Graph Neural Networks as Ordering Heuristics for Parallel Graph Coloring

Fredrik Manne **Kenneth Langedal**

University of Bergen

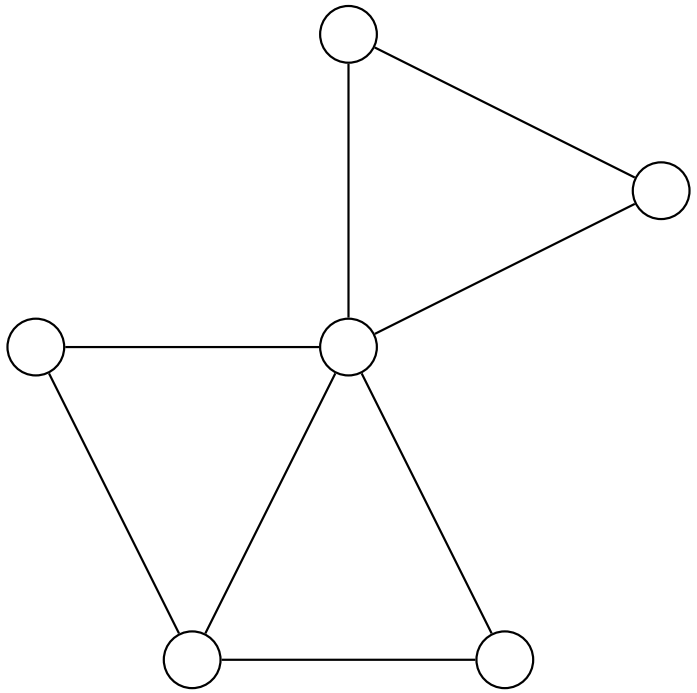
UNIVERSITY OF BERGEN

Problem

Graph Coloring

Input: Undirected graph $G = (V, E)$

Output: Assignment of colors $c : V \to \mathbb{Z}$ such that $\forall \{u, v\} \in E \implies c(u) \neq c(v)$ while minimizing the number of colors used

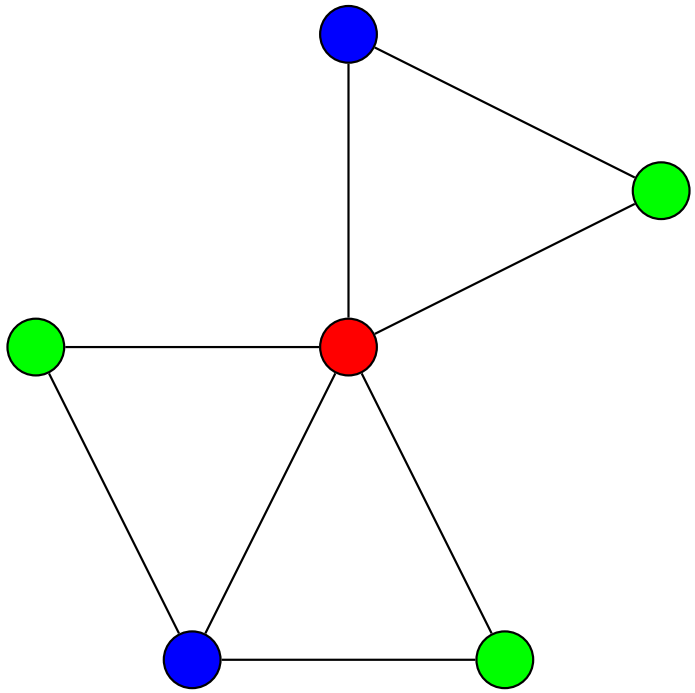


Problem

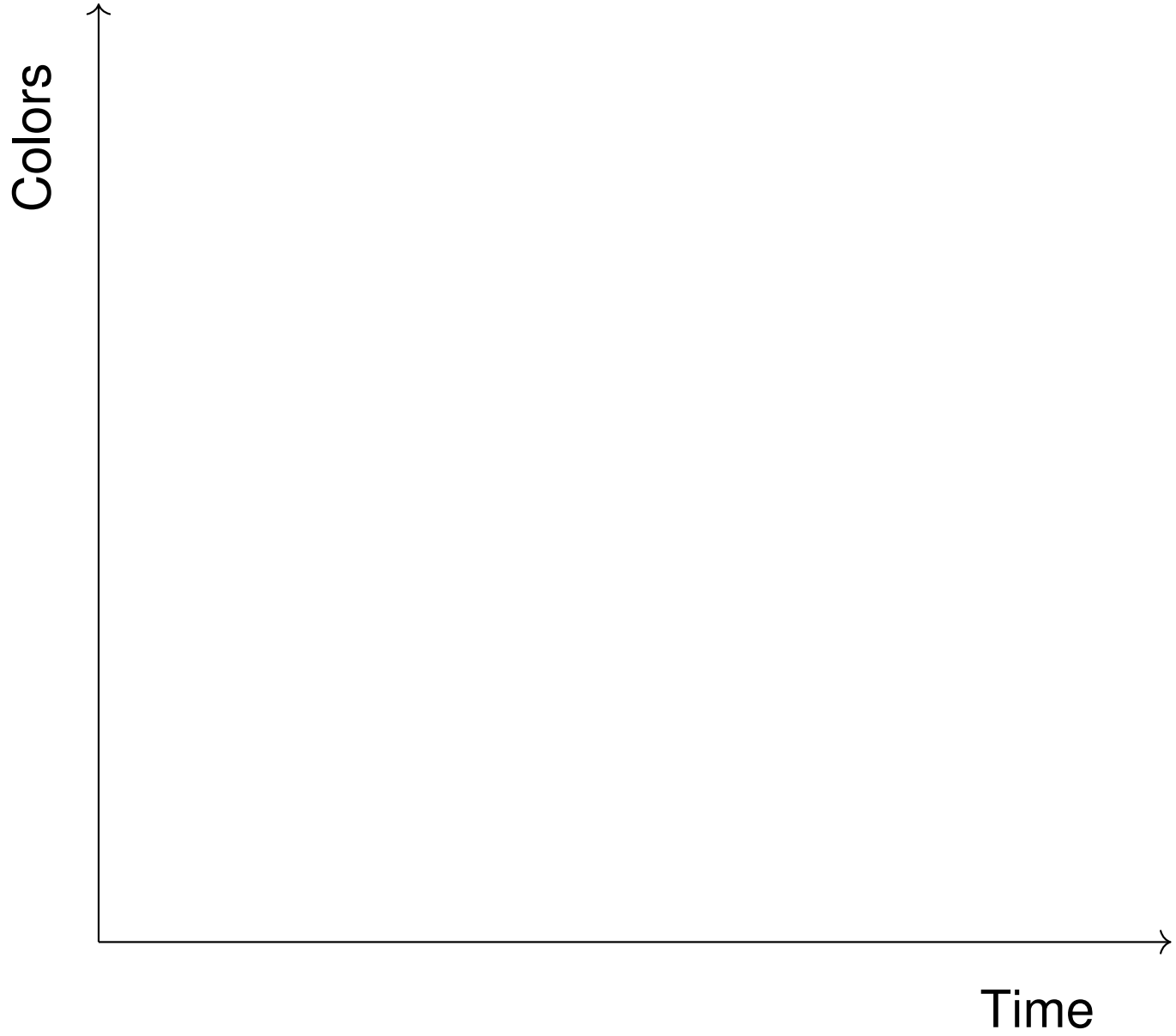
Graph Coloring

Input: Undirected graph $G = (V, E)$

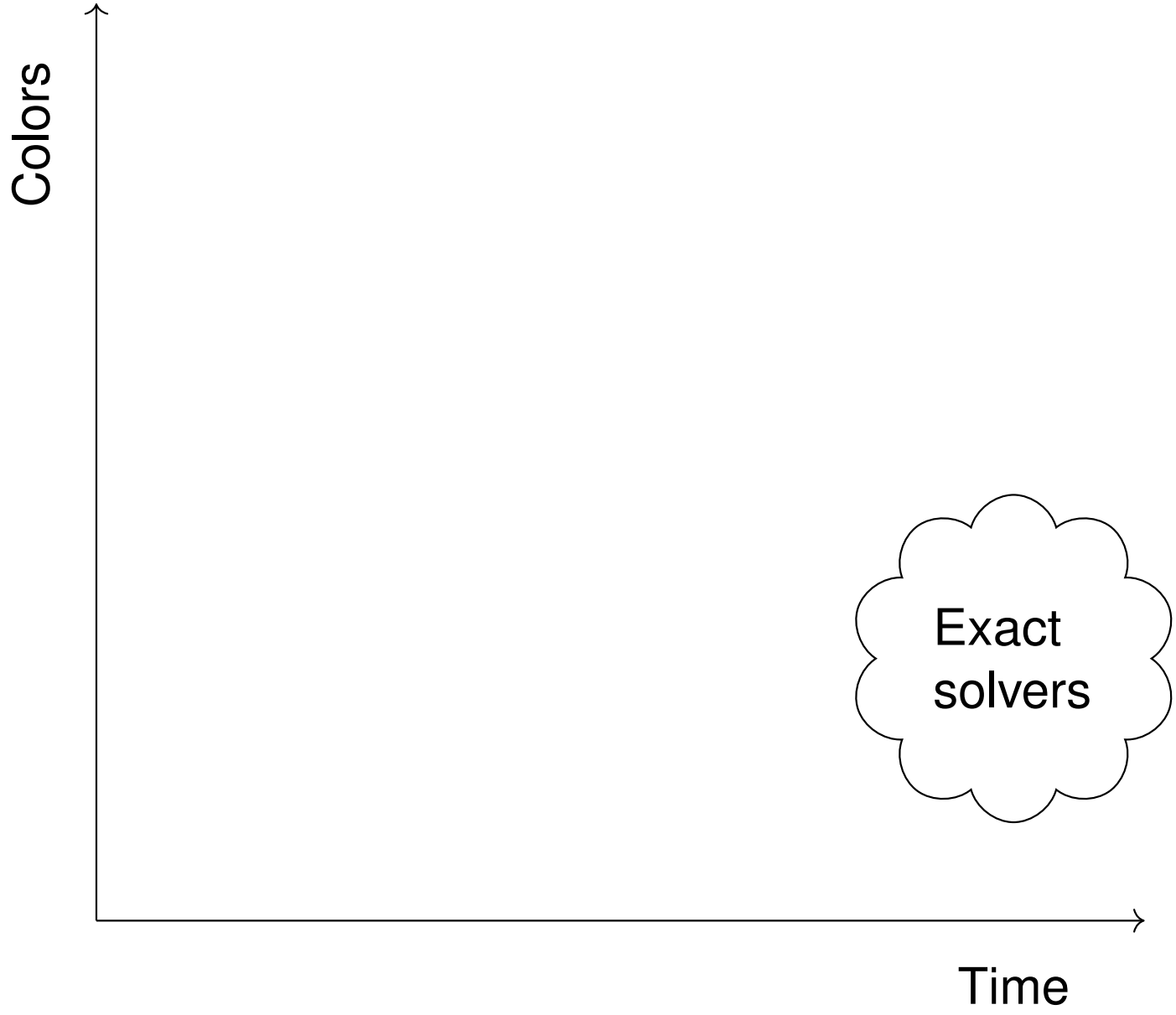
Output: Assignment of colors $c : V \to \mathbb{Z}$ such that $\forall \{u, v\} \in E \implies c(u) \neq c(v)$ while minimizing the number of colors used



