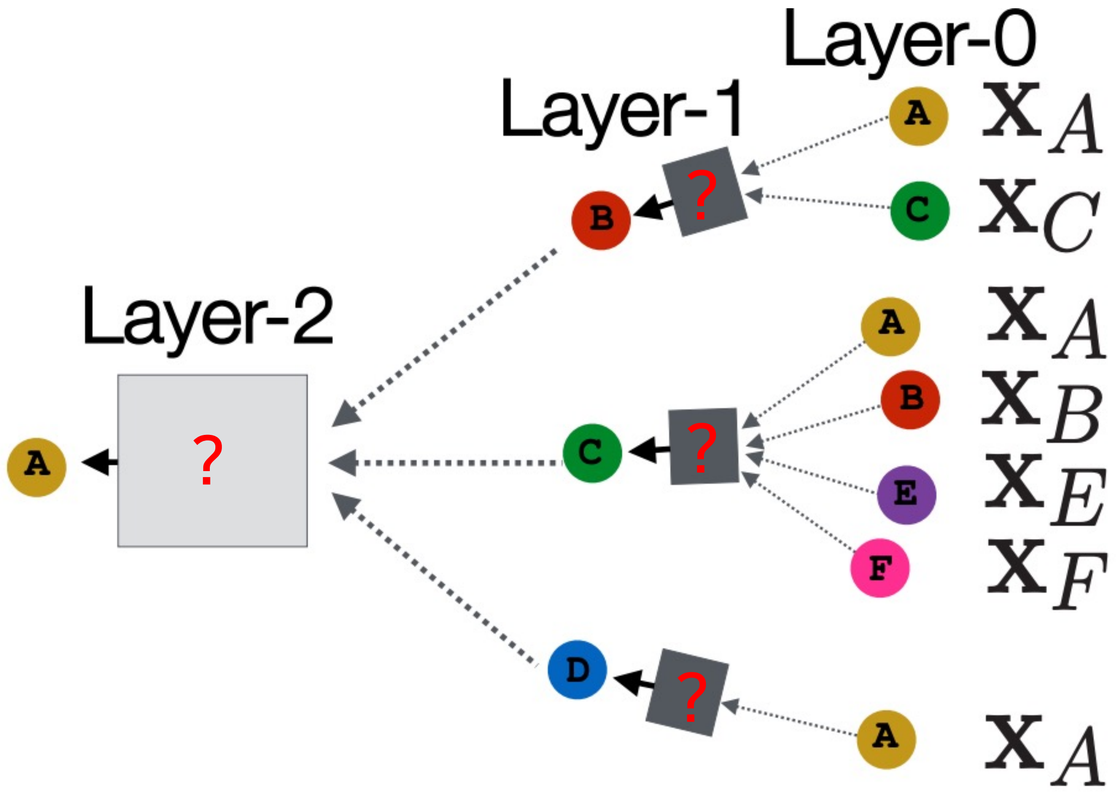


Jure Leskovec. Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu

2.2 Details

Graph Neural Network (Basics)

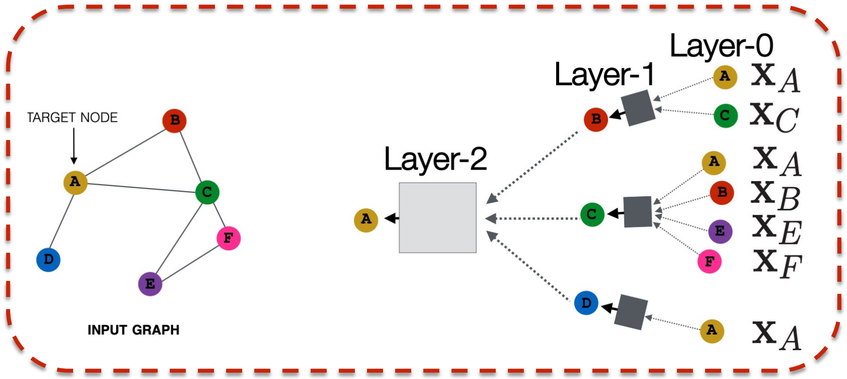




Jure Leskovec. Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu

2.2 Details

Graph Neural Network (Basics)



Initial 0-th layer embeddings are equal to node features

$$\mathrm{h}_v^0 = \mathrm{x}_v$$

embedding of v at layer l

$$\mathrm{h}_v^{(l+1)} = \sigma\left(\mathrm{W}_l \sum_{u \in \mathrm{N}(v)} \frac{\mathrm{h}_u^{(l)}}{|\mathrm{N}(v)|} + \mathrm{B}_l \mathrm{h}_v^{(l)}\right), \forall l \in \{0, \ldots, L-1\}$$

$$\mathrm{z}_v = \mathrm{h}_v^{(L)}$$

Embedding after L layers of neighborhood aggregation

Non-linearity (e.g., ReLU)

Average of neighbor's previous layer embeddings

Total number of layers

Jure Leskovec. Stanford CS224W: Machine Learning with Graphs, http://cs224w.stanford.edu

2.2 Details

Graph Neural Network

- Node 뿐만 아니라 Edge에 대한 Embedding을 학습하는 방식의 **Implicit Graph Neural Network 구조**를 제안 (Edge-GCN)

$$\textbf{While not converged do}$$

$$\textbf{Update edge: } e_{ij} = fc_1(\text{concat}[fc_0(v_i)|fc_0(v_j)|w_{ij}^e])$$

$$\textbf{Update node: } v_i = \text{mean}_{j \in N(v_i)}(e_{ij})$$

$$\textbf{end}$$

F. Gu et al. Implicit Graph Neural Networks, NeurIPS 2020

2.2 Details

Graph Neural Network

- GCN과 비교했을 때 훨씬 좋은 학습 성능을 보임
- 네트워크 구조가 비명시적(Implicit)인 특성에 의한 Generalization 효과로 추측

2.2 Details

Reward Function

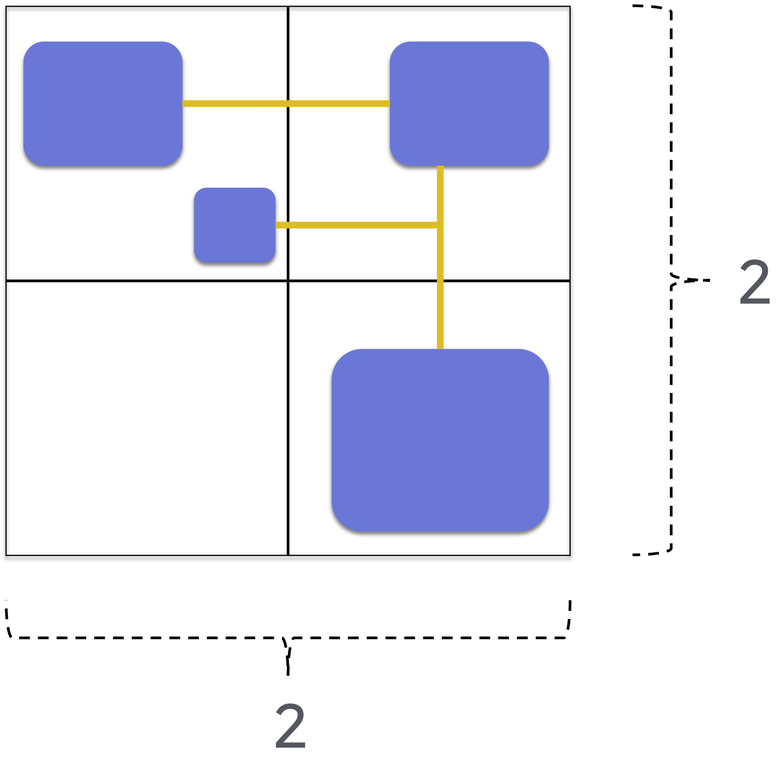
- Wirelength, Congestion, Density의 추정값에 대한 가중합산으로 정의
  (p: placement, g: netlist)

$$R_{p,g} = -Wirelength(p,g) - \lambda Congestion(p,g) - \gamma Density(p,g).$$

2.2 Details

Reward Function: Wirelength (HPWL)

- Net이 배치된 영역 둘레의 반으로 Wirelength를 근사
  (HPWL: Half Perimeter Wire Length)



$$\mathrm{HPWL}(i) = (\max_{b \in i}\{x_b\} - \min_{b \in i}\{x_b\} + 1) + (\max_{b \in i}\{y_b\} - \min_{b \in i}\{y_b\} + 1).$$

2.2 Details

Reward Function: Congestion

- 모든 Grid Cell에 대한 Vertical / Horizontal Routing Allocation을 관측
- 가장 높은 10%의 Congestion 값의 평균치를 Reward에 반영
- Congestion 값을 Smoothing 하기 위해 5 x 1의 Convolutional Filter를 사용
- 정확한 Congestion 계산 방법은 논문에서 언급하지 않음

2.2 Details

DEVIEW 2021

Reward Function: Density

- 특정 Grid Cell 영역이 과도하게 사용되는 것을 막기 위해 Penalty 적용
- 더불어 Density Threshold를 넘는 Grid에 대해 Masking 적용 (Hard Constraint)
- Density Threshold 값은 0.6으로 설정
- 구체적인 Density 계산 방법은 논문에서 언급하지 않음

2.2 Details

RL Agent

- 분산 PPO 사용
- Pretrained Model을 16개 Worker로 6시간 동안 Fine-Tuning
  (Pretraining은 20개 Worker로 48시간 학습)
- 각 Worker마다 10개의 CPU와 2GB의 메모리 사용
- Nvidia Volta Graphics Processing Unit (GPU) 사용

2.3 Experimental Results

DEVIEW 2021

Name	Method	Timing		Total area (μm²)	Total power (W)	Wirelength (m)	Congestion	
		WNS (ps)	TNS (ns)				H (%)	V (%)
Block 1	RePlAce	374	233.7	1,693,139	3.70	52.14	1.82	0.06
	Manual	136	47.6	1,680,790	3.74	51.12	0.13	0.03
	Our method	84	23.3	1,681,767	3.59	51.29	0.34	0.03
Block 2	RePlAce	97	6.6	785,655	3.52	61.07	1.58	0.06
	Manual	75	98.1	830,470	3.56	62.92	0.23	0.04
	Our method	59	170	694,757	3.13	59.11	0.45	0.03
Block 3	RePlAce	193	3.9	867,390	1.36	18.84	0.19	0.05
	Manual	18	0.2	869,779	1.42	20.74	0.22	0.07
	Our method	11	2.2	868,101	1.38	20.80	0.04	0.04
Block 4	RePlAce	58	11.2	944,211	2.21	27.37	0.03	0.03
	Manual	58	17.9	947,766	2.17	29.16	0.00	0.01
	Our method	52	0.7	942,867	2.21	28.50	0.03	0.02
Block 5	RePlAce	156	254.6	1,477,283	3.24	31.83	0.04	0.03
	Manual	107	97.2	1,480,881	3.23	37.99	0.00	0.01
	Ours	68	141.0	1,472,302	3.28	36.59	0.01	0.03

Here, we compare our method with the state-of-the-art method (RePlAce[14]) and with manual placements using an industry-standard EDA tool. For all metrics in this table, lower is better.
H, horizontal; V, vertical.

3. Hands-On Experience

3.1 Objective

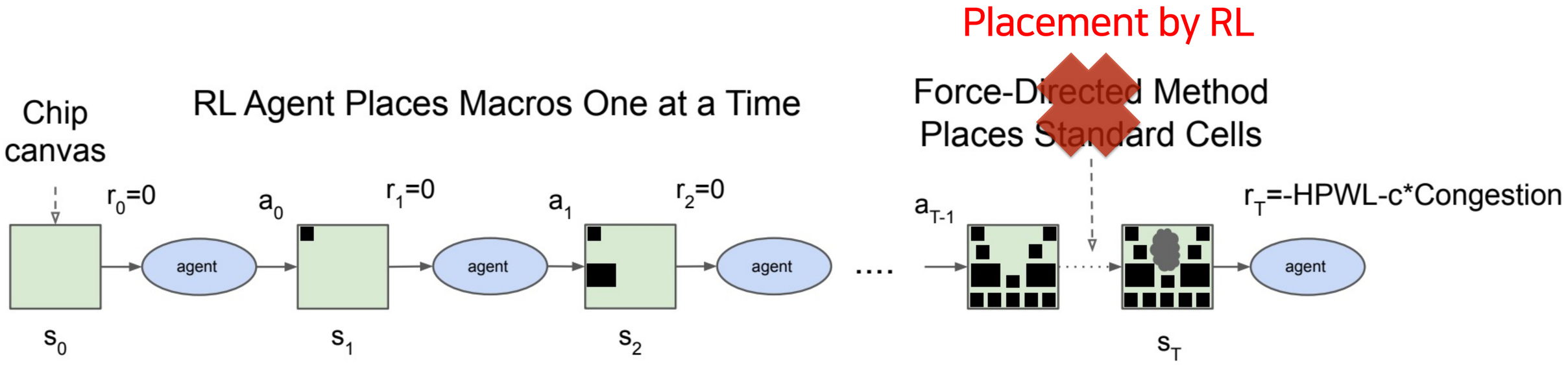
DEVIEW 2021

Key Objectives

- 좀 더 단순한 배치방식 (e. g. without FDM)
- 구글 논문에서 구체적으로 언급하지 않은 Congestion 추정법 등의 구체화
- 배치에 대한 공간적인 정보를 직접적으로 활용하는 Neural Network 구조 사용
- Generalization을 배제하고, 특정 Netlist에 대한 From-Scratch 학습을 목표로 함

3.2 Overview

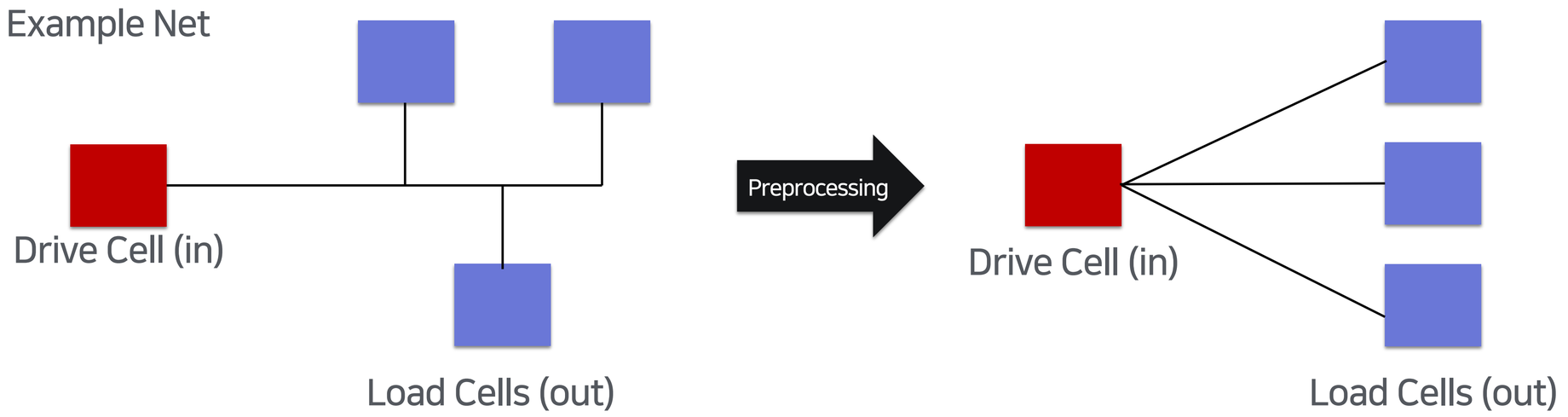
RL Process



3.3 RL Environment

States

- 각 소자의 Feature 정보 (width, height, is_macro, n_pins)
- 이번 step에서 배치할 Cell의 Index (Macro -> STD Cell, 큰 -> 작은 순서로 배치)
- 각 소자의 연결관계 (Nets)
- Hypergraph 형태의 연결관계를 Ordinary Graph 형태의 연결관계로 전처리

