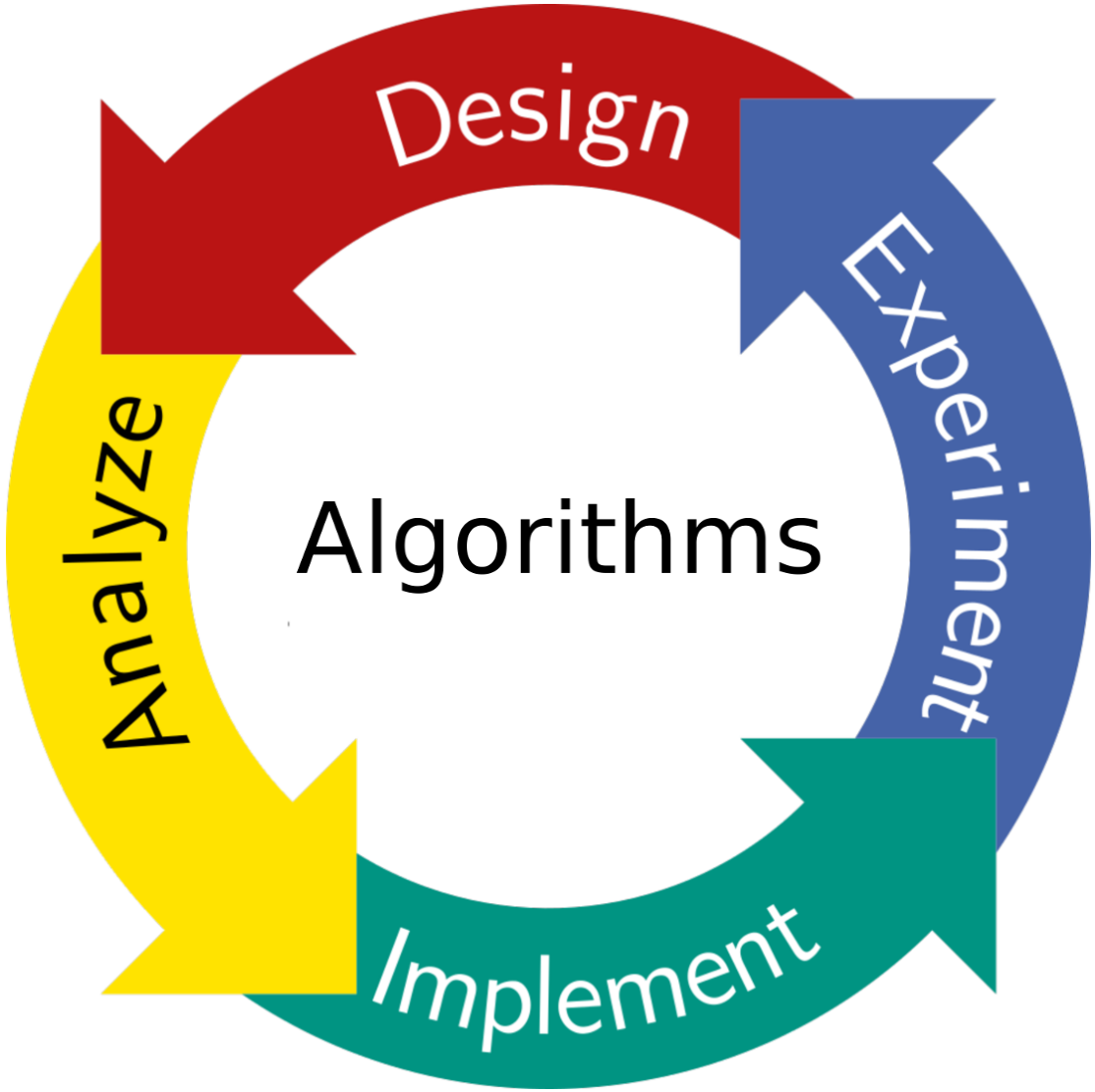


If you want to participate, turn on your camera.