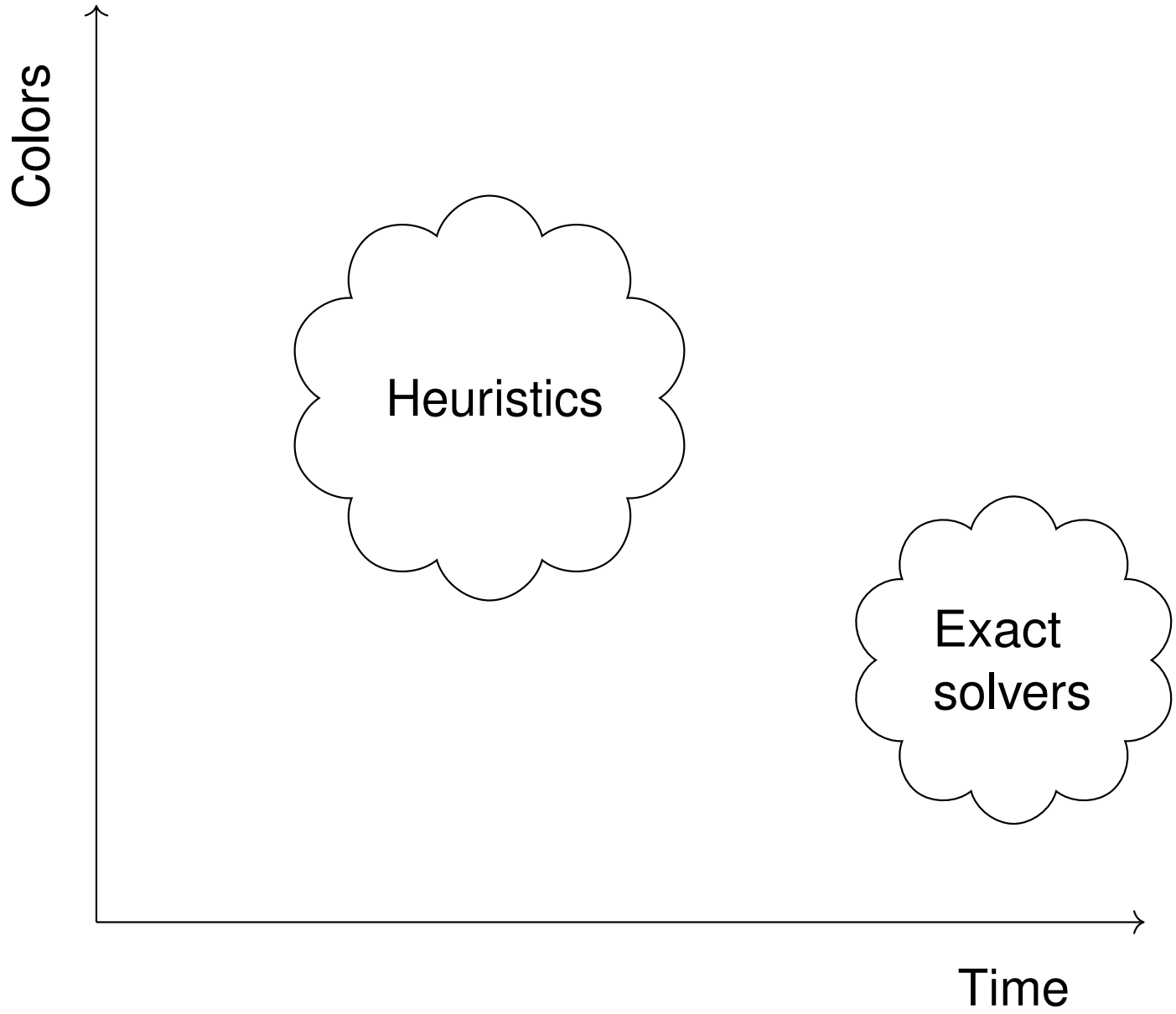
Overview



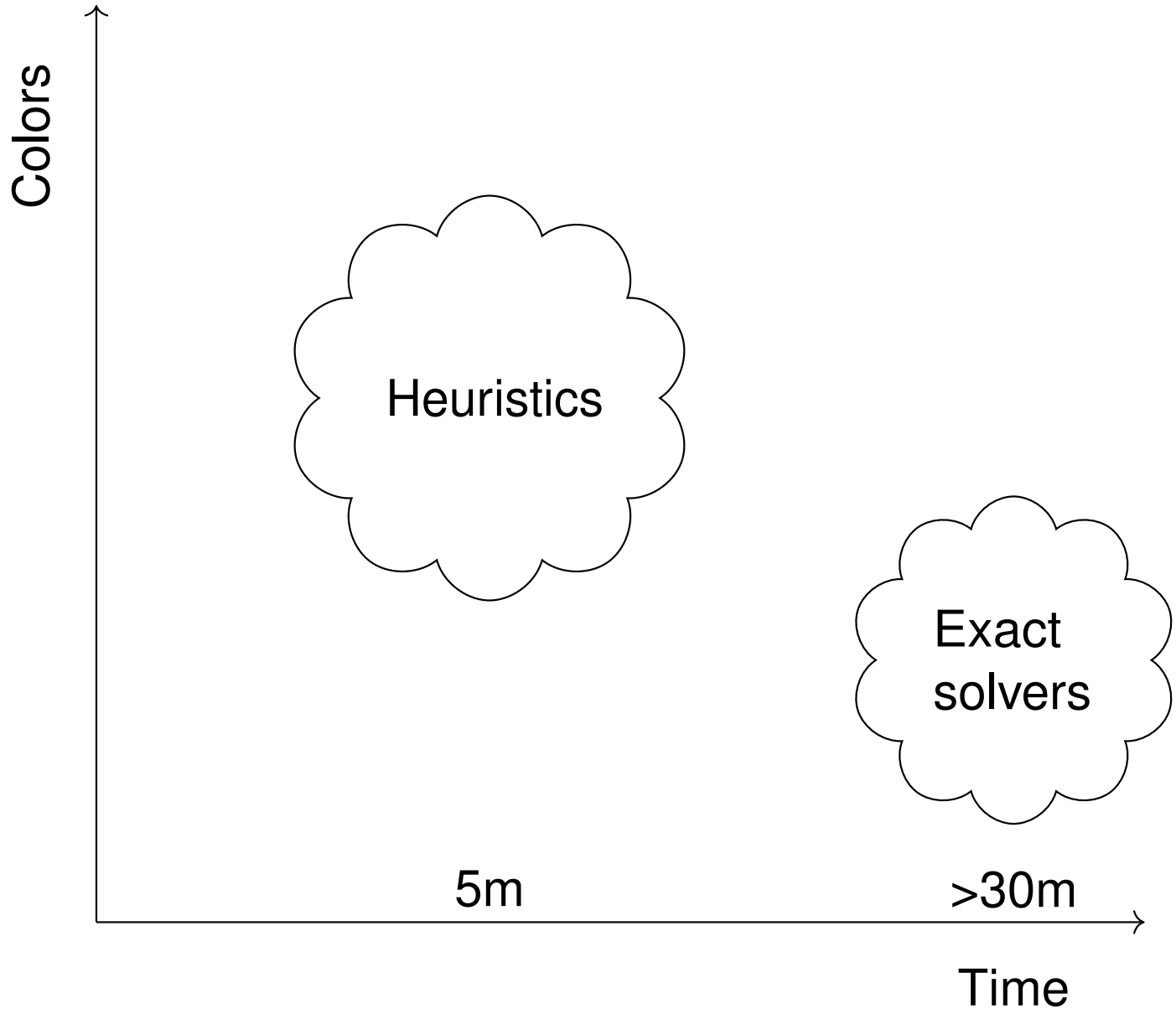
Overview



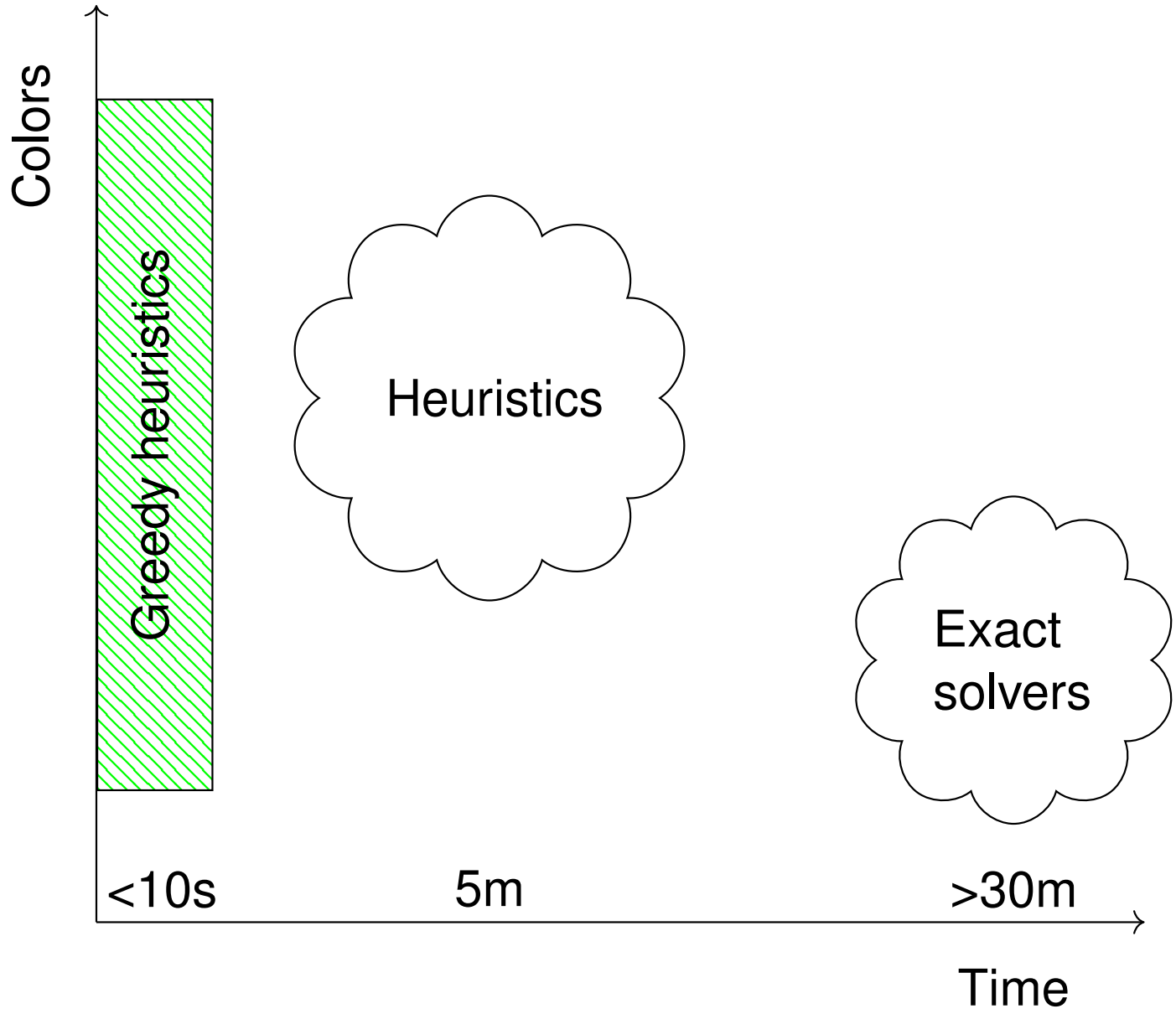
Overview



Overview



Overview


Colors
Greedy heuristics
Heuristics
Exact solvers
<10s
5m
>30m
Time

Ordering Heuristics

Ordering Heuristics

First Fit (FF)

Color the vertices in the order they appear in the input
(Welsh and Powell 1967)

Ordering Heuristics

First Fit (FF)

Color the vertices in the order they appear in the input
(Welsh and Powell 1967)

Largest Degree First (LF)

Color the vertices in decreasing order of degree
(Welsh and Powell 1967)

Ordering Heuristics

First Fit (FF)

Color the vertices in the order they appear in the input
(Welsh and Powell 1967)

Largest Degree First (LF)

Color the vertices in decreasing order of degree
(Welsh and Powell 1967)

Smallest Degree Last (SL)

Color the vertices in the reverse order induced by iteratively selecting and removing the smallest degree vertex from the graph
(Matula and Beck 1983)

Ordering Heuristics

Incidence Degree (ID)

Iteratively color the next uncolored vertex with the highest number of colored neighbors (Newsam and Ramsdell 1983)