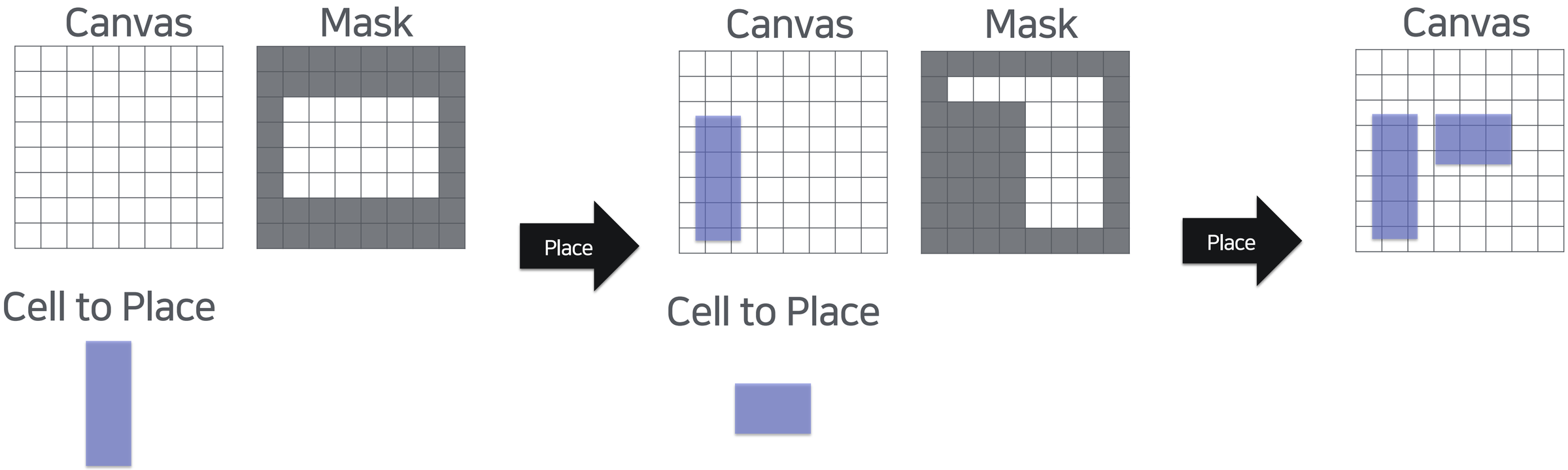


3.3 RL Environment

DEVIEW 2021

Actions

- N x N으로 나눈 Canvas 공간의 Grid Index
- 각 Grid의 중앙에 Cell을 배치
- 다음 Cell을 배치할 수 없는 영역은 Masking

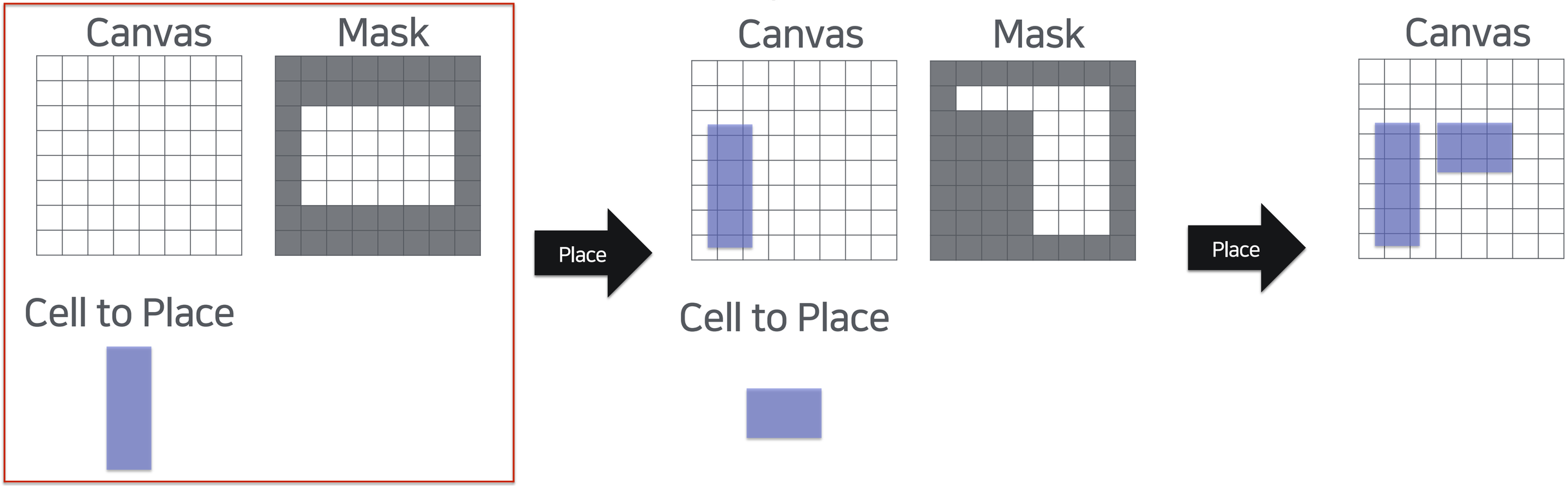


3.3 RL Environment

DEVIEW 2021

Actions

- N x N으로 나눈 Canvas 공간의 Grid Index
- 각 Grid의 중앙에 Cell을 배치
- 다음 Cell을 배치할 수 없는 영역은 Masking

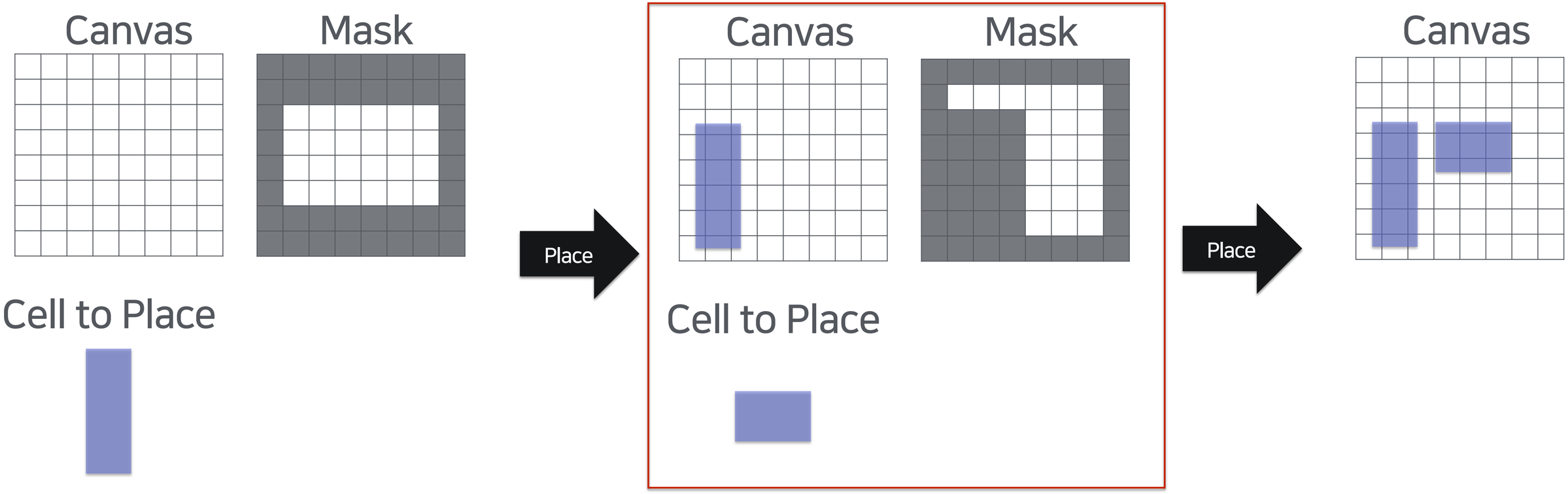


3.3 RL Environment

DEVIEW 2021

Actions

- N x N으로 나눈 Canvas 공간의 Grid Index
- 각 Grid의 중앙에 Cell을 배치
- 다음 Cell을 배치할 수 없는 영역은 Masking



3.3 RL Environment

DEVIEW 2021

Rewards

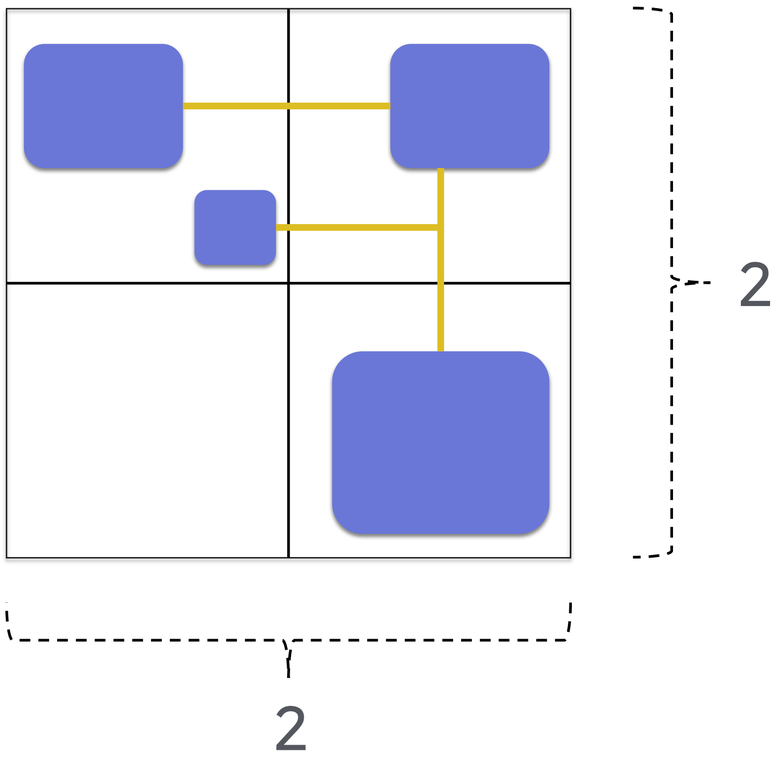
- Wirelength와 Congestion의 가중합산 (p: placement, g: graph)
- 전체 스케일 조정을 위해 Wirelength, Congestion 각각에 Coefficient 적용

$$R_{p,g} = -\alpha\, Wirelength(p,g) - \beta\, Congestion(p,g)$$

3.3 RL Environment

Rewards: Wirelength

- 모든 Net에 대한 HPWL (Half-Perimeter Wire Length)의 총합



$$\mathrm{HPWL}(i) = (\max_{b \in i}\{x_b\} - \min_{b \in i}\{x_b\} + 1) + (\max_{b \in i}\{y_b\} - \min_{b \in i}\{y_b\} + 1).$$

3.3 RL Environment

DEVIEW 2021

Rewards: Congestion

- Congestion은 Grid Cell에서의 Routing Resource 대비, Net이 요구하는 Routing Resource로 정의 (v: Grid Cell)
- 즉, Demand를 줄이는 것으로 전체적인 Congestion을 낮출 수 있음

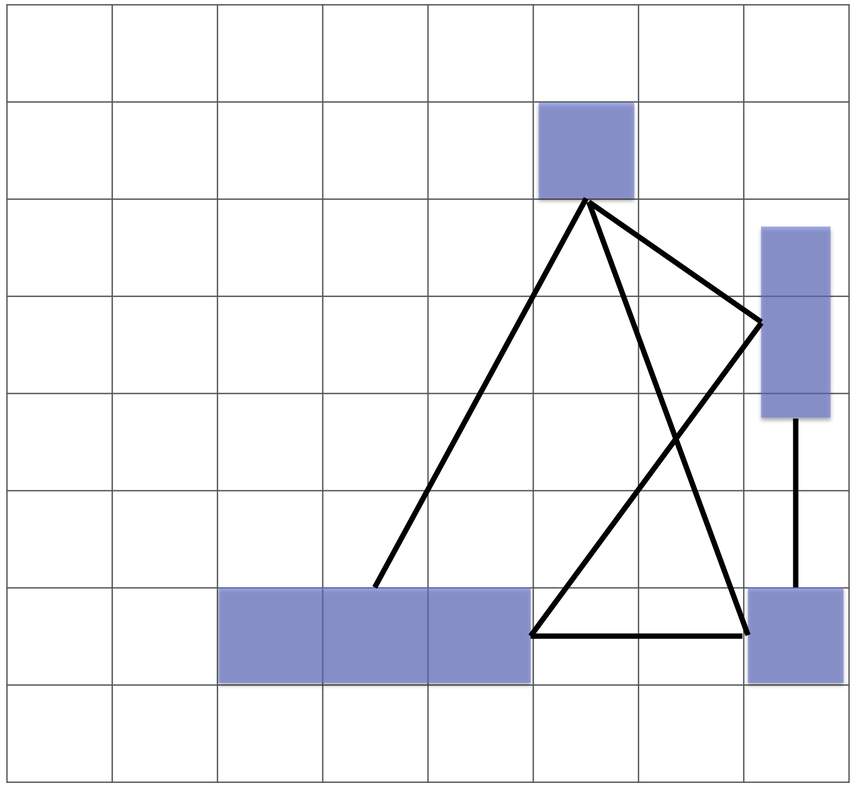
$$C(v) = \frac{demand(v)}{supply(v)}.$$

demand와 비례관계

3.3 RL Environment

Rewards: Routing Resource Demand (1)