Ordering Heuristics

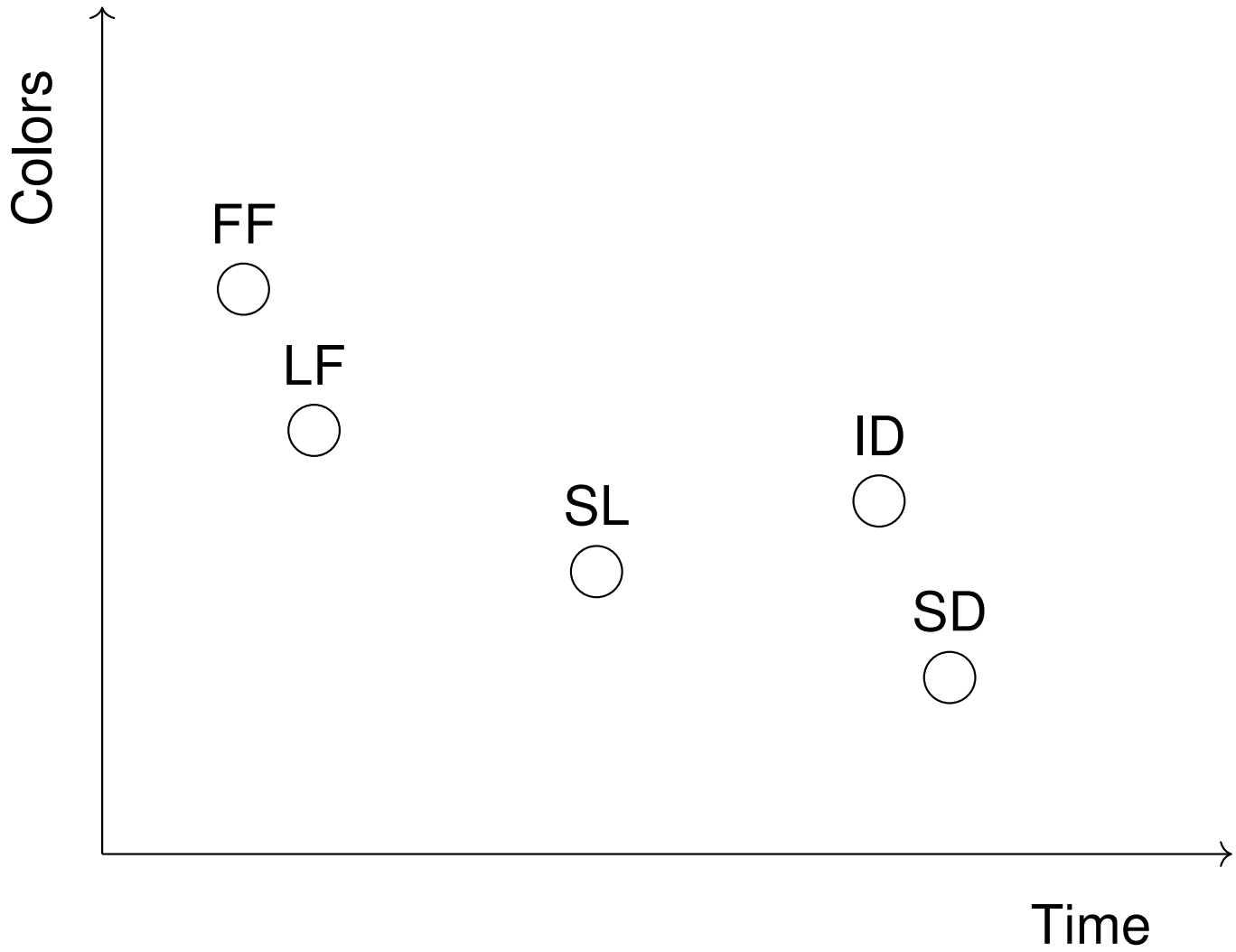
Incidence Degree (ID)

Iteratively color the next uncolored vertex with the highest number of colored neighbors (Newsam and Ramsdell 1983)

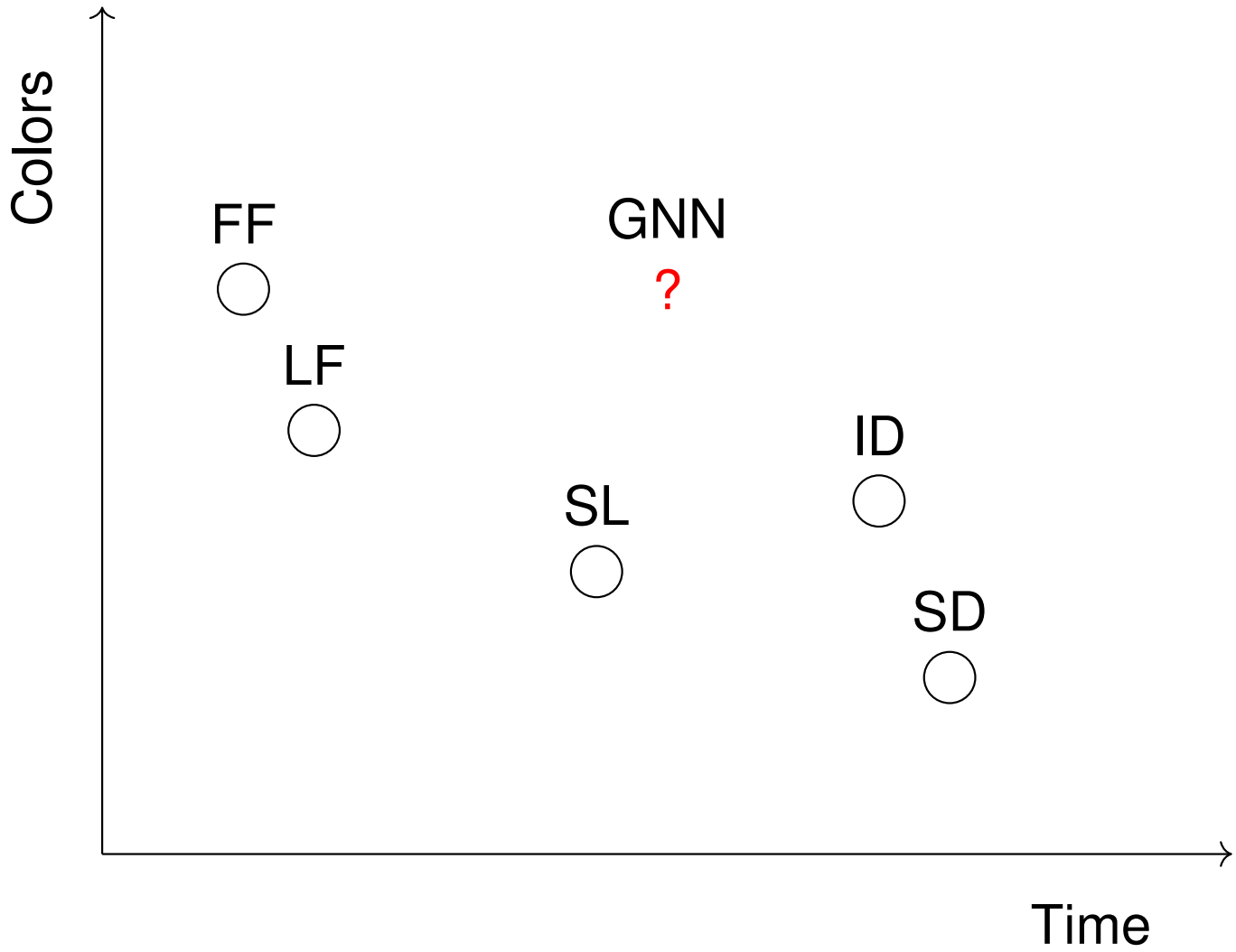
Saturation Degree (SD)

Iteratively color the next uncolored vertex with the highest number of distinct colors in its neighborhood (i.e. its saturation degree)

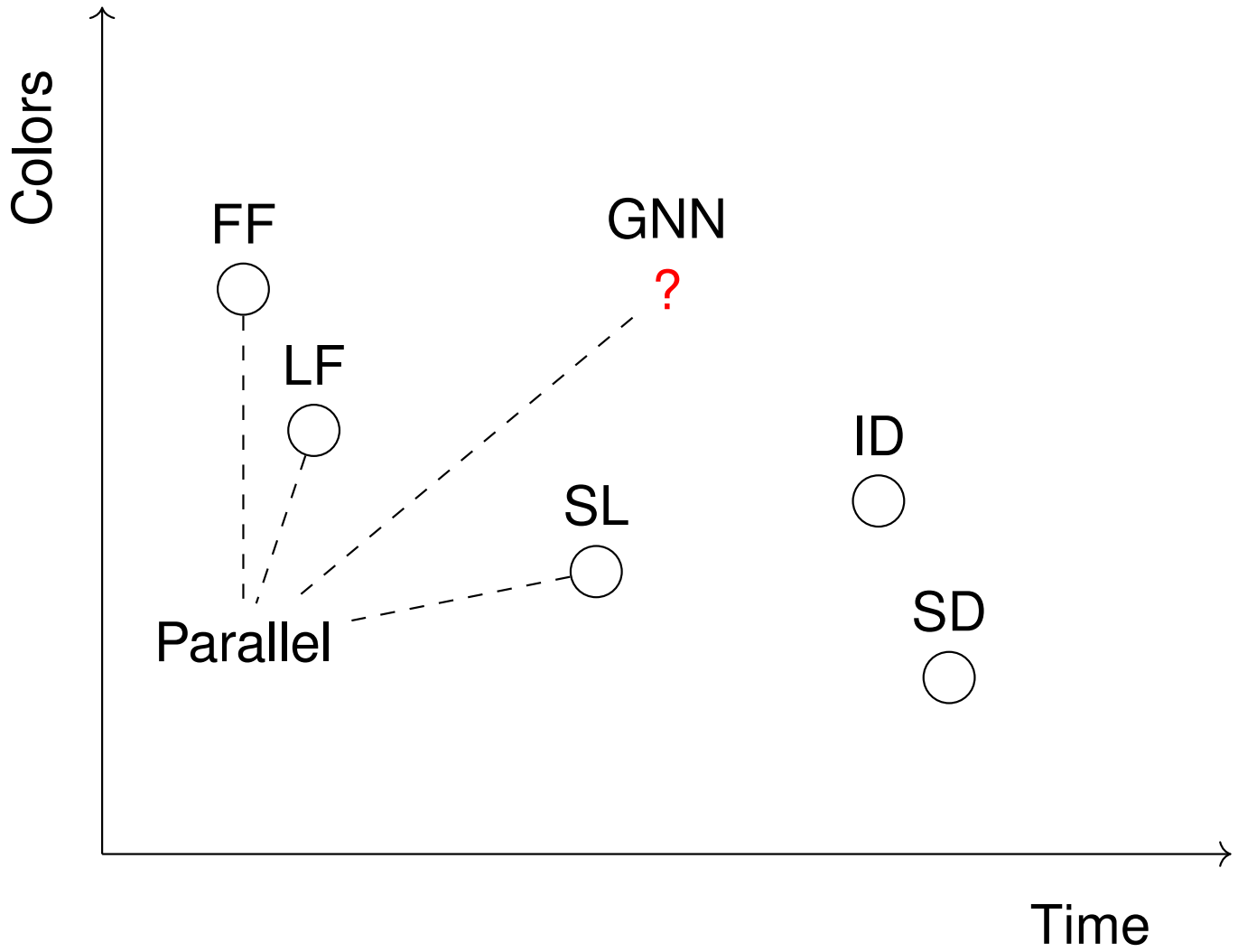
Ordering Heuristics



Ordering Heuristics



Ordering Heuristics



Parallel Coloring

- An ordering heuristic produces a permutation of the vertices
- Can be viewed as a partial ordering on V
- Can be achieved by a priority function $p : V \to \mathbb{R}$

Parallel Coloring

- An ordering heuristic produces a permutation of the vertices
- Can be viewed as a partial ordering on V
- Can be achieved by a priority function $p : V \to \mathbb{R}$

Two Stages

Parallel Coloring

- An ordering heuristic produces a permutation of the vertices
- Can be viewed as a partial ordering on V
- Can be achieved by a priority function $p : V \to \mathbb{R}$

Two Stages

Create Ordering

Give priority function p using LF, SL, or a GNN

Parallel Coloring

- An ordering heuristic produces a permutation of the vertices
- Can be viewed as a partial ordering on V
- Can be achieved by a priority function $p : V \to \mathbb{R}$

Two Stages

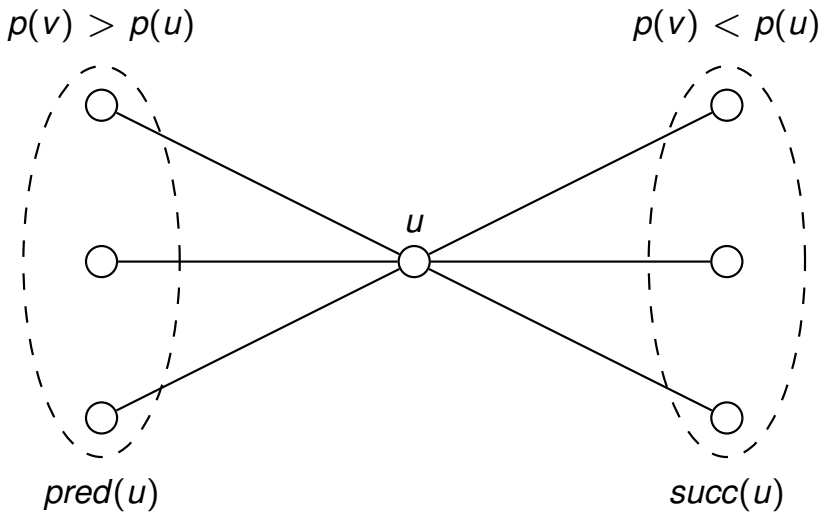
Create Ordering	Color the Graph
Give priority function p using LF, SL, or a GNN	Follow p and color vertices using a greedy approach

Parallel coloring: Jones-Plassmann

1: $next \leftarrow \emptyset, prev \leftarrow \emptyset$
2: **parallel for** $u \in V$
3: $\quad pred(u) \leftarrow v \in N(u) : p(v) > p(u)$
4: $\quad succ(u) \leftarrow v \in N(u) : p(u) > p(v)$
5: $\quad count(u) \leftarrow |pred(u)|$
6: $\quad$ **if** $count(u) = 0$
7: $\quad\quad$ **atomic** $prev \leftarrow prev + \{u\}$

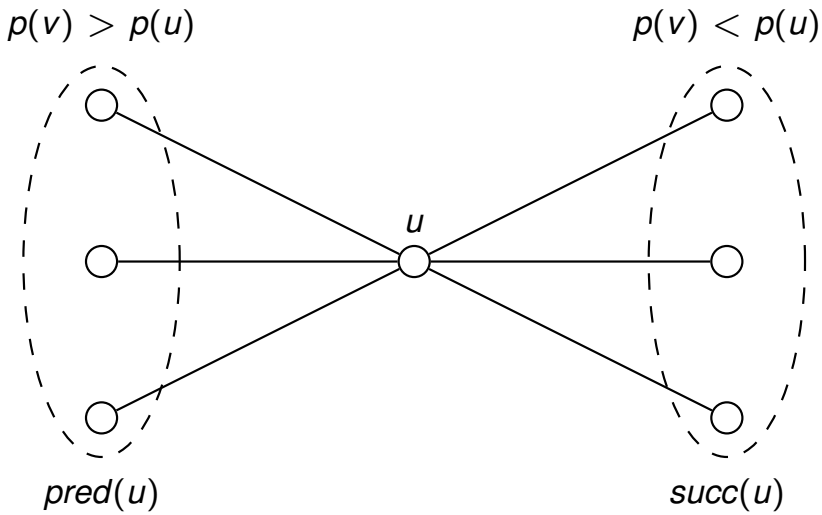
Parallel coloring: Jones-Plassmann

1: $next \leftarrow \emptyset, prev \leftarrow \emptyset$
2: **parallel for** $u \in V$
3: $\quad pred(u) \leftarrow v \in N(u) : p(v) > p(u)$
4: $\quad succ(u) \leftarrow v \in N(u) : p(u) > p(v)$
5: $\quad count(u) \leftarrow |pred(u)|$
6: $\quad$ **if** $count(u) = 0$
7: $\quad\quad$ **atomic** $prev \leftarrow prev + \{u\}$



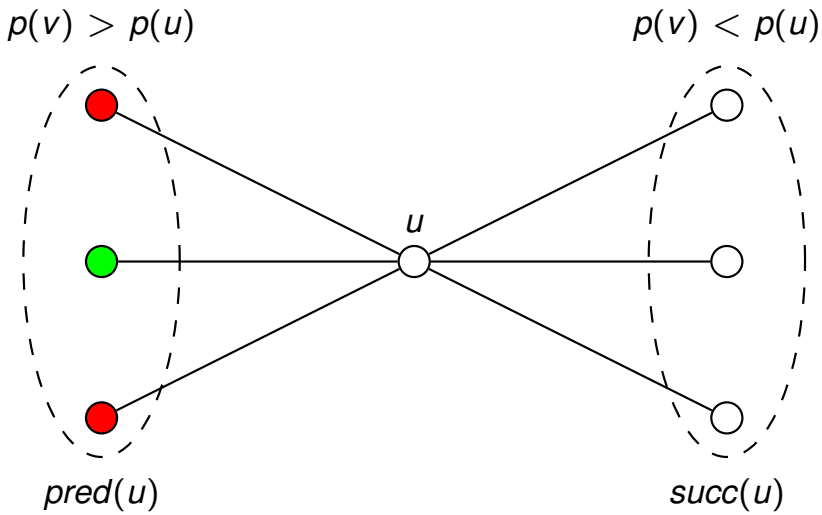
Parallel coloring: Jones-Plassmann

1: $next \leftarrow \emptyset, prev \leftarrow \emptyset$
2: **parallel for** $u \in V$
3: $pred(u) \leftarrow v \in N(u) : p(v) > p(u)$
4: $succ(u) \leftarrow v \in N(u) : p(u) > p(v)$
5: $count(u) \leftarrow |pred(u)|$
6: **if** $count(u) = 0$
7: **atomic** $prev \leftarrow prev + \{u\}$
8: **while** $prev \neq \emptyset$
9: **parallel for** $u \in prev$
10: $c(u) \leftarrow$ min unused color among $pred(u)$
11: **for** $v \in succ(u)$
12: **atomic** $count(v) \leftarrow count(v) - 1$
13: **if** $count(v) = 0$
14: **atomic** $next \leftarrow next + \{v\}$
15: $swap(next, prev)$



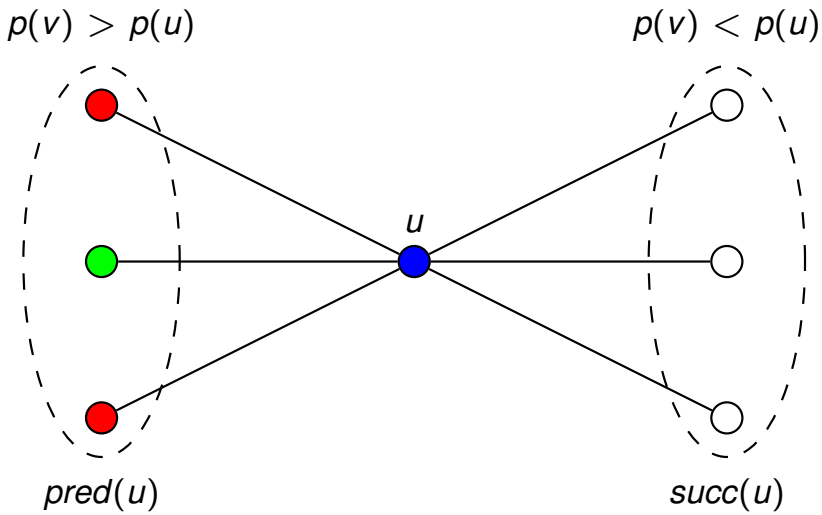
Parallel coloring: Jones-Plassmann

1: $next \leftarrow \emptyset, prev \leftarrow \emptyset$
2: **parallel for** $u \in V$
3: $\quad pred(u) \leftarrow v \in N(u) : p(v) > p(u)$
4: $\quad succ(u) \leftarrow v \in N(u) : p(u) > p(v)$
5: $\quad count(u) \leftarrow |pred(u)|$
6: $\quad$ **if** $count(u) = 0$
7: $\quad\quad$ **atomic** $prev \leftarrow prev + \{u\}$
8: **while** $prev \neq \emptyset$
9: $\quad$ **parallel for** $u \in prev$
10: $\quad\quad c(u) \leftarrow$ min unused color among $pred(u)$
11: $\quad\quad$ **for** $v \in succ(u)$
12: $\quad\quad\quad$ **atomic** $count(v) \leftarrow count(v) - 1$
13: $\quad\quad\quad$ **if** $count(v) = 0$
14: $\quad\quad\quad\quad$ **atomic** $next \leftarrow next + \{v\}$
15: $\quad swap(next, prev)$



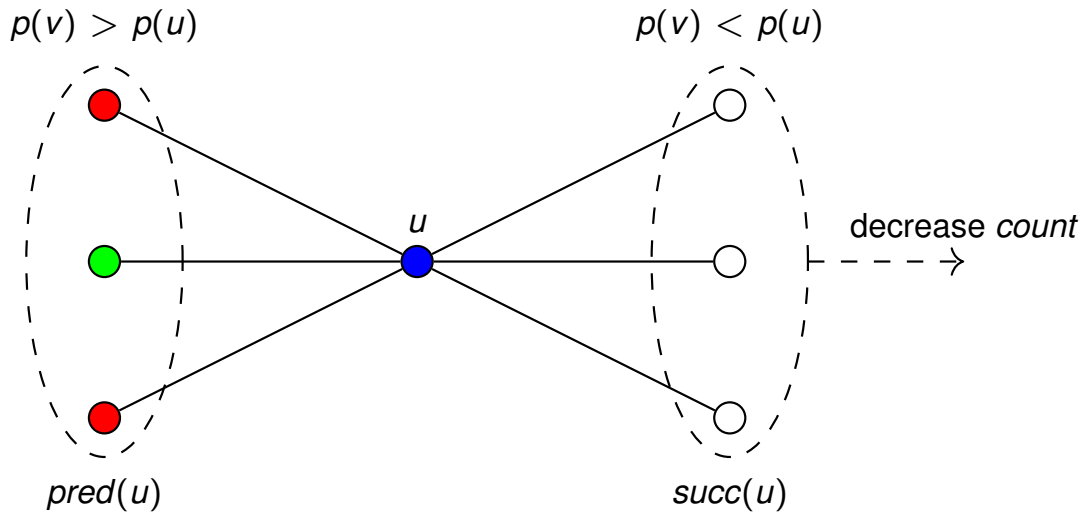
Parallel coloring: Jones-Plassmann

1: $next \leftarrow \emptyset, prev \leftarrow \emptyset$
2: **parallel for** $u \in V$
3: $\quad pred(u) \leftarrow v \in N(u) : p(v) > p(u)$
4: $\quad succ(u) \leftarrow v \in N(u) : p(u) > p(v)$
5: $\quad count(u) \leftarrow |pred(u)|$
6: $\quad$ **if** $count(u) = 0$
7: $\quad\quad$ **atomic** $prev \leftarrow prev + \{u\}$
8: **while** $prev \neq \emptyset$
9: $\quad$ **parallel for** $u \in prev$
10: $\quad\quad c(u) \leftarrow$ min unused color among $pred(u)$
11: $\quad\quad$ **for** $v \in succ(u)$
12: $\quad\quad\quad$ **atomic** $count(v) \leftarrow count(v) - 1$
13: $\quad\quad\quad$ **if** $count(v) = 0$
14: $\quad\quad\quad\quad$ **atomic** $next \leftarrow next + \{v\}$
15: $\quad swap(next, prev)$