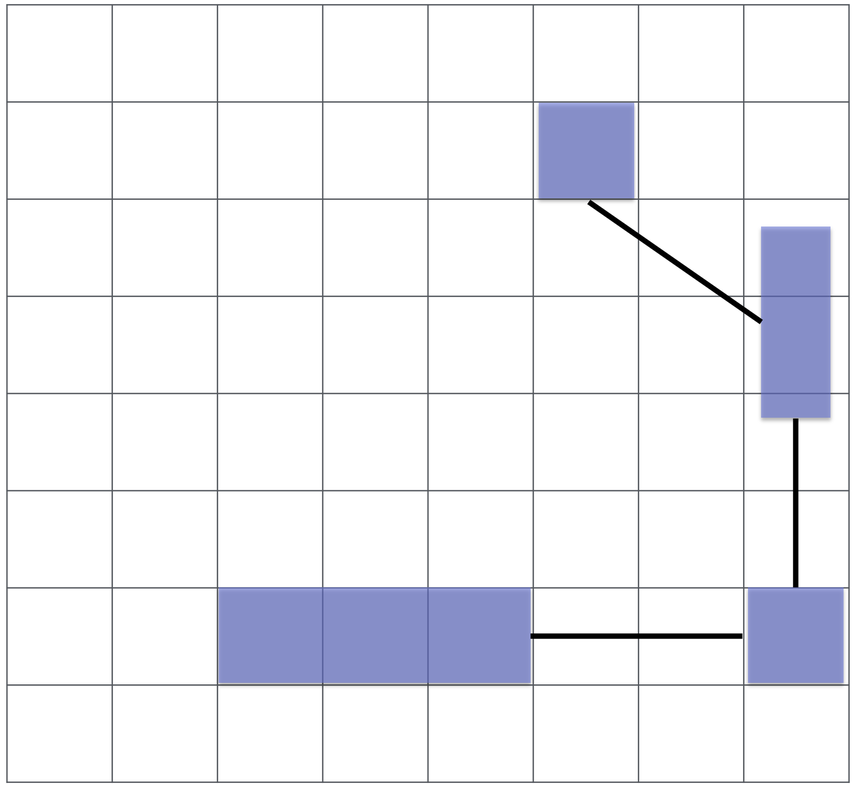
- 배치가 완료된 Net을 Complete Graph로 표현



3.3 RL Environment

Rewards: Routing Resource Demand (2)

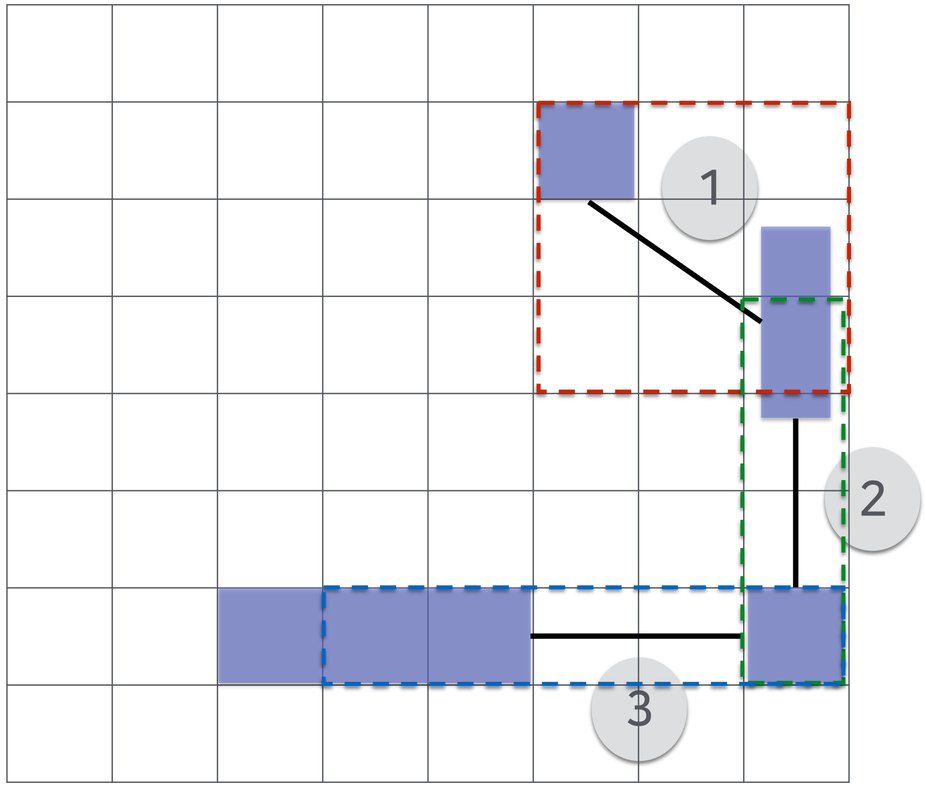
- Complete Graph를 Minimum Spanning Tree (또는 Steiner Tree)로 변환



3.3 RL Environment

Rewards: Routing Resource Demand (3)

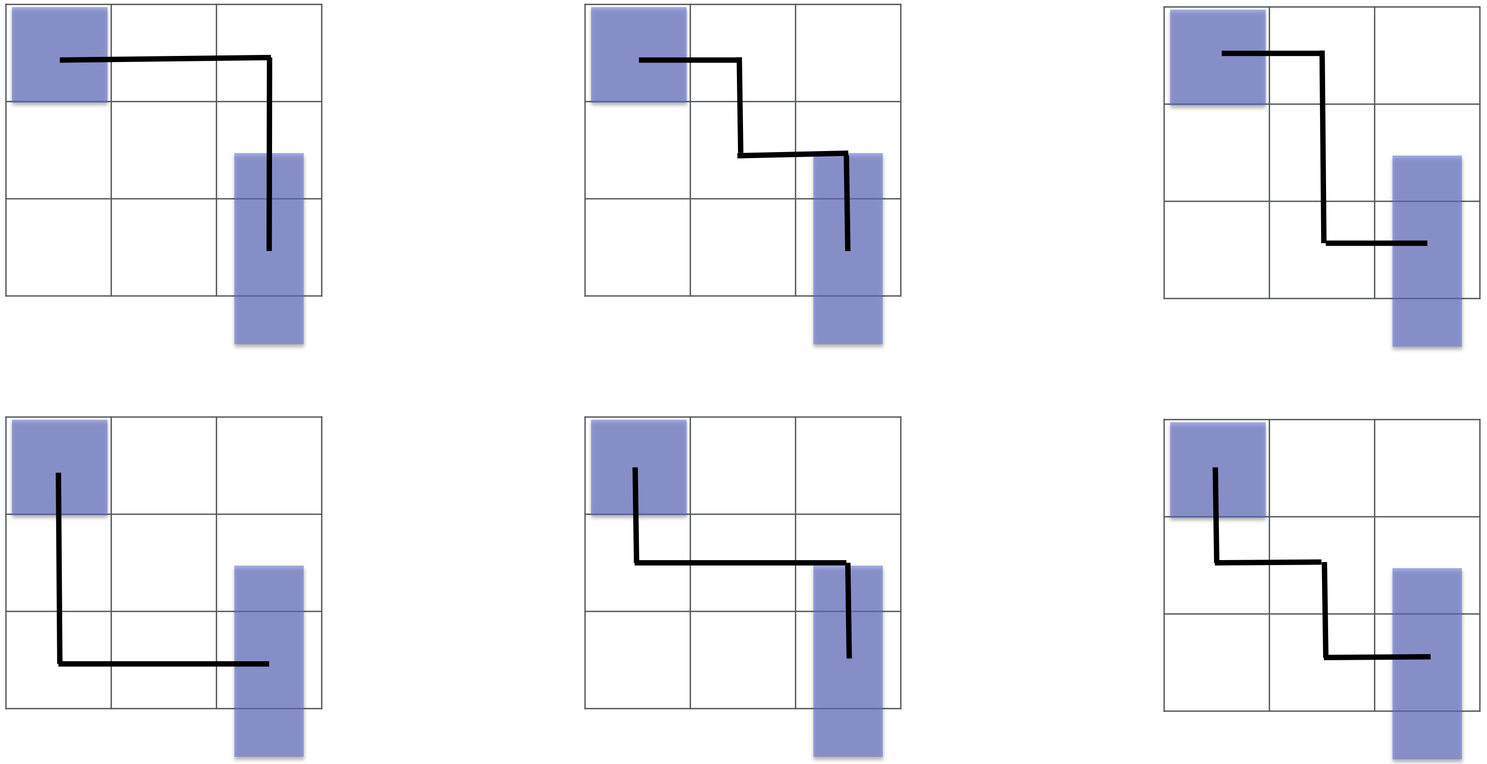
- 각 Edge에 대한 Routing Resource Demand를 계산



3.3 RL Environment

Rewards: Routing Resource Demand (4)

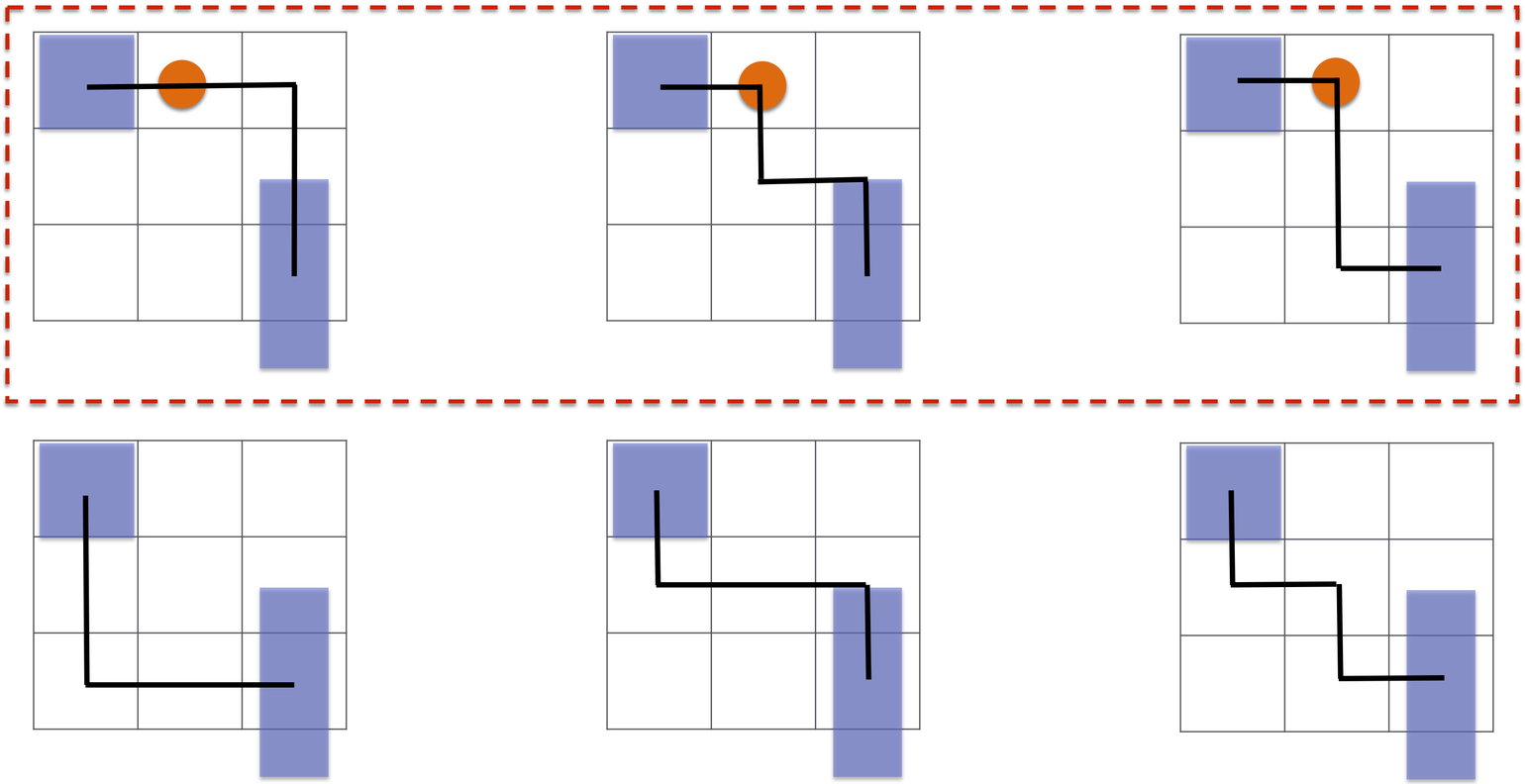
- 두 소자를 연결할 수 있는 모든 경우의 수 계산 (총 6개)



3.3 RL Environment

Rewards: Routing Resource Demand (4)

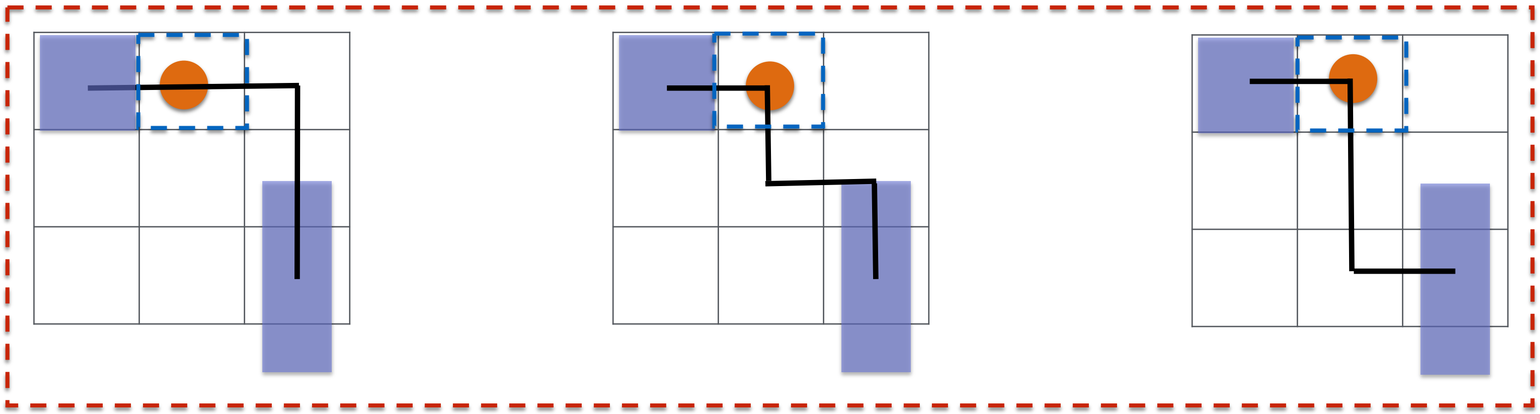
- 두 소자를 연결할 수 있는 모든 경우의 수 계산 (총 6개)



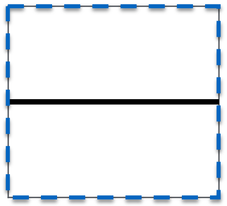
3.3 RL Environment

Rewards: Routing Resource Demand (5)

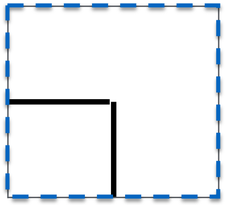
- Grid Cell (0,1) 에서의 Expected Routing Resource Demand를 계산



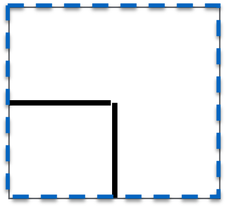
Horizontal Demand 1.0



Horizontal Demand 0.5
Vertical Demand 0.5



Horizontal Demand 0.5
Vertical Demand 0.5



Horizontal Demand	1/6 x 1	+	1/6 x 0.5	+	1/6 x 0.5	= 4 / 12
Vertical Demand	0	+	1/6 x 0.5	+	1/6 x 0.5	= 2 / 12

3.3 RL Environment

DEVIEW 2021

Rewards: Routing Resource Demand (6)