Parallel coloring: Jones-Plassmann

1: $next \leftarrow \emptyset, prev \leftarrow \emptyset$
2: **parallel for** $u \in V$
3: $\quad pred(u) \leftarrow v \in N(u) : p(v) > p(u)$
4: $\quad succ(u) \leftarrow v \in N(u) : p(u) > p(v)$
5: $\quad count(u) \leftarrow |pred(u)|$
6: $\quad$ **if** $count(u) = 0$
7: $\quad\quad$ **atomic** $prev \leftarrow prev + \{u\}$
8: **while** $prev \neq \emptyset$
9: $\quad$ **parallel for** $u \in prev$
10: $\quad\quad c(u) \leftarrow$ min unused color among $pred(u)$
11: $\quad\quad$ **for** $v \in succ(u)$
12: $\quad\quad\quad$ **atomic** $count(v) \leftarrow count(v) - 1$
13: $\quad\quad\quad$ **if** $count(v) = 0$
14: $\quad\quad\quad\quad$ **atomic** $next \leftarrow next + \{v\}$
15: $\quad swap(next, prev)$

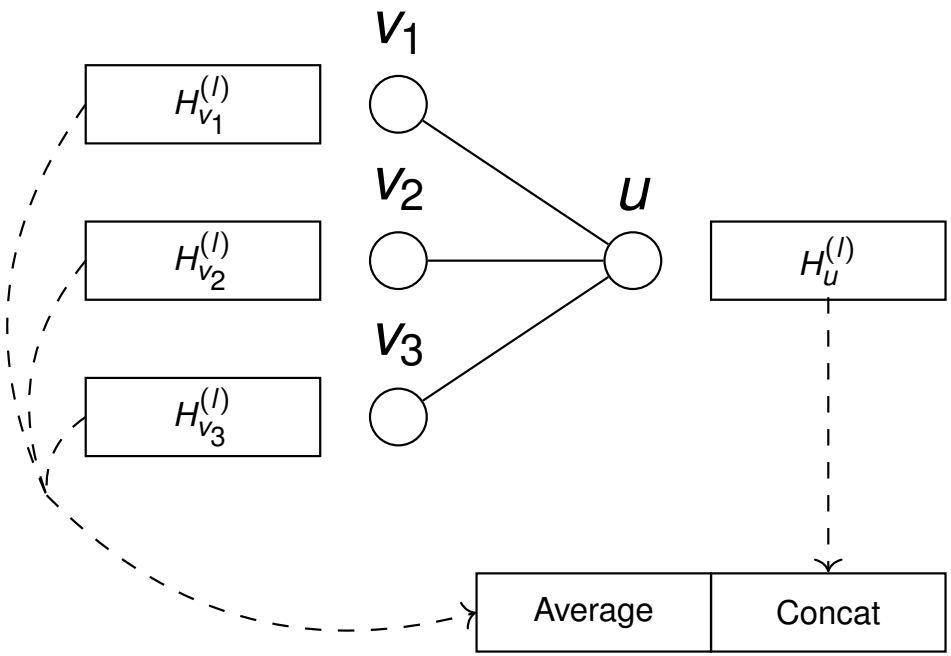


The Jones-Plassmann Algorithm 1993

- $\mathcal{O}(V + E)$ running time
- $\mathcal{O}(\log(V)/\log\log(V))$ expected parallel running time on bounded degree graphs
- $\Omega(V)$ worst case

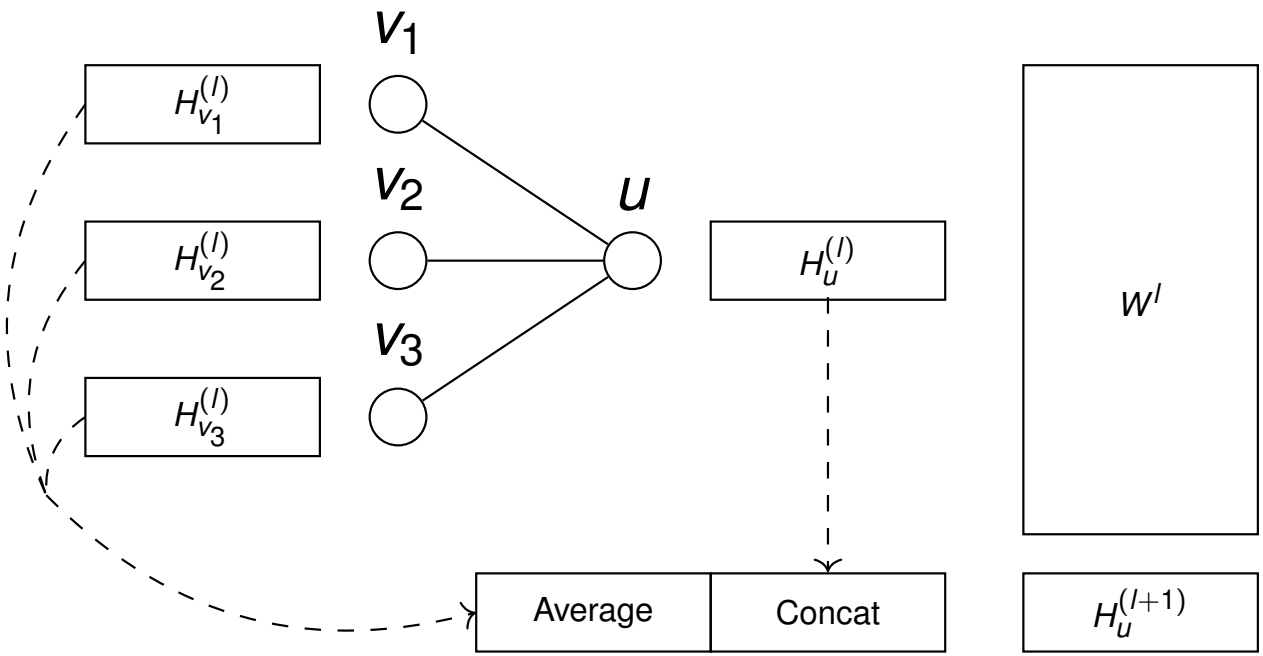
Graph Neural Networks

- GraphSAGE inspired architecture (Hamilton, Ying and Leskovec 2017)
- No feature engineering, only degree and node-ID



Graph Neural Networks

- GraphSAGE inspired architecture (Hamilton, Ying and Leskovec 2017)
- No feature engineering, only degree and node-ID



Implementation

Instance	$\|V\|$	$\|E\|$	FF	LF	SL	SD
com-Youtube	1,134,890	5,975,248	0.131	0.151	0.218	1.482
web-Google	916,428	8,644,102	0.163	0.218	0.391	1.739
wiki-Talk	2,394,385	9,319,130	0.269	0.276	0.381	4.173

Implementation

Instance	$\|V\|$	$\|E\|$	FF	LF	SL	SD
com-Youtube	1,134,890	5,975,248	0.131	0.151	0.218	1.482
web-Google	916,428	8,644,102	0.163	0.218	0.391	1.739
wiki-Talk	2,394,385	9,319,130	0.269	0.276	0.381	4.173

- PyTorch, 16 hidden features, 3 layers

Implementation

Instance	FF	LF	SL	**PyT**	SD
com-Youtube	0.131	0.151	0.218	**0.847**	1.482
web-Google	0.163	0.218	0.391	**1.176**	1.739
wiki-Talk	0.269	0.276	0.381	**1.521**	4.173

- PyTorch, 16 hidden features, 3 layers

Implementation

- PyTorch, 16 hidden features, 3 layers
- Manual message passing and OpenBLAS

Implementation

Instance	FF	LF	SL	**OB**	**PyT**	SD
com-Youtube	0.131	0.151	0.218	**0.553**	**0.847**	1.482
web-Google	0.163	0.218	0.391	**0.764**	**1.176**	1.739
wiki-Talk	0.269	0.276	0.381	**0.961**	**1.521**	4.173

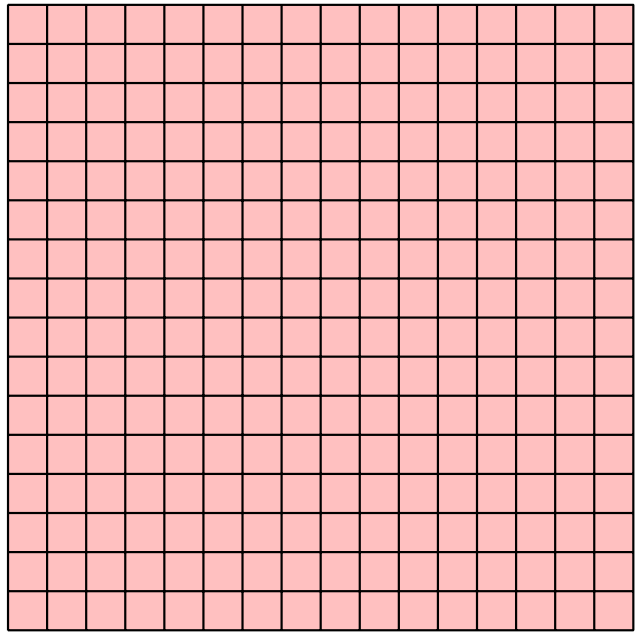
- PyTorch, 16 hidden features, 3 layers
- Manual message passing and OpenBLAS

Implementation

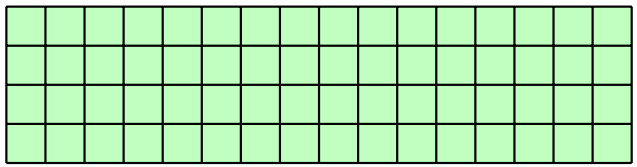
Instance	FF	LF	SL	**OB**	**PyT**	SD
com-Youtube	0.131	0.151	0.218	**0.553**	**0.847**	1.482
web-Google	0.163	0.218	0.391	**0.764**	**1.176**	1.739
wiki-Talk	0.269	0.276	0.381	**0.961**	**1.521**	4.173

- PyTorch, 16 hidden features, 3 layers
- Manual message passing and OpenBLAS
- Can we do better?

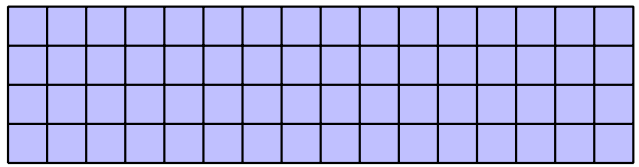
Manual Implementation - Kernel



$W^{(l)}$

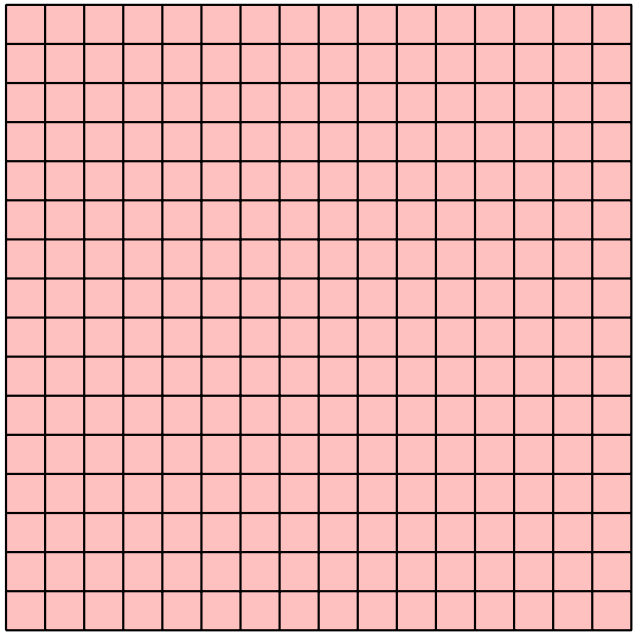


$H^{(l)}$

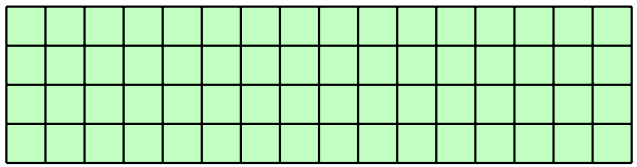


$H^{(l+1)}$

Manual Implementation - Kernel



$W^{(l)}$



$H^{(l)}$

ymm0	ymm1
ymm2	ymm3
ymm4	ymm5
ymm6	ymm7

$H^{(l+1)}$

Manual Implementation - Kernel

- EPYC 7302P 16-core processor

Manual Implementation - Kernel

- EPYC 7302P 16-core processor
- $\approx$ 100 GFLOP/s compute and $\approx$ 25 GB/s bandwidth per core

Manual Implementation - Kernel

- EPYC 7302P 16-core processor
- $\approx$ 100 GFLOP/s compute and $\approx$ 25 GB/s bandwidth per core
- Compute work: $16 * 2 * N * 16 = 512N$

Manual Implementation - Kernel

- EPYC 7302P 16-core processor
- $\approx$ 100 GFLOP/s compute and $\approx$ 25 GB/s bandwidth per core
- Compute work: $16 * 2 * N * 16 = 512N$
- Amount of data: $4 * N * 16 * 3 = 192N$

Manual Implementation - Kernel

- EPYC 7302P 16-core processor
- $\approx$ 100 GFLOP/s compute and $\approx$ 25 GB/s bandwidth per core
- Compute work: $16 * 2 * N * 16 = 512N$
- Amount of data: $4 * N * 16 * 3 = 192N$
- $1 : 2.67$

Manual Implementation - Kernel

- EPYC 7302P 16-core processor
- $\approx$ 100 GFLOP/s compute and $\approx$ 25 GB/s bandwidth per core
- Compute work: $16 * 2 * N * 16 = 512N$
- Amount of data: $4 * N * 16 * 3 = 192N$
- $1 : 2.67$
- Non-temporal write

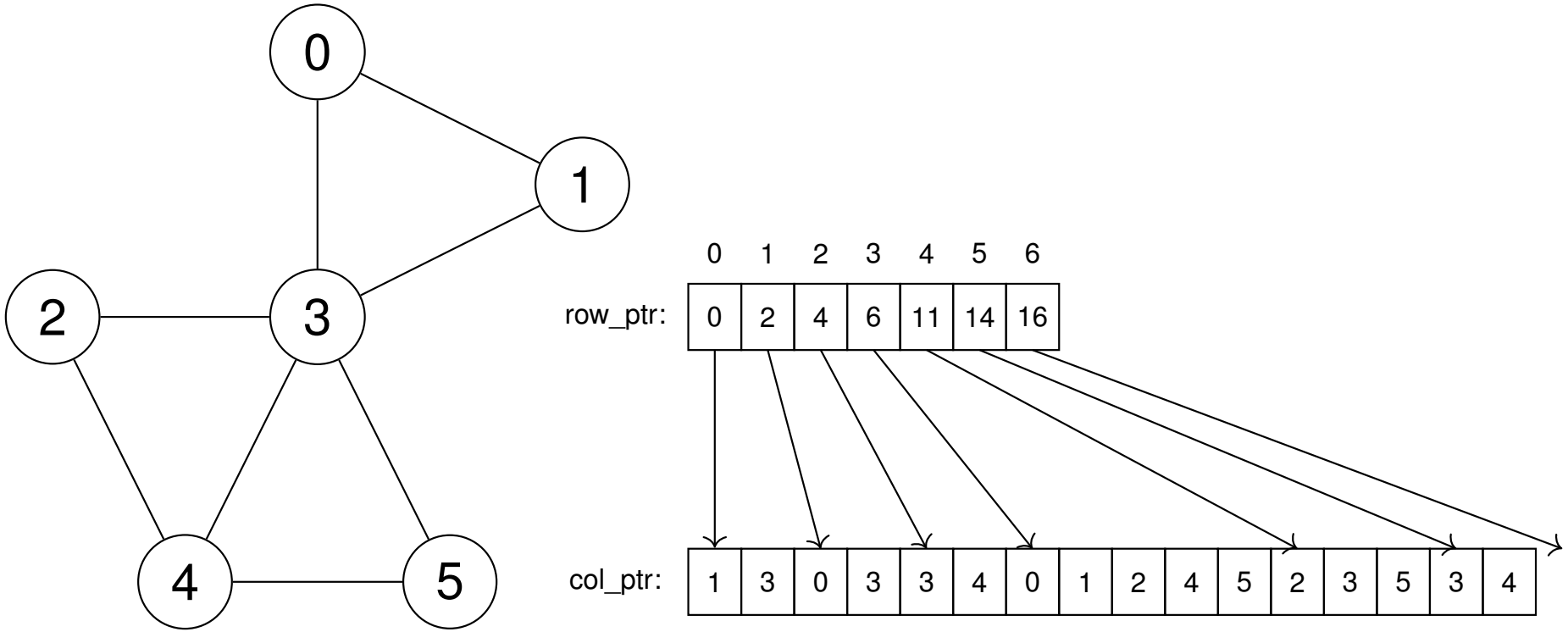
Manual Implementation - Kernel

- EPYC 7302P 16-core processor
- $\approx$ 100 GFLOP/s compute and $\approx$ 25 GB/s bandwidth per core
- Compute work: $16 * 2 * N * 16 = 512N$
- Amount of data: $4 * N * 16 * 3 = 192N$
- $1 : 2.67$
- Non-temporal write
- Amount of data: $4 * N * 16 * 2 = 128N$

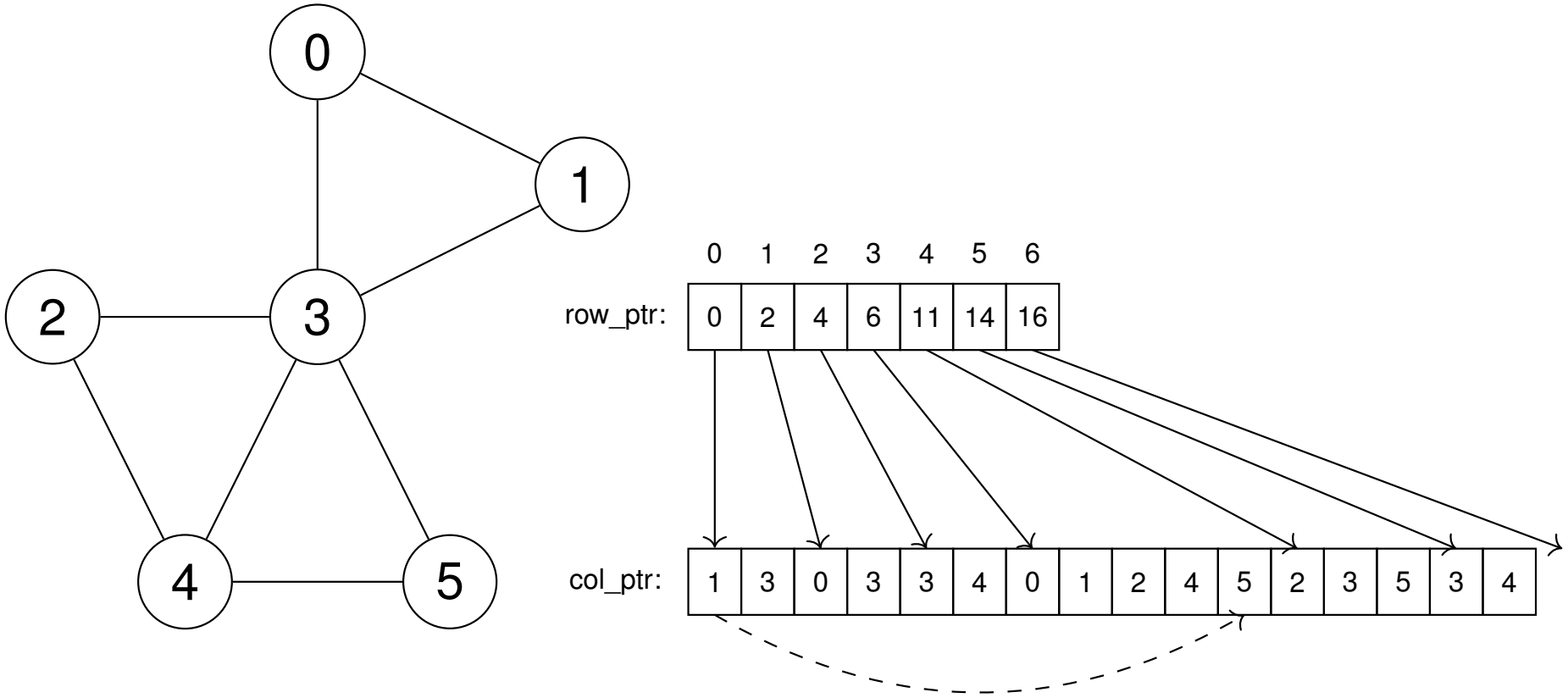
Manual Implementation - Kernel

- EPYC 7302P 16-core processor
- $\approx$ 100 GFLOP/s compute and $\approx$ 25 GB/s bandwidth per core
- Compute work: $16 * 2 * N * 16 = 512N$
- Amount of data: $4 * N * 16 * 3 = 192N$
- 1 : 2.67
- Non-temporal write
- Amount of data: $4 * N * 16 * 2 = 128N$
- 1 : 4

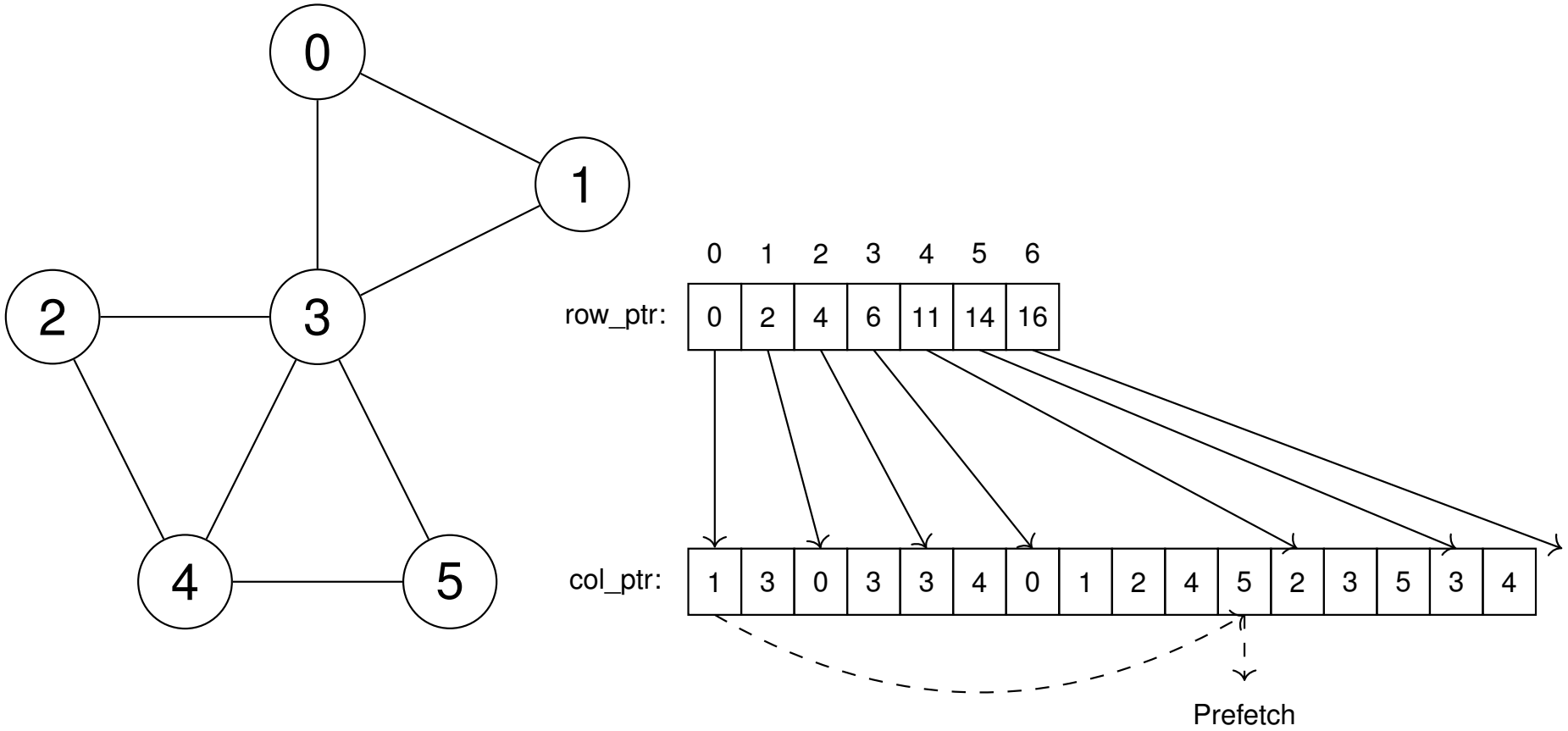
Manual Implementation - Compressed Sparse Row