- 계산한 4/12, 2/12를 각각 Expected Routing Resource Demand Map에 기입

Expected Horizontal Demand

	4/12	

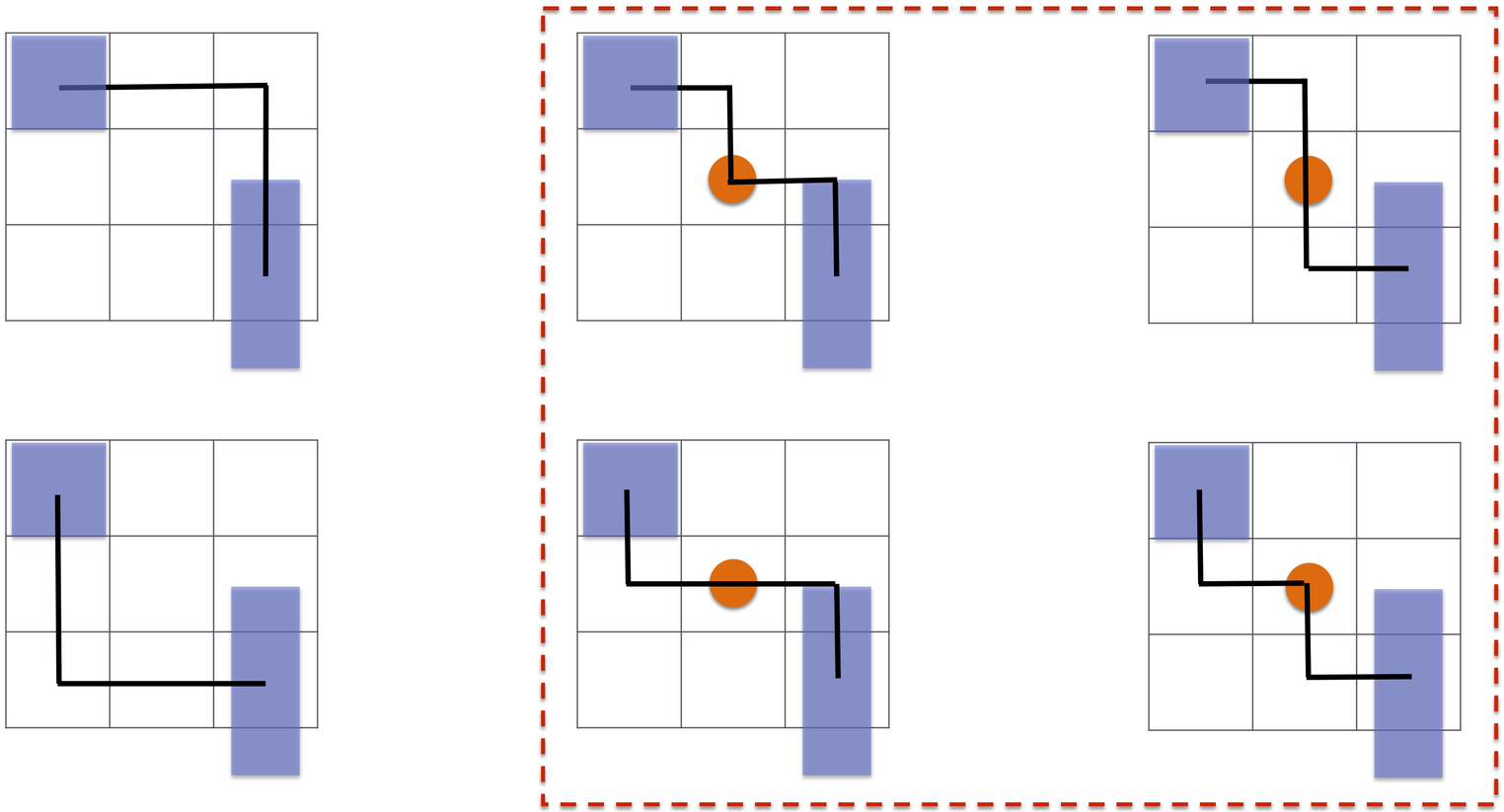
Expected Vertical Demand

	2/12	

3.3 RL Environment

Rewards: Routing Resource Demand (7)

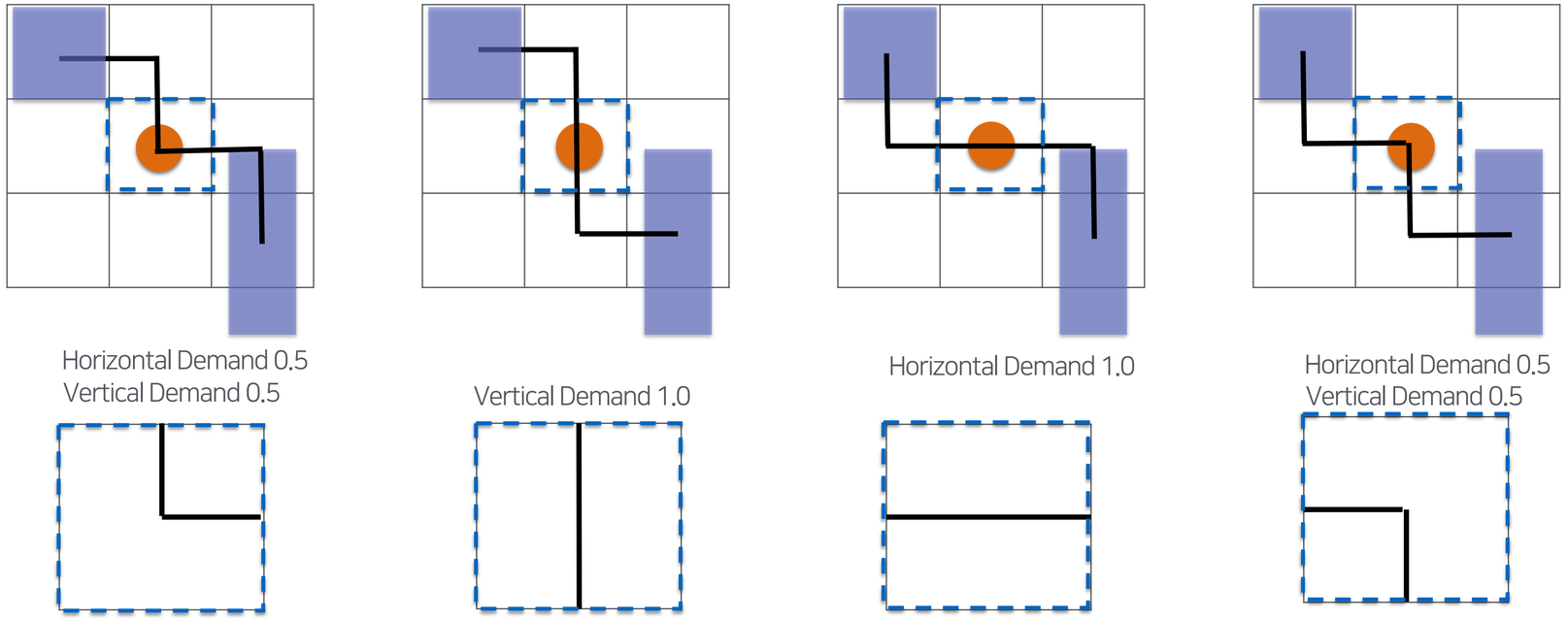
- Grid Cell (1, 1)을 지나는 경로의 수 (총 4개)



3.3 RL Environment

Rewards: Routing Resource Demand (8)

- Grid Cell (1,1) 에서의 Expected Routing Resource Demand를 계산



Horizontal Demand	1/6 x 0.5	+	0	+	1/6 x 1.0	+	1/6 x 0.5	= 4 / 12
Vertical Demand	1/6 x 0.5	+	1/6 x 1.0	+	0	+	1/6 x 0.5	= 4 / 12

3.3 RL Environment

Rewards: Routing Resource Demand (9)

- 계산한 4/12, 4/12를 각각 Expected Routing Resource Demand Map에 기입

Expected Horizontal Demand

	4/12	
	4/12	

Expected Vertical Demand

	2/12	
	4/12	

3.3 RL Environment

Rewards: Routing Resource Demand (10)

- 같은 방법으로 모든 Grid Cell에 대한 Expected Routing Resource Demand를 계산

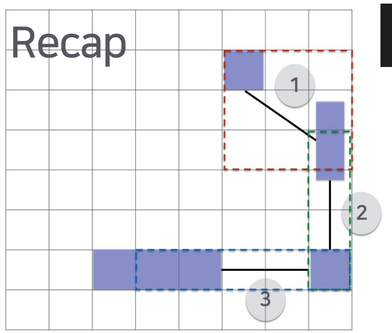
Expected Horizontal Demand

3/12	4/12	1/12
2/12	4/12	2/12
1/12	4/12	3/12

Expected Vertical Demand

3/12	2/12	1/12
4/12	4/12	4/12
1/12	2/12	3/12

3.3 RL Environment



DEVIEW 2021

Rewards: Routing Resource Demand (11)

- 계산 결과를 Global Congestion Map에 반영

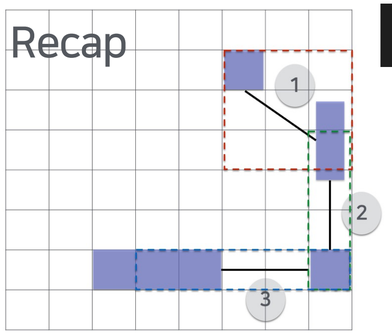
Horizontal Global Congestion Map

					3/12	4/12	1/12
					2/12	4/12	2/12
					1/12	4/12	3/12

Vertical Global Congestion Map

					3/12	2/12	1/12
					4/12	4/12	4/12
					1/12	2/12	3/12

3.3 RL Environment



Rewards: Routing Resource Demand (12)

- 다음 Edge에 대한 Congestion Demand를 Global Congestion Map에 누적합 (반복)

Horizontal Global Congestion Map

					3/12	4/12	1/12
					2/12	4/12	2/12
					1/12	4/12	3/12
							0
							0
							0

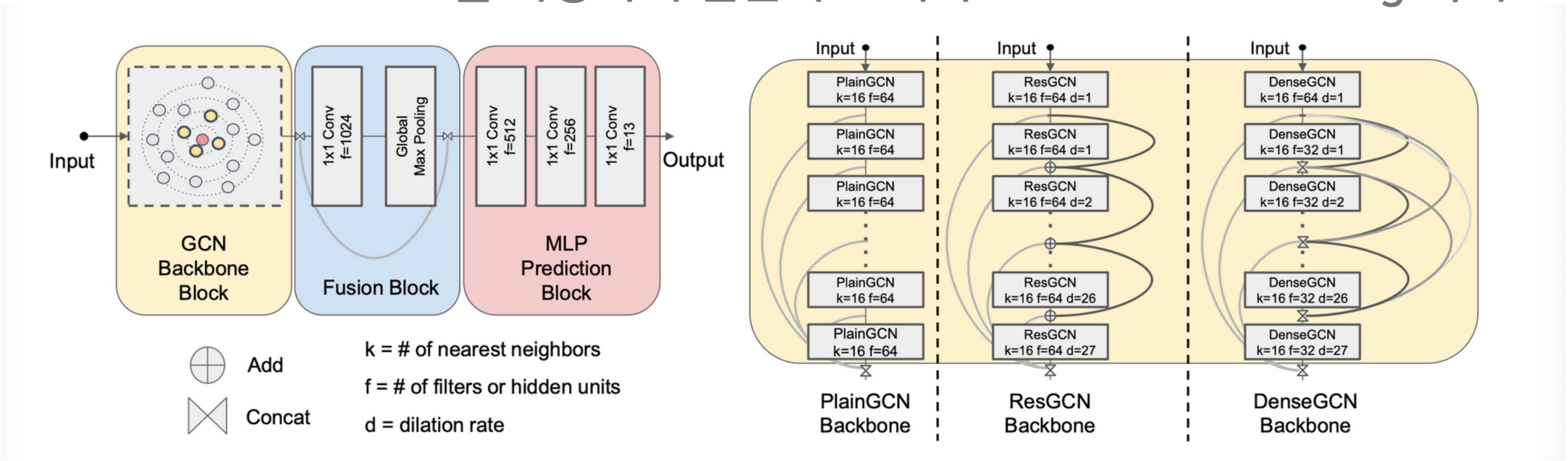
Vertical Global Congestion Map

					3/12	2/12	1/12
					4/12	4/12	4/12
					1/12	2/12	9/12
							1
							1
							6/12

3.4 Graph Neural Networks

Graph Neural Network

- Implicit GNN 보다 Automatic Differentiation Tool(e. g. PyTorch)이용한 구현 편의성이 높은 DeeperGCN 구조 사용
- Residual Connection을 사용하여 깊은 구조에서도 Over-Smoothing 억제

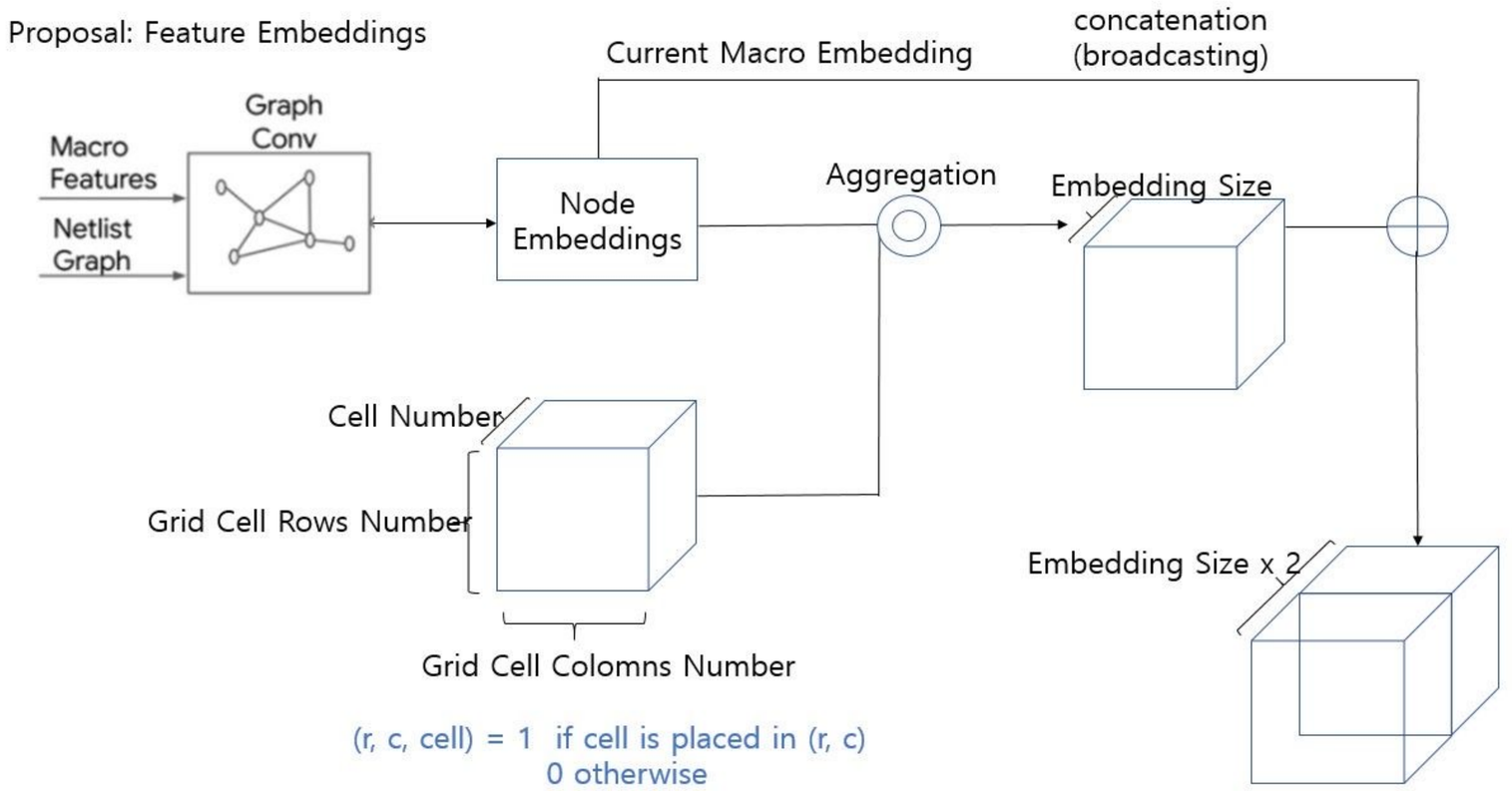


G. Li et al. DeepGCN: Can GCNs Go as Deep as CNNs?, ICCV 2019
G. Li et al. DeeperGCN: All You Need to Train Deeper GCNs, arXiv 2020

3.5 Policy Network

Policy Network (1)

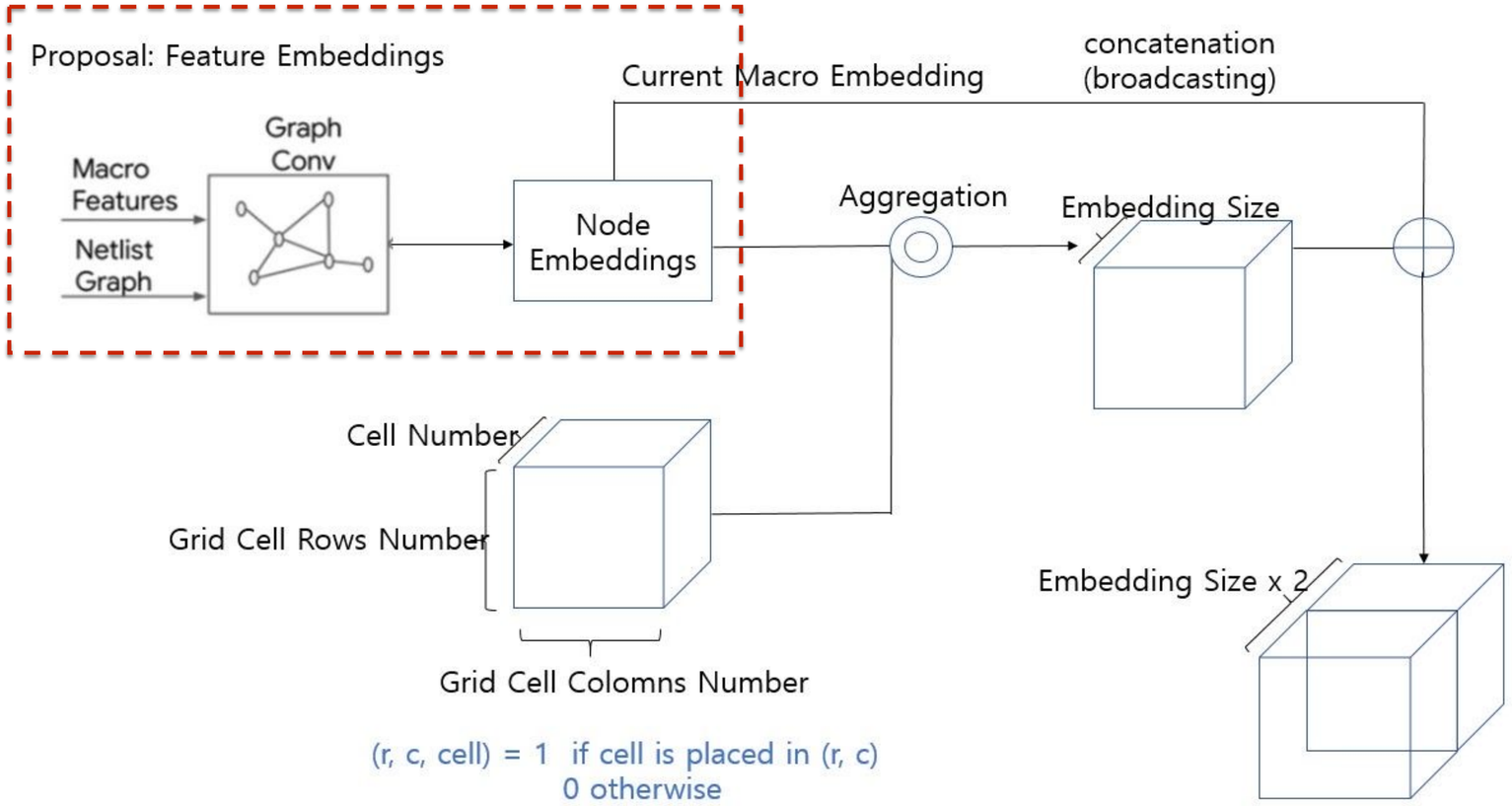
- 배치된 Cell 들의 Embedding과 배치할 Cell의 Embedding을 2D Grid Space 상에서 Aggregation 및 Concatenation



3.5 Policy Network

Policy Network (1)

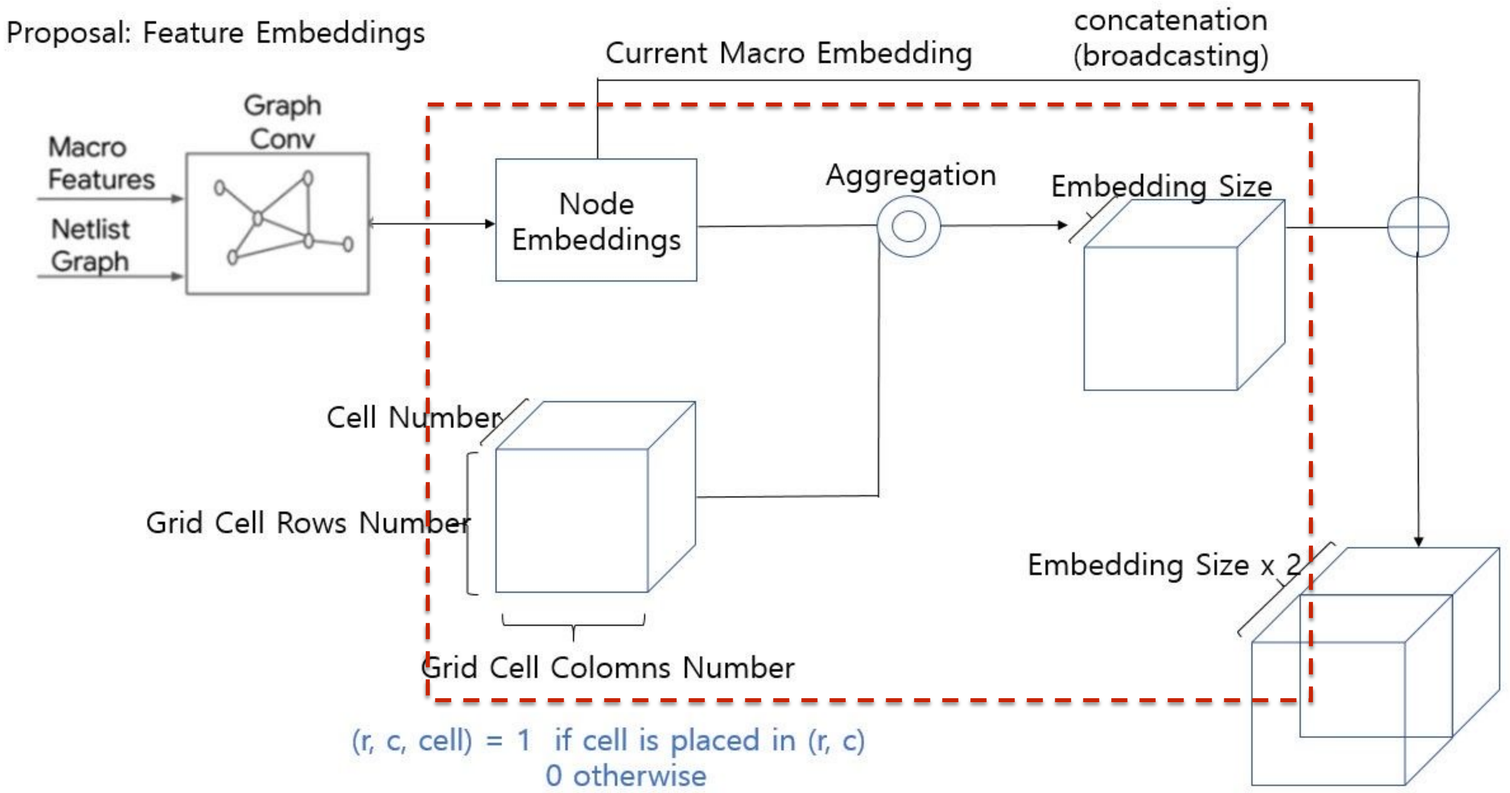
- 배치된 Cell 들의 Embedding과 배치할 Cell의 Embedding을 2D Grid Space 상에서 Aggregation 및 Concatenation



3.5 Policy Network

Policy Network (1)

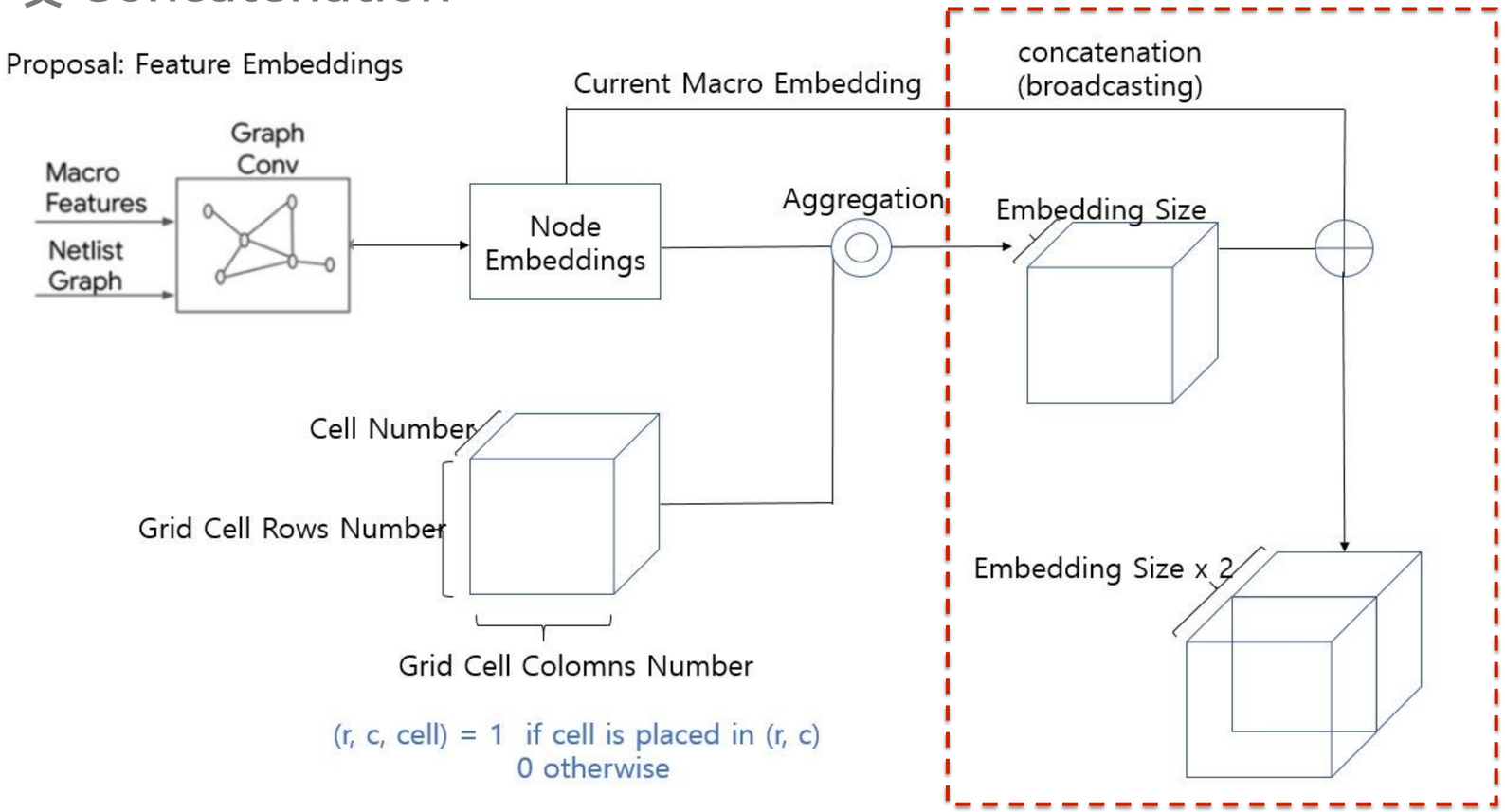
- 배치된 Cell 들의 Embedding과 배치할 Cell의 Embedding을 2D Grid Space 상에서 Aggregation 및 Concatenation



3.5 Policy Network

Policy Network (1)

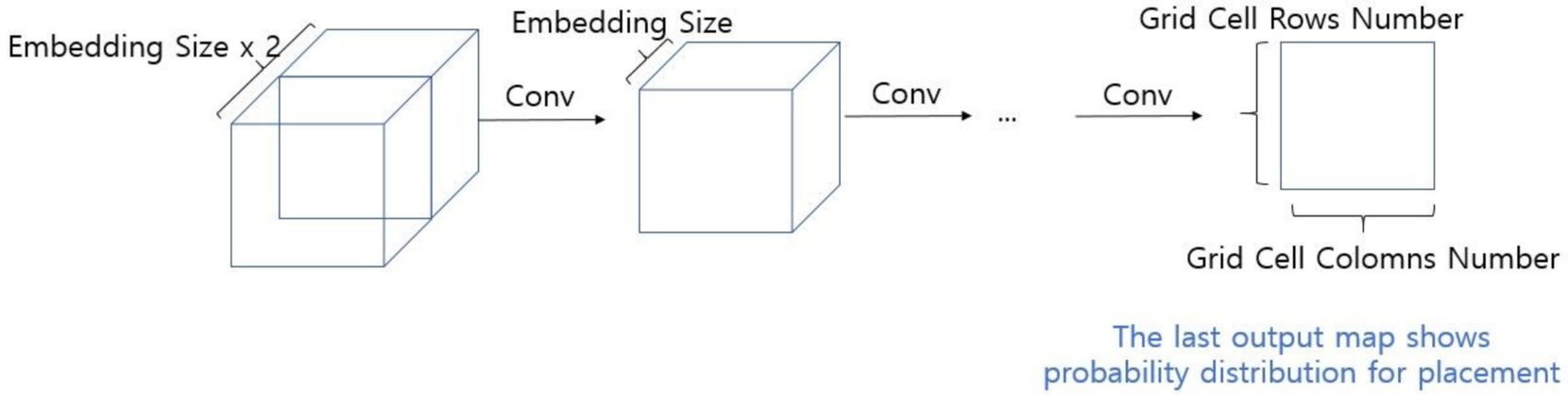
- 배치된 Cell 들의 Embedding과 배치할 Cell의 Embedding을 2D Grid Space 상에서 Aggregation 및 Concatenation