Manual Implementation - Compressed Sparse Row



Manual Implementation - Compressed Sparse Row



Manual Implementation - Overlapping Message and Transform

- Perform the message passing for a fixed number of vertices and store these rows in a buffer

Manual Implementation - Overlapping Message and Transform

- Perform the message passing for a fixed number of vertices and store these rows in a buffer
- When full, perform the matrix multiplication and store the corresponding rows of $H^{(l+1)}$.

Manual Implementation - Overlapping Message and Transform

- Perform the message passing for a fixed number of vertices and store these rows in a buffer
- When full, perform the matrix multiplication and store the corresponding rows of $H^{(l+1)}$.
- Parameters will fit in cache.

Manual Implementation - Overlapping Message and Transform

- Perform the message passing for a fixed number of vertices and store these rows in a buffer
- When full, perform the matrix multiplication and store the corresponding rows of $H^{(l+1)}$.
- Parameters will fit in cache.
- Use prefetching to hide memory latency resulting from the irregular access in message-passing

Implementation

Instance	FF	**C-GNN**	LF	SL	**OB**	**PyT**	SD
com-Youtube	0.131	**0.123**	0.151	0.218	**0.553**	**0.847**	1.482
web-Google	0.163	**0.162**	0.218	0.391	**0.764**	**1.176**	1.739
wiki-Talk	0.269	**0.238**	0.276	0.381	**0.961**	**1.521**	4.173

Supervised Training

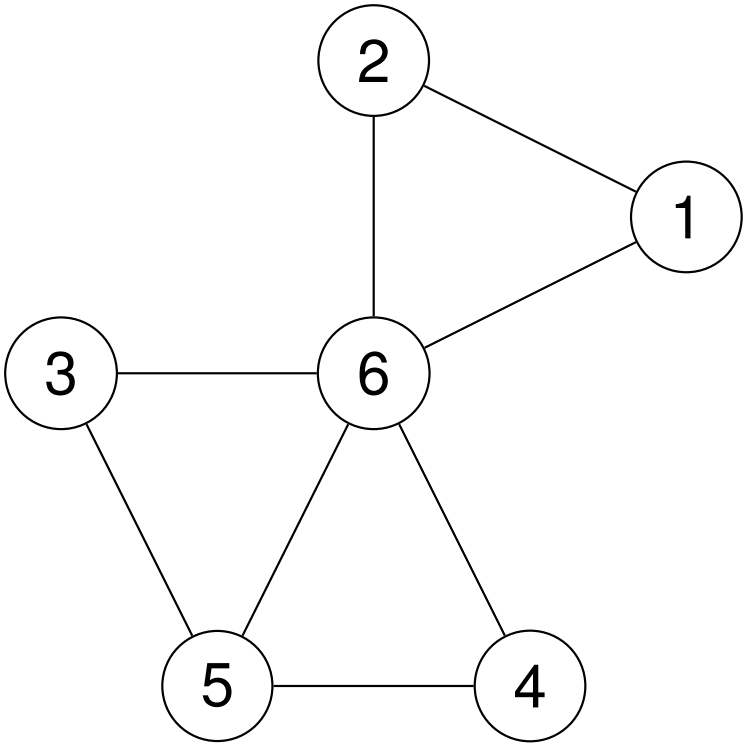
- Supervised learning on SL and SD heuristics

Supervised Training

- Supervised learning on SL and SD heuristics
- How to learn ordering?

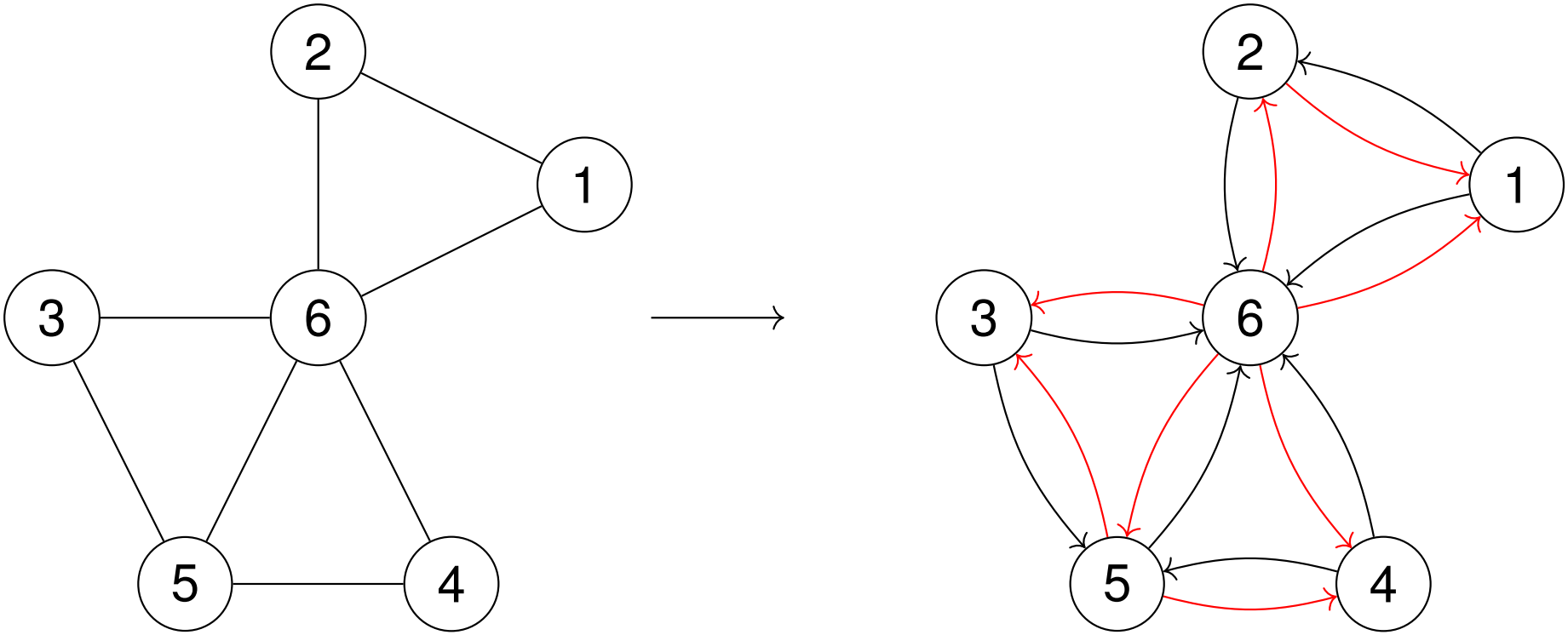
Supervised Training

- Supervised learning on SL and SD heuristics
- How to learn ordering?



Supervised Training

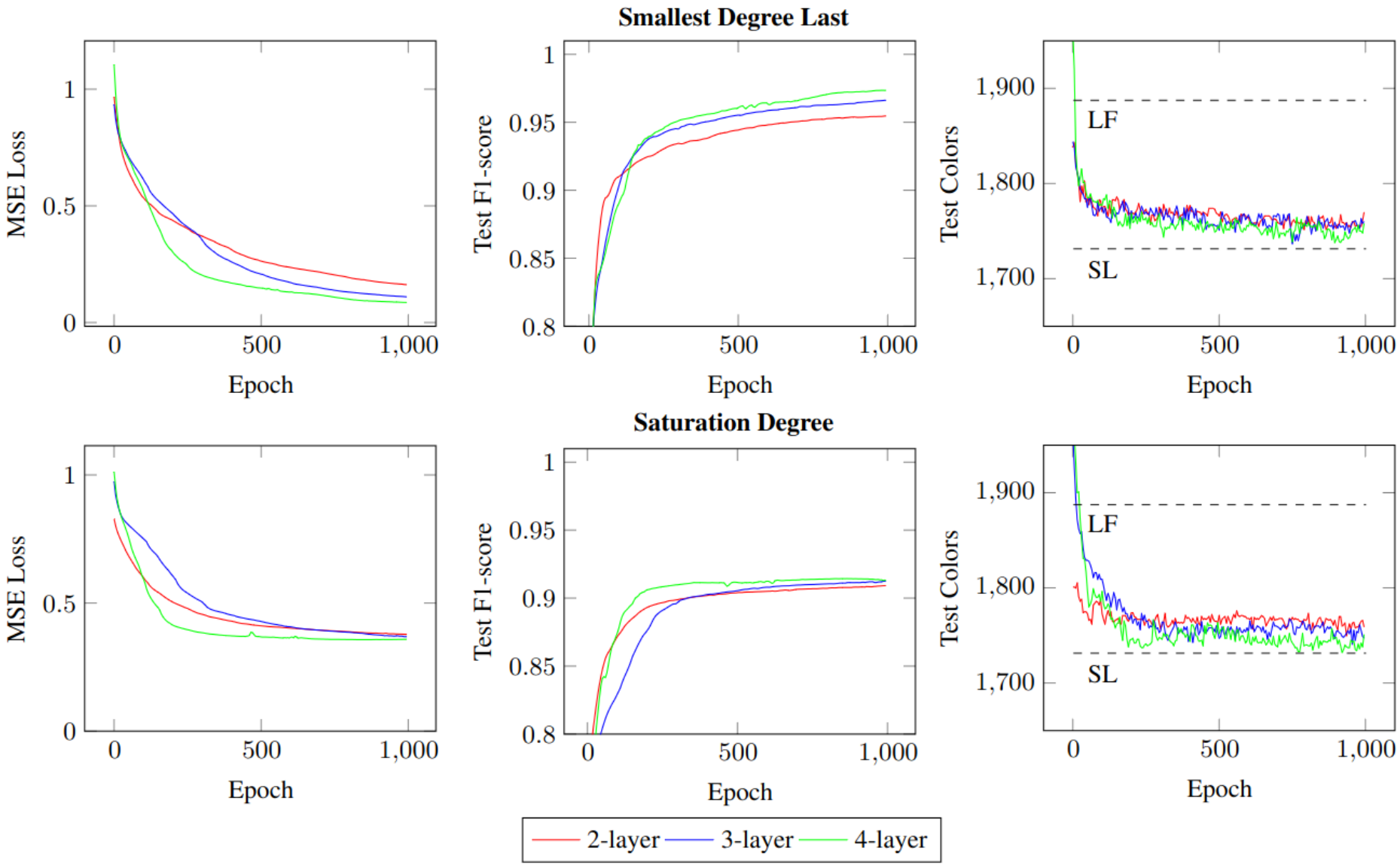
- Supervised learning on SL and SD heuristics
- How to learn ordering?