3.5 Policy Network

Policy Network (2)

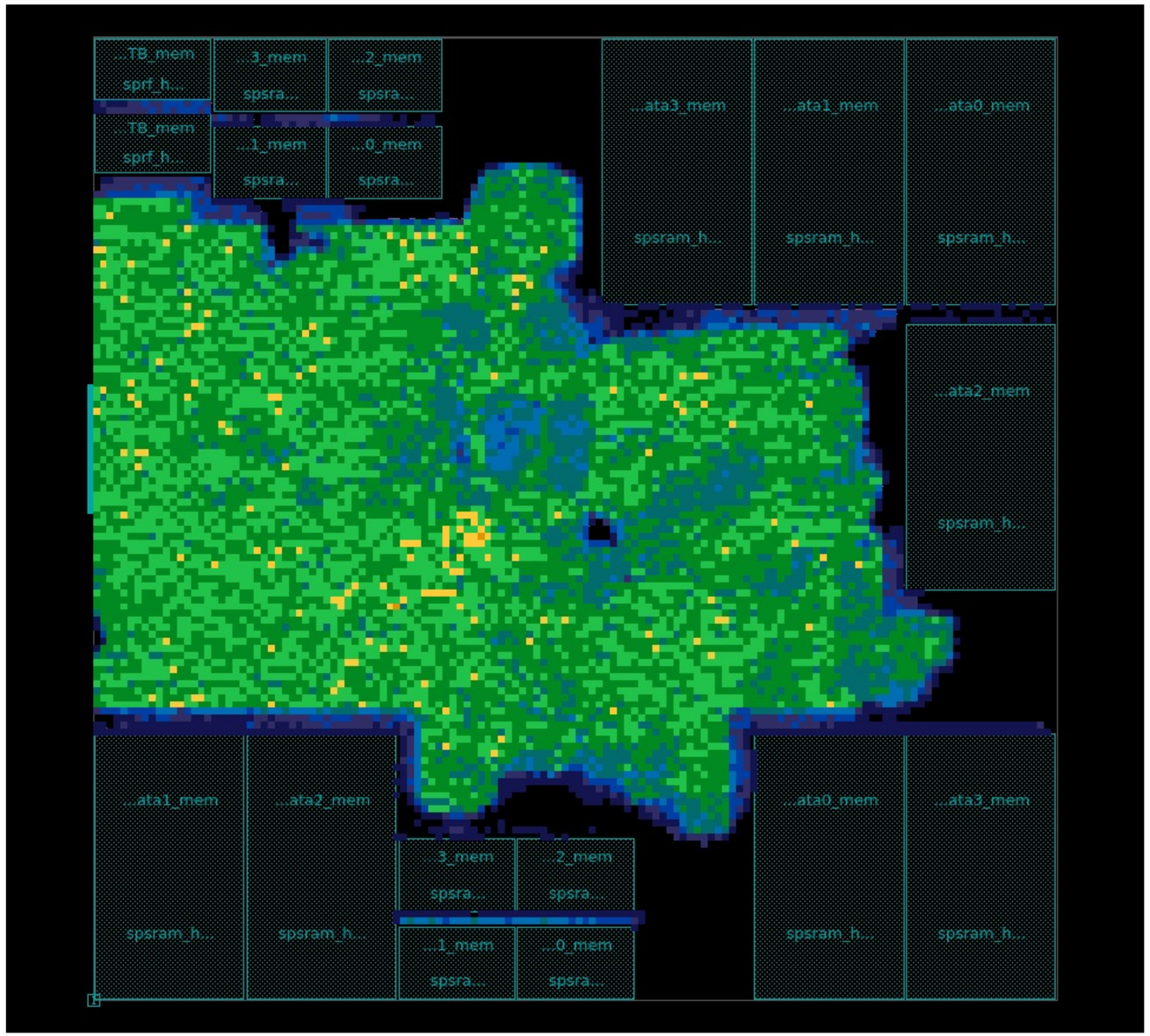
- 앞단계에서의 (N_ROWS, N_COLS, EMBEDDING_SIZE x 2)의 Embedding을 Convolutional Neural Network(e. g. VGG)를 통해 점진적으로 축소



3.6 Netlist

Netlist

- Macro 개수: 18
- 전체 Cell 개수: 122,406
- 전체 Net 개수: 127,548
- Single Core CPU with L1 Cache



3.7 Evaluation

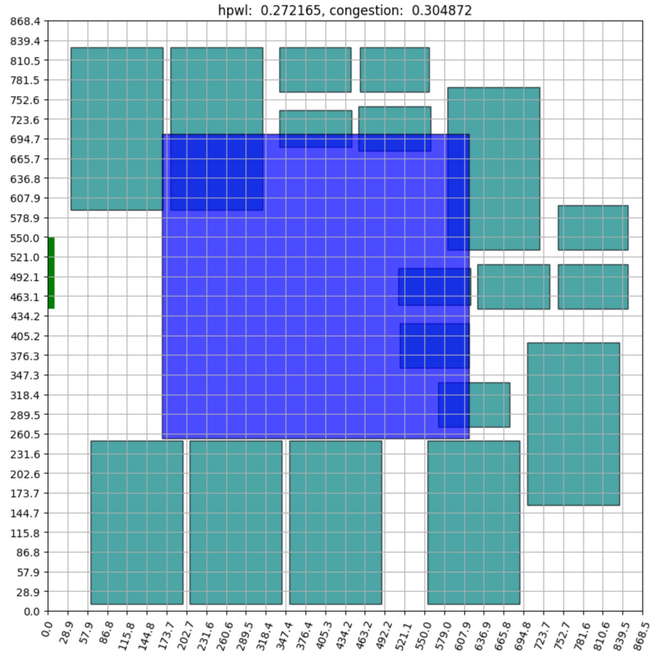
Evaluation Metrics

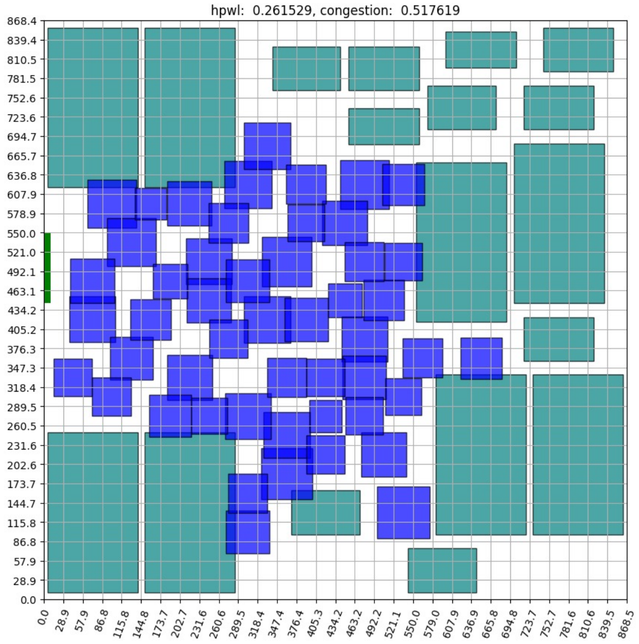
- WNS (Worst Negative Slack): 현재의 설계가 특정 Clock Frequency를 유지할 수 있는지 나타내는 척도 (음수: Violation)
- Frequency: 예상되는 Clock Frequency

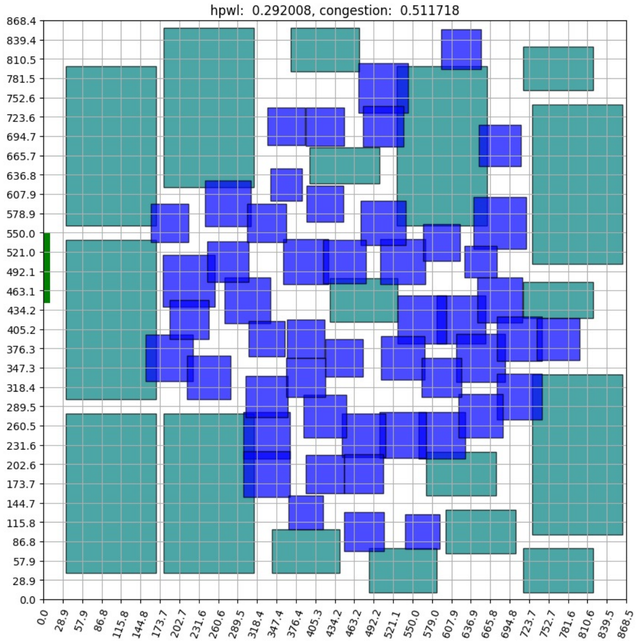
4. 실험 결과

4.1 Scenarios

- Macro placement with a single std cell cluster
- Separated masks for std cells and macros
- Placement w/o heuristics





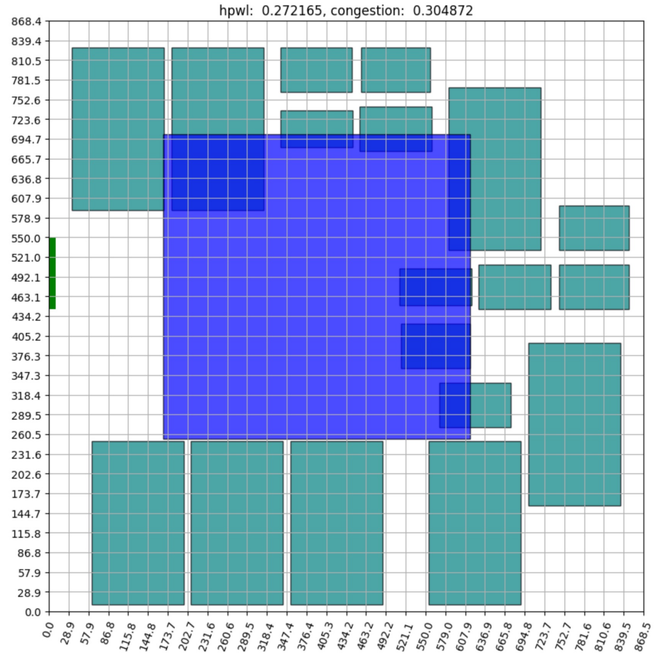


4.2 with a Single STD Cell Cluster

DEVIEW 2021

모든 Standard Cell을 하나의 Cluster로 표현

- Backend Engineer들이 Floorplan 시, Macro Cell을 주로 Canvas의 모서리에 위치시키고, Standard Cell이 Canvas의 가운데로 배치되게끔 하는 것에 착안
- Macro cell의 경우 가운데 영역을 Masking하여 Canvas의 모서리에 배치

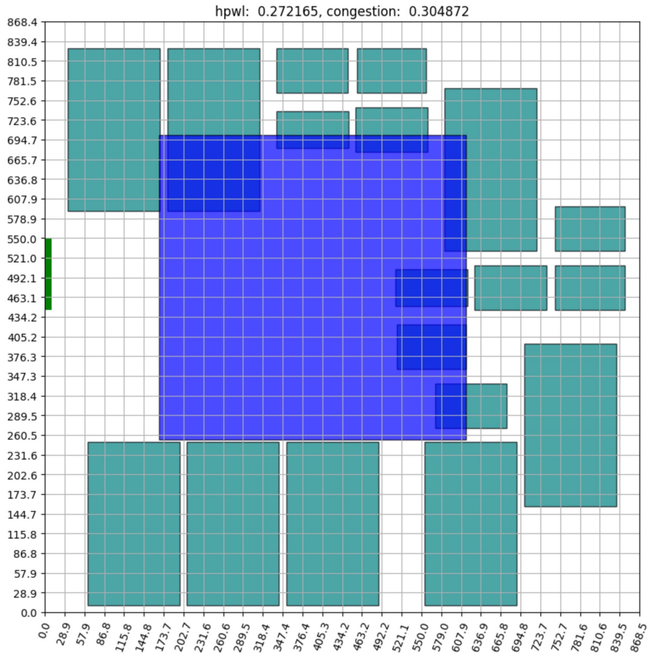


4.2 with a Single STD Cell Cluster

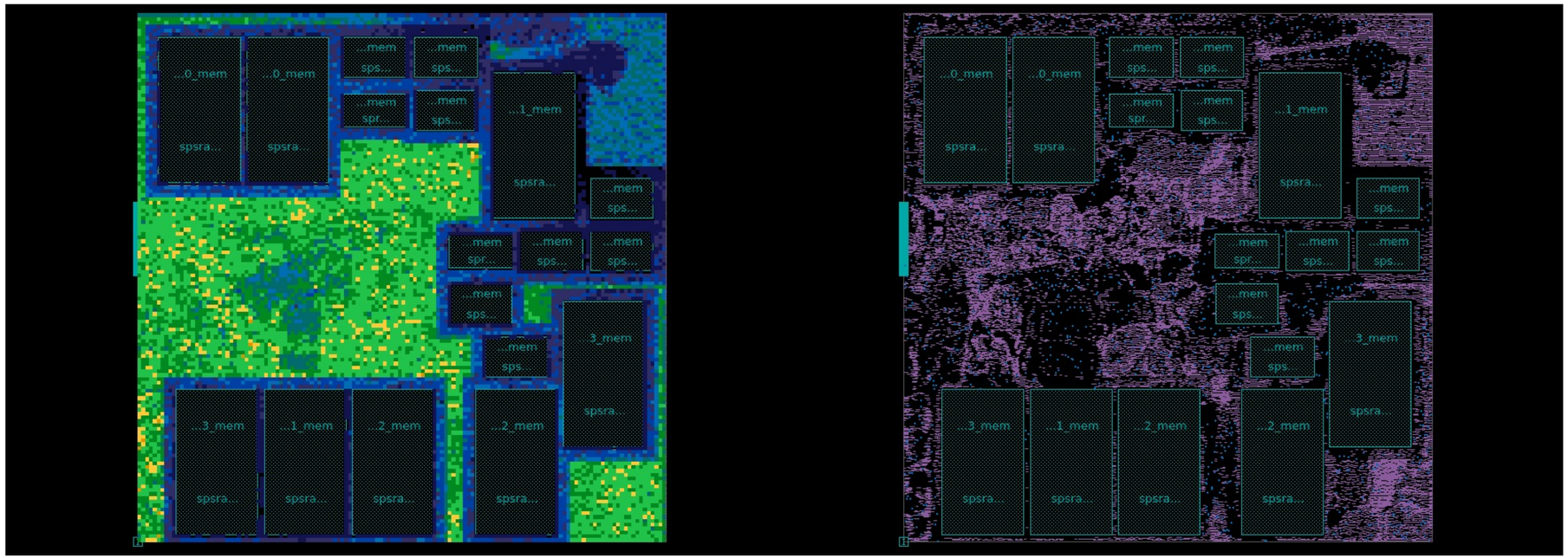
DEVIEW 2021

ICC2 Evaluation

RL Placement



Cell Density Map

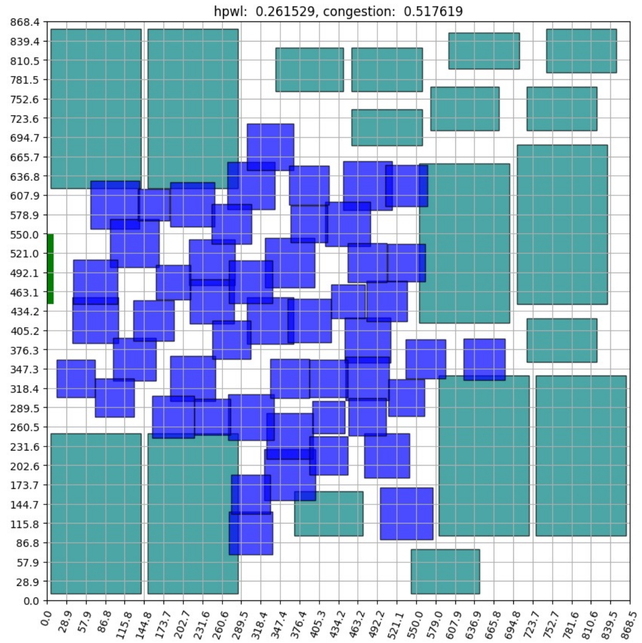


Global Congestion Map

4.3 Separated Masks for STD Cells and Macros

Standard Cell을 다수의 클러스터로 표현

- Netlist의 Group 정보를 분석하여 클러스터 개수를 50개로 설정
- Macro cell의 경우 가운데 영역을 Masking하여 Canvas의 모서리에 배치

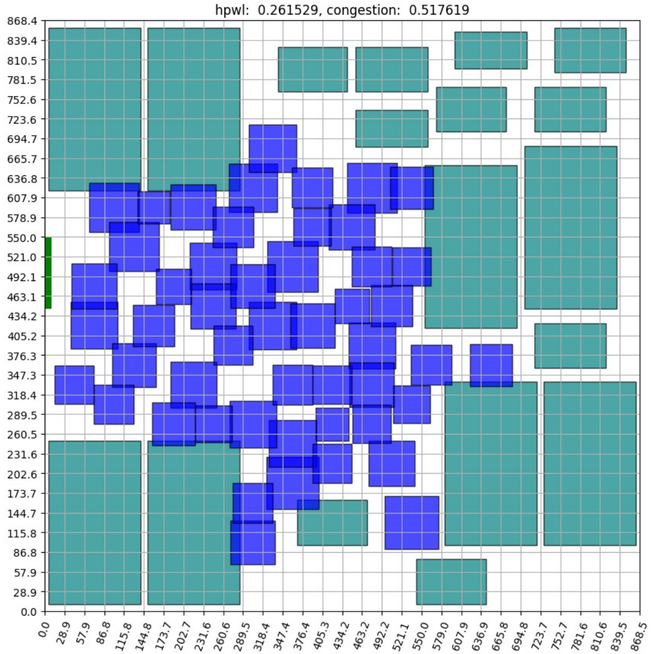


4.3 Separated Masks for STD Cells and Macros

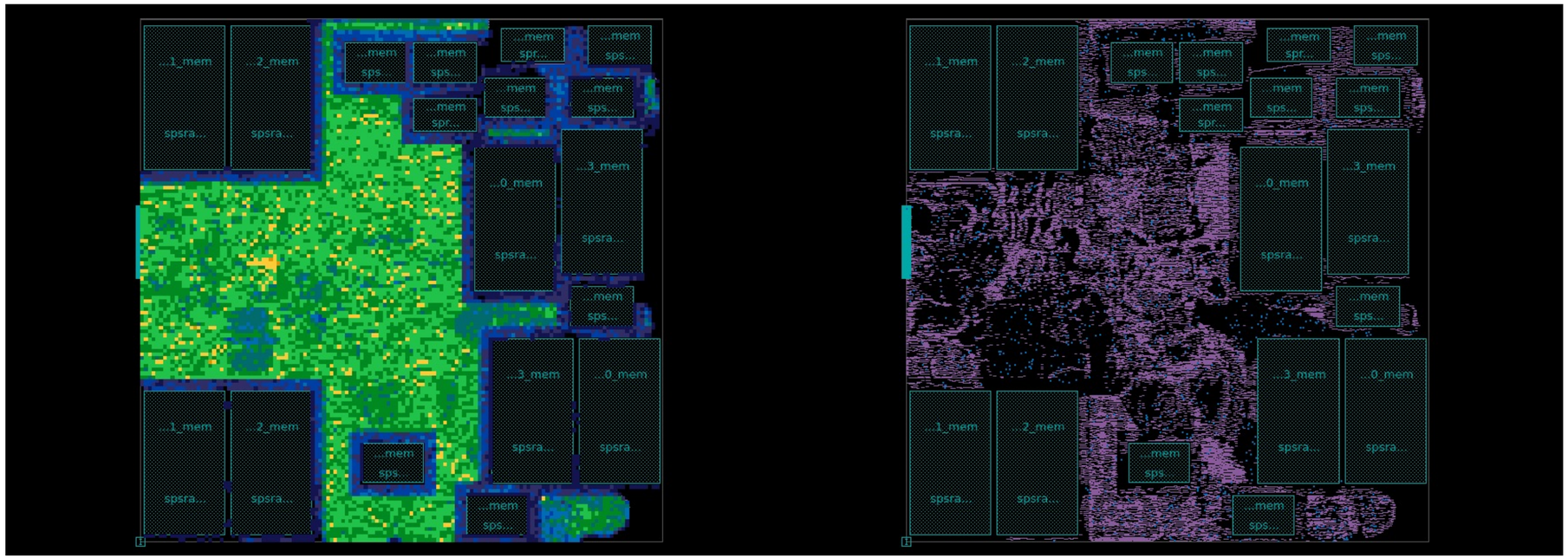
DEVIEW 2021

ICC2 Evaluation

RL Placement



Cell Density Map



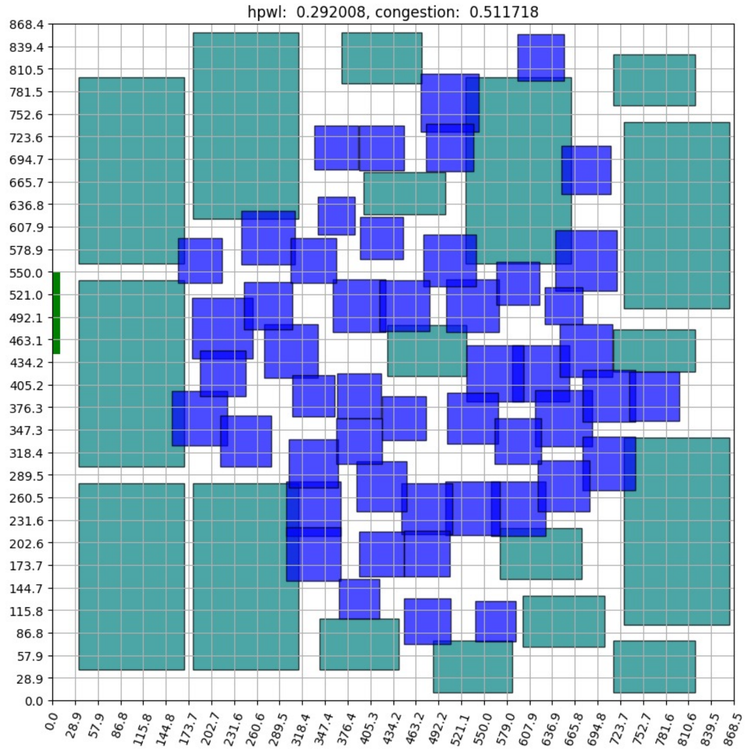
Global Congestion Map

4.4 Placement without Heuristics

DEVIEW 2021

Macro Cell과 STD Cell Cluster를 Canvas 전체 영역에 배치

- Macro Cell의 배치 위치에 대한 Heuristic을 제거

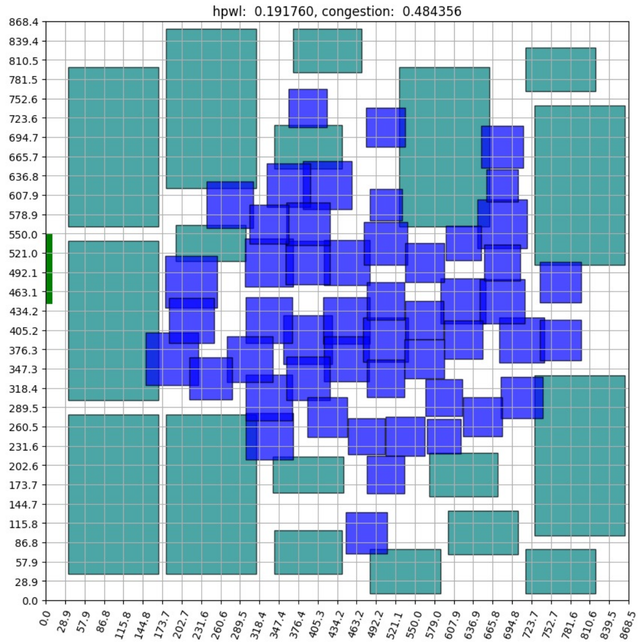


4.4 Placement without Heuristics

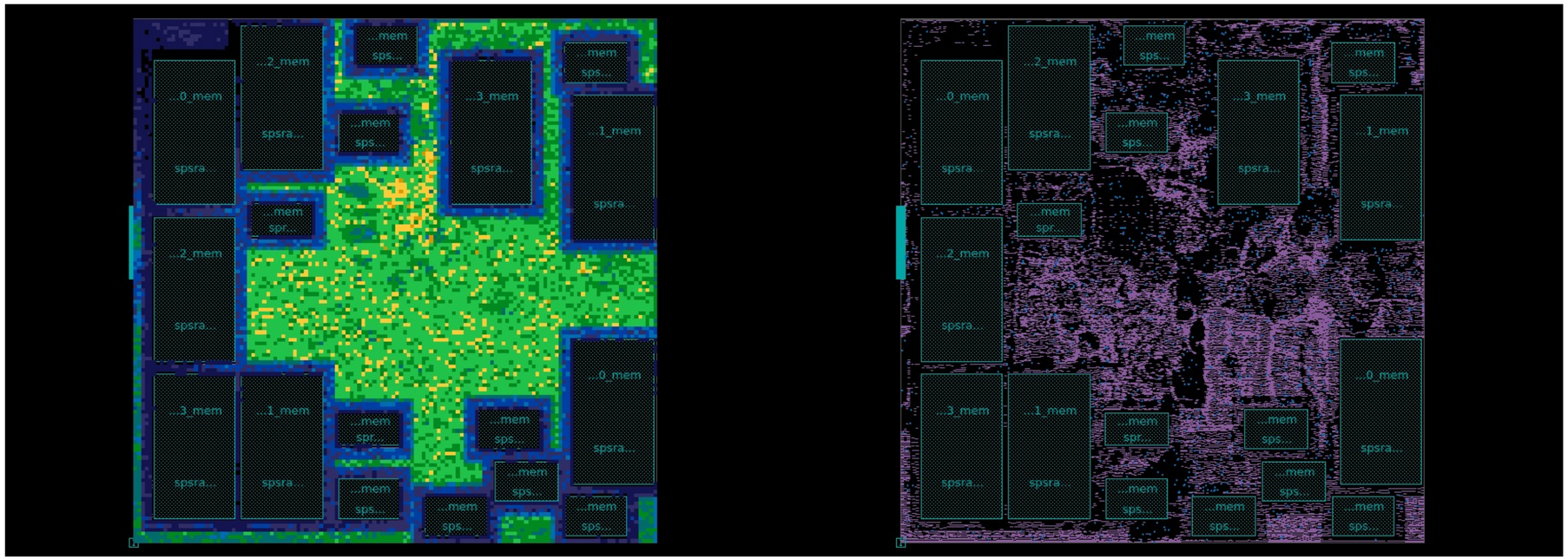
DEVIEW 2021

ICC2 Evaluation

RL Placement



Cell Density Map



Global Congestion Map

4.5 Experimental Results

DEVIEW 2021

Benchmark Results

- 앞선 세 가지 실험 케이스와 ICC2 Auto Placement 및 Human Placement의 배치 결과 비교 (테이블)

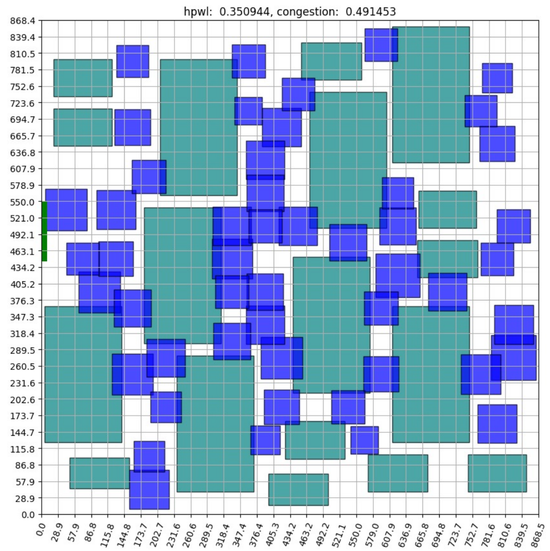
Path Group	Expert		ICC2		Single Cluster		Bounding Box		Non-Heuristic	
	WNS	FREQ	WNS	FREQ	WNS	FREQ	WNS	FREQ	WNS	FREQ
Group 0	0.00	143MHz	0.00	143MHz	0.00	143MHz	0.00	143MHz	0.00	143MHz
Group 1	0.05	144MHz	0.06	144MHz	0.14	146MHz	0.05	144MHz	0.07	144MHz
Group 2	0.45	153MHz	0.42	152MHz	0.49	154MHz	0.45	153MHz	0.42	152MHz
Group 3	0.79	161MHz	0.79	161MHz	0.65	157MHz	0.76	160MHz	0.79	160MHz
Group 4	2.60	227MHz	2.65	230MHz	2.66	230MHz	2.70	233MHz	2.55	230MHz
Group 5	2.67	231MHz	2.74	235MHz	2.73	234MHz	2.76	236MHz	2.63	234MHz