Supervised Training

- Node to edge layer is only used for training
- Instances from SNAP (Leskovec and Krevl 2014) and DIMACS (Johnson and Trick 1996)
- Training, Test, and Validation sets
- Adam optimizer (Kingma and Ba 2014)
- One mini-batch corresponds to one graph
- Mean squared error loss

Supervised Results



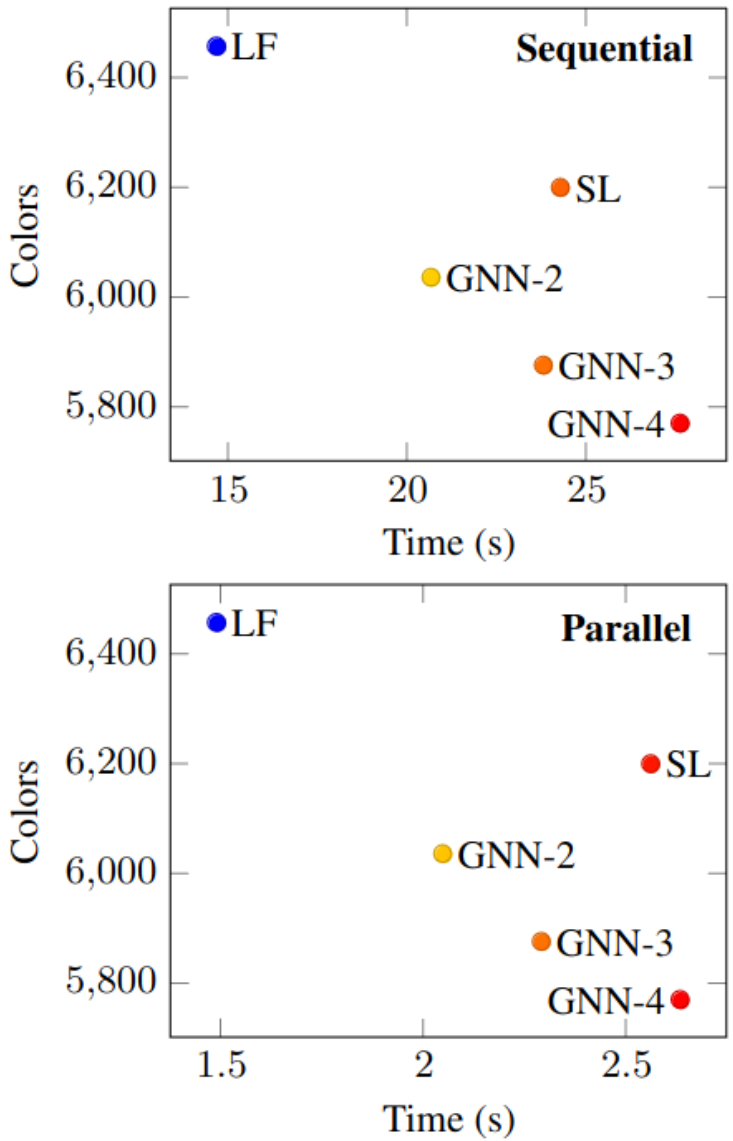
Genetic Training

- Models are trained to imitate other heuristics
- But we care about coloring
- Simple genetic procedure directly on the model parameters
- Evaluate on colors used
- Break ties on the number of vertices colored with the highest color
- $\approx$ 3% reduction on training data
- $\approx$ 1.3% reduction on test data

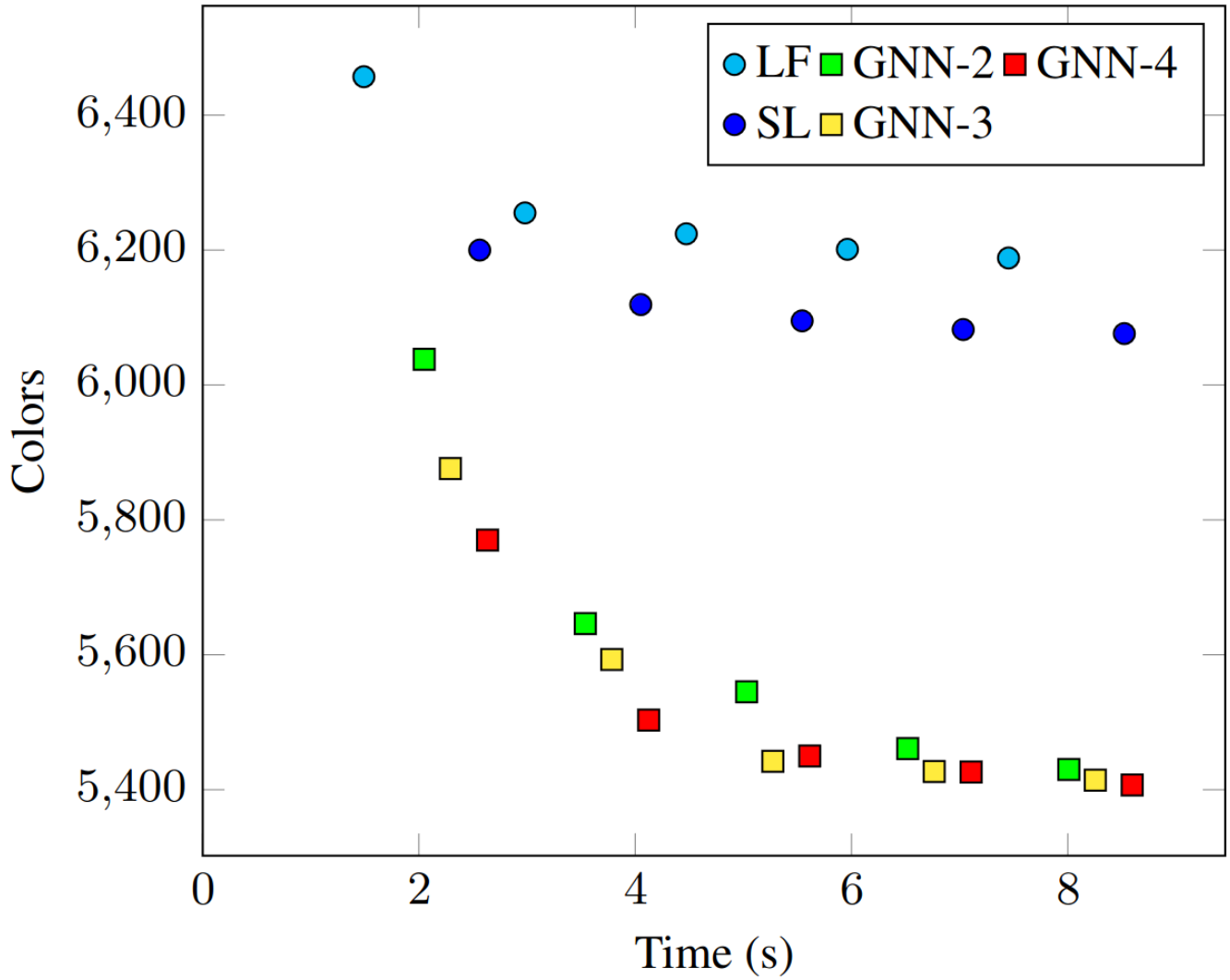
Results

		H	C	T_s	T_1	T_s / T_1	T_{16}	T_{32}	T_1 / T_{16}	T_1 / T_{32}
	twitter7	FF	1,072	48.96	68.25	0.72	19.56	17.22	3.49	3.96
$\|V\|$	41,652,230	LF	1,077	72.71	73.69	0.99	13.99	12.18	5.27	6.05
$\|E\|$	2,405,026,092	SL	917	117.74	122.82	0.96	22.14	20.93	5.55	5.87
		GNN-2	1,103	122.93	124.55	0.99	19.50	16.20	6.39	7.69
		GNN-3	1,005	149.77	152.79	0.98	21.16	18.30	7.22	8.35
		GNN-4	1,031	181.72	183.93	0.99	23.93	20.47	7.69	8.98
	com-Orkut	FF	116	1.02	2.99	0.34	0.59	0.52	5.07	5.72
$\|V\|$	3,072,441	LF	87	4.50	4.72	0.95	0.55	0.39	8.57	12.14
$\|E\|$	234,370,166	SL	83	7.74	8.15	0.95	0.94	0.74	8.64	10.98
		GNN-2	85	7.02	7.17	0.98	0.76	0.59	9.40	12.19
		GNN-3	82	8.07	8.20	0.98	0.85	0.67	9.67	12.20
		GNN-4	82	9.38	9.51	0.99	0.97	0.79	9.80	12.05
	LiveJournal1	FF	325	0.47	1.81	0.26	0.33	0.28	5.54	6.46
$\|V\|$	4,847,571	LF	323	2.21	2.32	0.95	0.32	0.24	7.24	9.74
$\|E\|$	85,702,474	SL	322	3.42	3.67	0.93	0.50	0.39	7.33	9.32
		GNN-2	324	3.21	3.33	0.96	0.41	0.33	8.08	10.20
		GNN-3	322	3.66	3.79	0.97	0.45	0.36	8.50	10.60
		GNN-4	321	4.20	4.32	0.97	0.50	0.41	8.63	10.60
	cit-Patents	FF	15	0.26	1.28	0.20	0.11	0.08	11.42	15.91
$\|V\|$	3,774,768	LF	14	1.56	1.69	0.92	0.15	0.10	10.93	16.55
$\|E\|$	33,037,894	SL	13	2.26	2.50	0.90	0.23	0.16	10.79	15.52
		GNN-2	13	2.00	2.16	0.92	0.19	0.14	11.16	15.09
		GNN-3	13	2.31	2.50	0.92	0.23	0.18	10.79	14.27
		GNN-4	13	2.64	2.85	0.93	0.27	0.20	10.72	13.94

Results


LF
Sequential
SL
GNN-2
GNN-3
GNN-4
6,400
6,200
6,000
5,800
Colors
15
20
25
Time (s)
LF
Parallel
SL
GNN-2
GNN-3
GNN-4
6,400
6,200
6,000
5,800
Colors
1.5
2
2.5
Time (s)

Culberson


LF
GNN-2
GNN-4
SL
GNN-3
Colors
6,400
6,200
6,000
5,800
5,600
5,400
0
2
4
6
8
Time (s)

References I

- Brélaz, Daniel (1979). 'New methods to color the vertices of a graph'. In: *Communications of the ACM* 22.4, pp. 251–256.
- Hamilton, Will, Zhitao Ying and Jure Leskovec (2017). 'Inductive representation learning on large graphs'. In: *Advances in neural information processing systems* 30.
- Hasenplaugh, William et al. (2014). 'Ordering heuristics for parallel graph coloring'. In: *Proceedings of the 26th ACM symposium on Parallelism in algorithms and architectures*, pp. 166–177.
- Johnson, David S and Michael A Trick (1996). *Cliques, coloring, and satisfiability: second DIMACS implementation challenge, October 11-13, 1993*. Vol. 26. American Mathematical Soc.
- Jones, Mark T and Paul E Plassmann (1993). 'A parallel graph coloring heuristic'. In: *SIAM Journal on Scientific Computing* 14.3, pp. 654–669.

References II

- Kingma, Diederik P and Jimmy Ba (2014). 'Adam: A method for stochastic optimization'. In: *arXiv preprint arXiv:1412.6980*.
- Leskovec, Jure and Andrej Krevl (June 2014). *SNAP Datasets: Stanford Large Network Dataset Collection*. http://snap.stanford.edu/data.
- Matula, David W and Leland L Beck (1983). 'Smallest-last ordering and clustering and graph coloring algorithms'. In: *Journal of the ACM (JACM)* 30.3, pp. 417–427.
- Newsam, Garry N and John D Ramsdell (1983). 'Estimation of sparse Jacobian matrices'. In: *SIAM Journal on Algebraic Discrete Methods* 4.3, pp. 404–418.
- Welsh, Dominic JA and Martin B Powell (1967). 'An upper bound for the chromatic number of a graph and its application to timetabling problems'. In: *The Computer Journal* 10.1, pp. 85–86.