4.5 Experimental Results

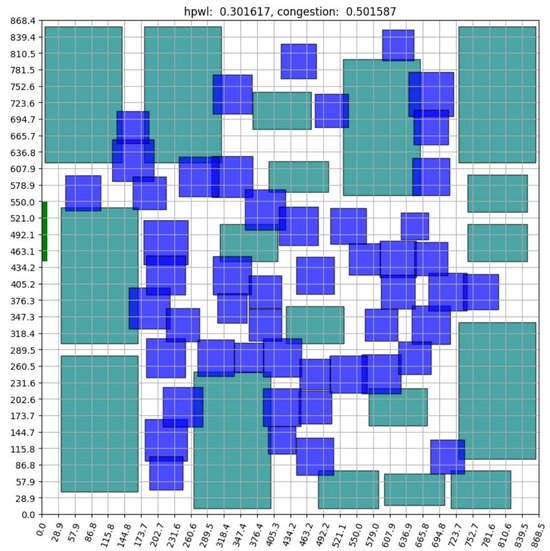
Placements during Training

- 학습이 진행됨에 따라 Standard Cell이 점진적으로 중앙으로 이동

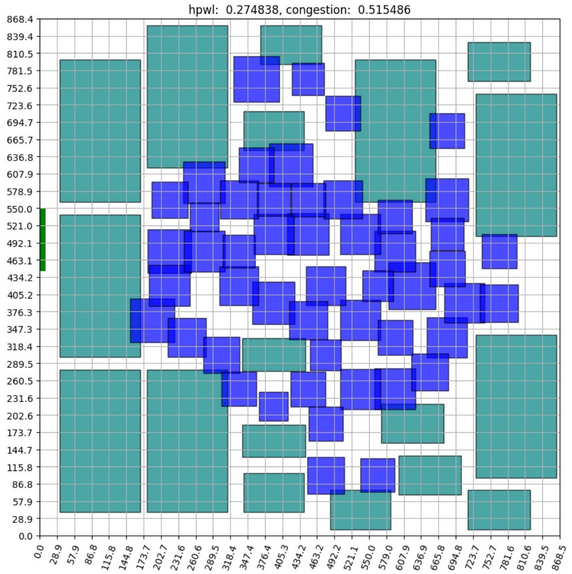
Iteration 1



Iteration 150



Iteration 500



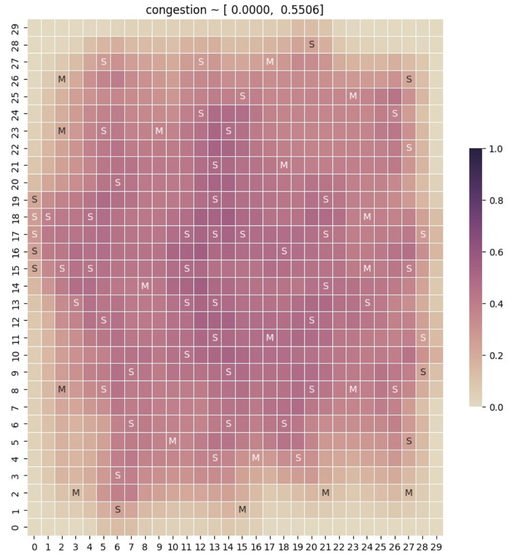
4.5 Experimental Results

DEVIEW 2021

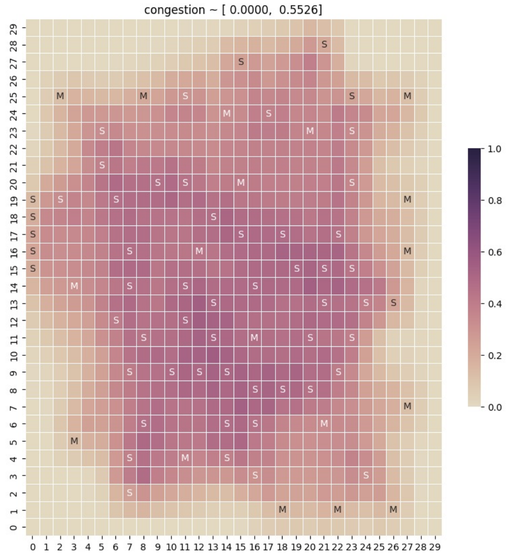
Congestion Maps during Training

- 학습이 진행됨에 따라 Congestion이 중앙으로 이동

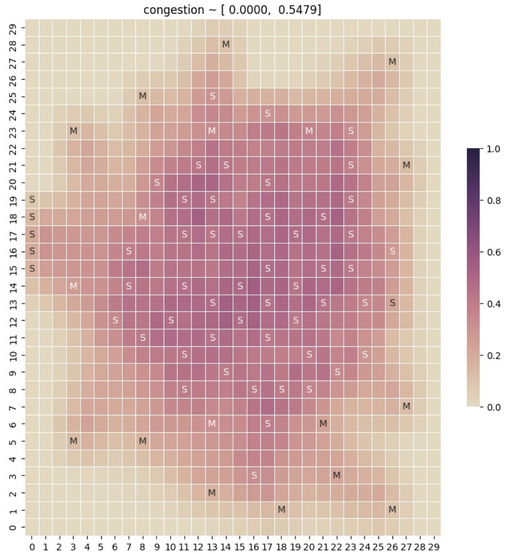
Iteration 1



Iteration 150



Iteration 1k

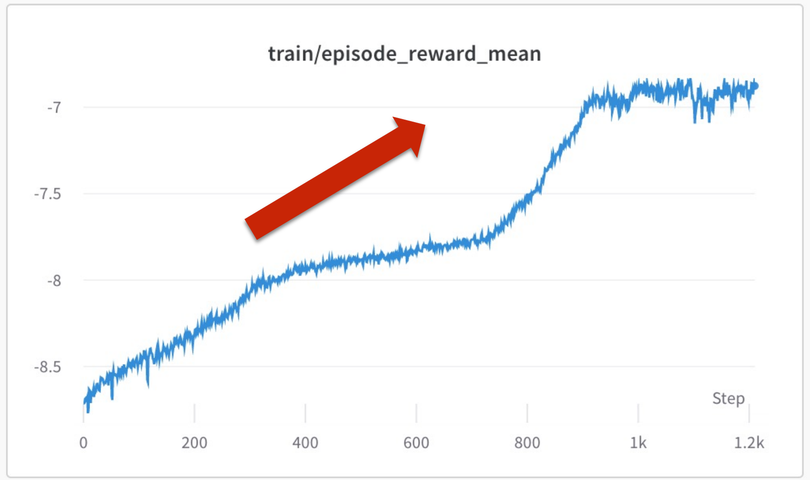


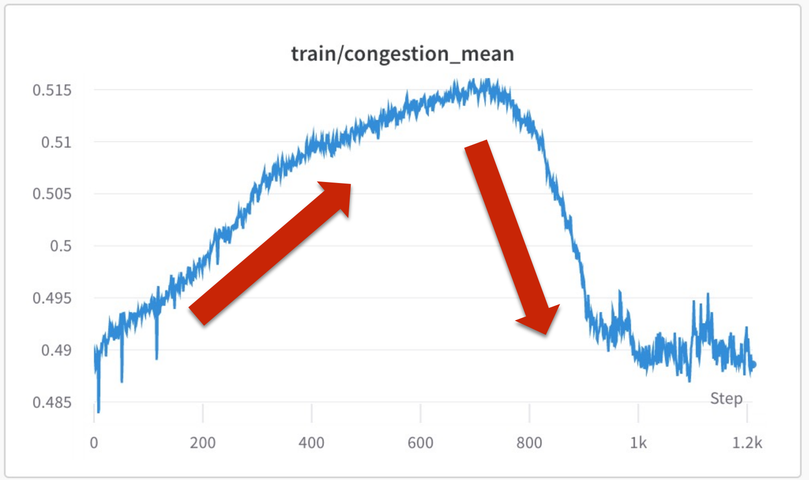
4.5 Experimental Results

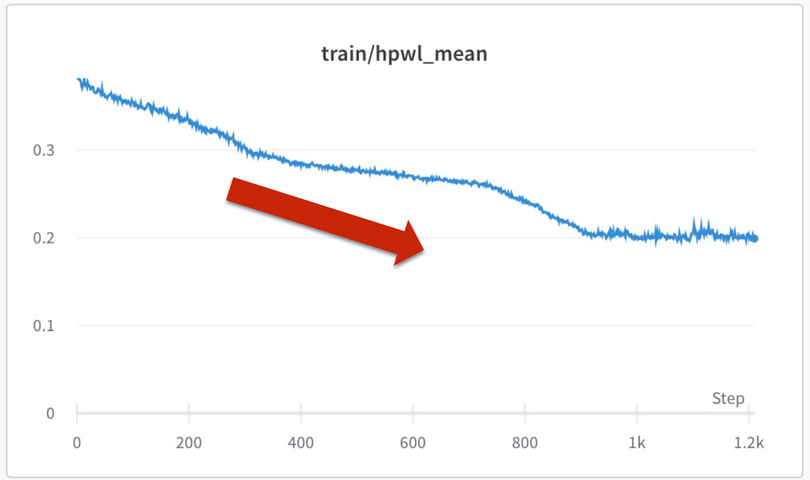
DEVIEW 2021

Training Progress

- HPWL을 줄이다가 학습 후반에 Congestion을 줄이는 경향성을 보임





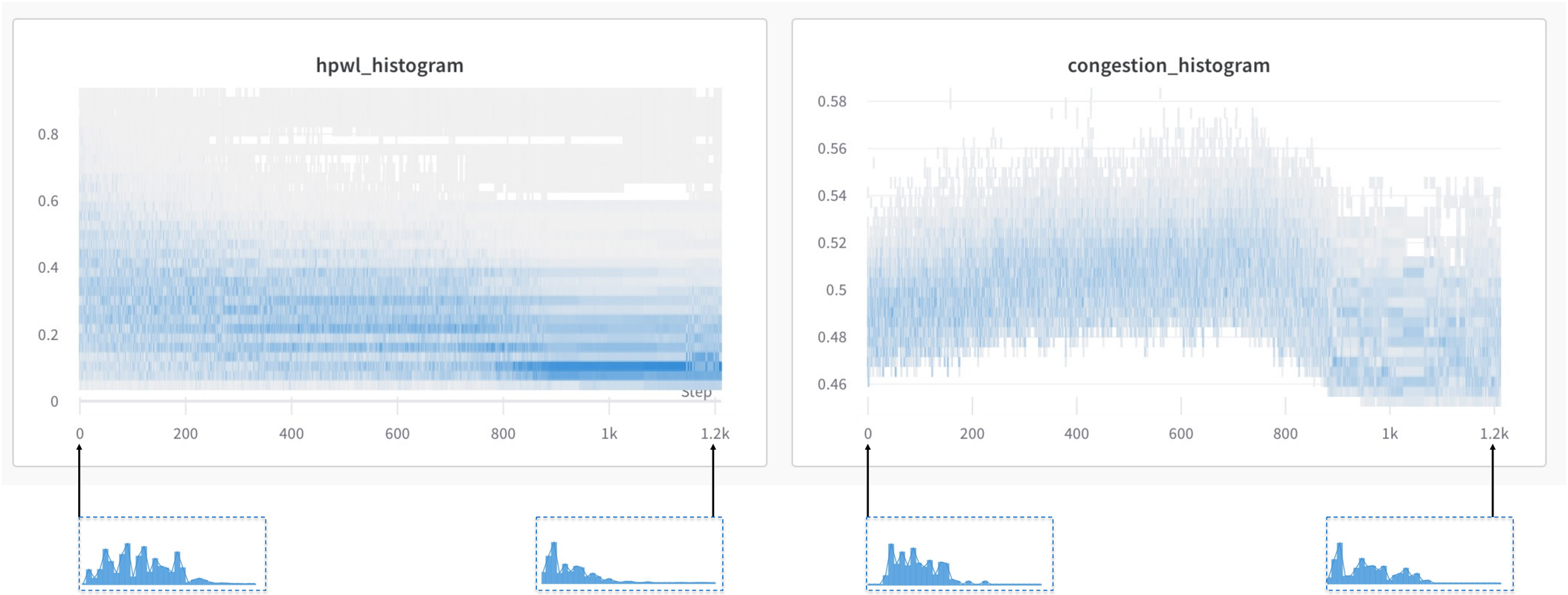


4.5 Experimental Results

DEVIEW 2021

Congestion & HPWL Shift

- 학습이 진행되며 Congestion과 HPWL의 분포가 낮은 쪽으로 이동

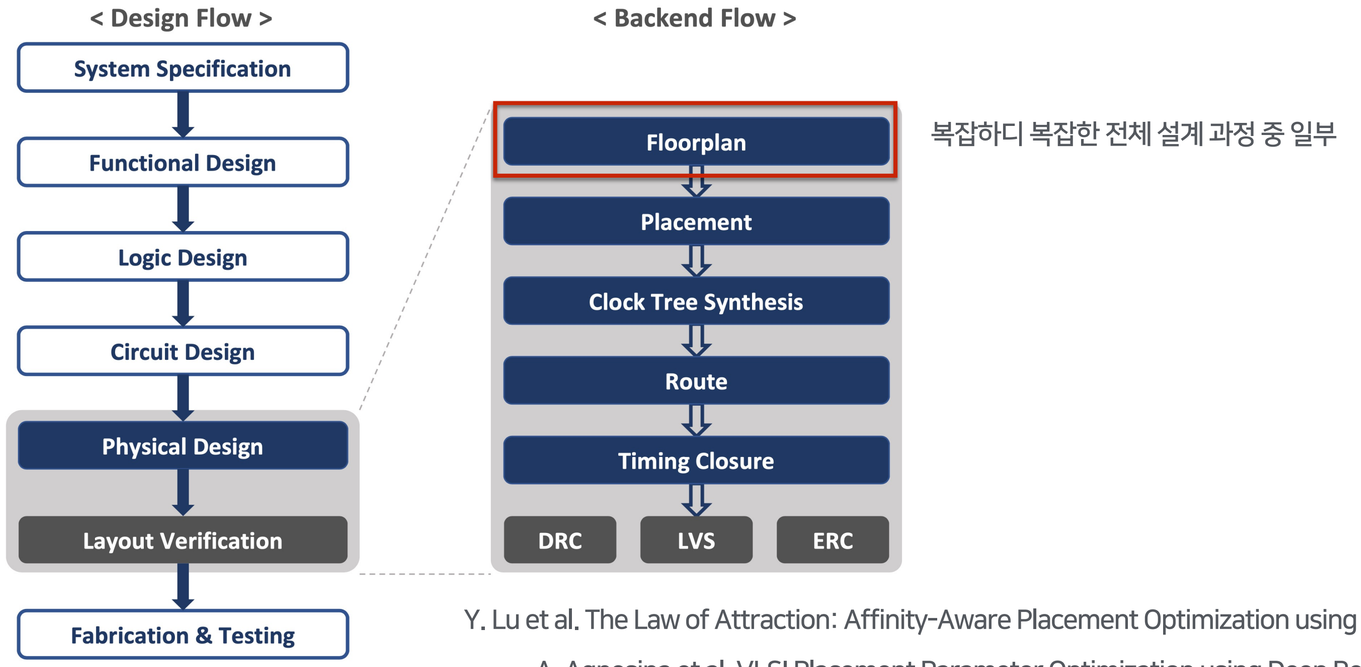


5. 맺음말

5.1 발표를 마치며

AI가 AI 반도체를 설계한다?!

- 반은 맞고 반은 틀리다
- 반도체 설계의 많은 과정 중, 사람에게 전적으로 의존적인 일부 과정을 자동화
- 앞으로도 ML 기술은 부분적인 개선을 위한 활용이 주를 이룰 것으로 보임



복잡하디 복잡한 전체 설계 과정 중 일부

Y. Lu et al. The Law of Attraction: Affinity-Aware Placement Optimization using Graph Neural Networks, ISPD 2021

A. Agnesina et al. VLSI Placement Parameter Optimization using Deep Reinforcement Learning, ICCAD 2020

5.2 차후 도전과제

Scale Up & Generalization

- 100 ~ 200개의 Macro Cell, 약 100만개의 Standard Cell로 구성된 Netlist 배치
- 다양한 Netlist에 대한 Generalization으로 학습에 대한 부담 축소
- Simulated Annealing / Genetic Algorithm과 같은 방법론으로 RL과 상호보완
- 확보한 기술을 다른 조합 최적화 문제에 확대 적용

Thank You