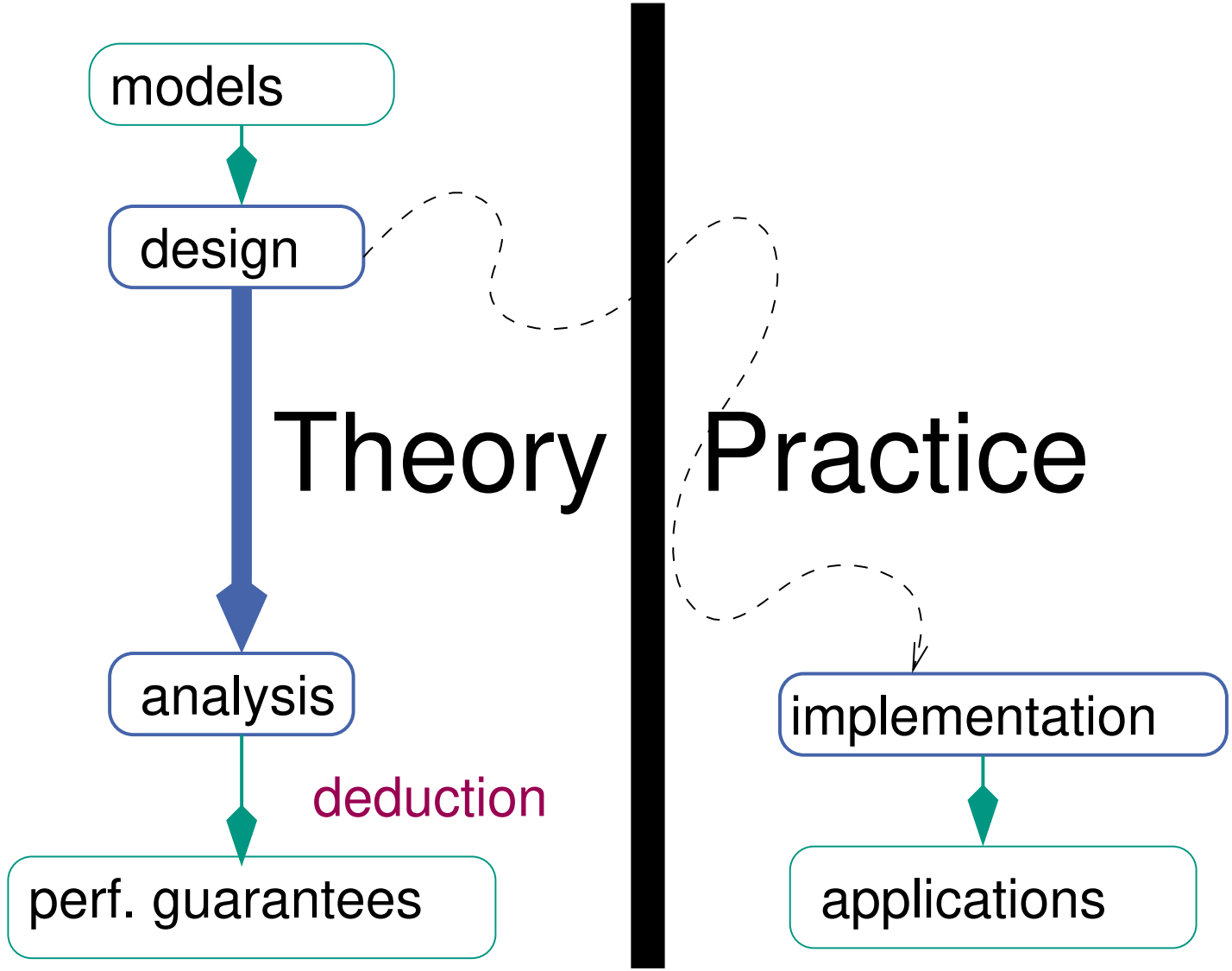
Organization

Lecturer

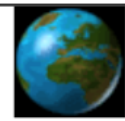
Christian Schulz

- University of Heidelberg
- Mail:
  `christian.schulz@informatik.uni-heidelberg.de`
- Room 1/328
- Consultation: Sa. 00:00-01:00 Uhr
- This seminar $\rightarrow$ 4 LP

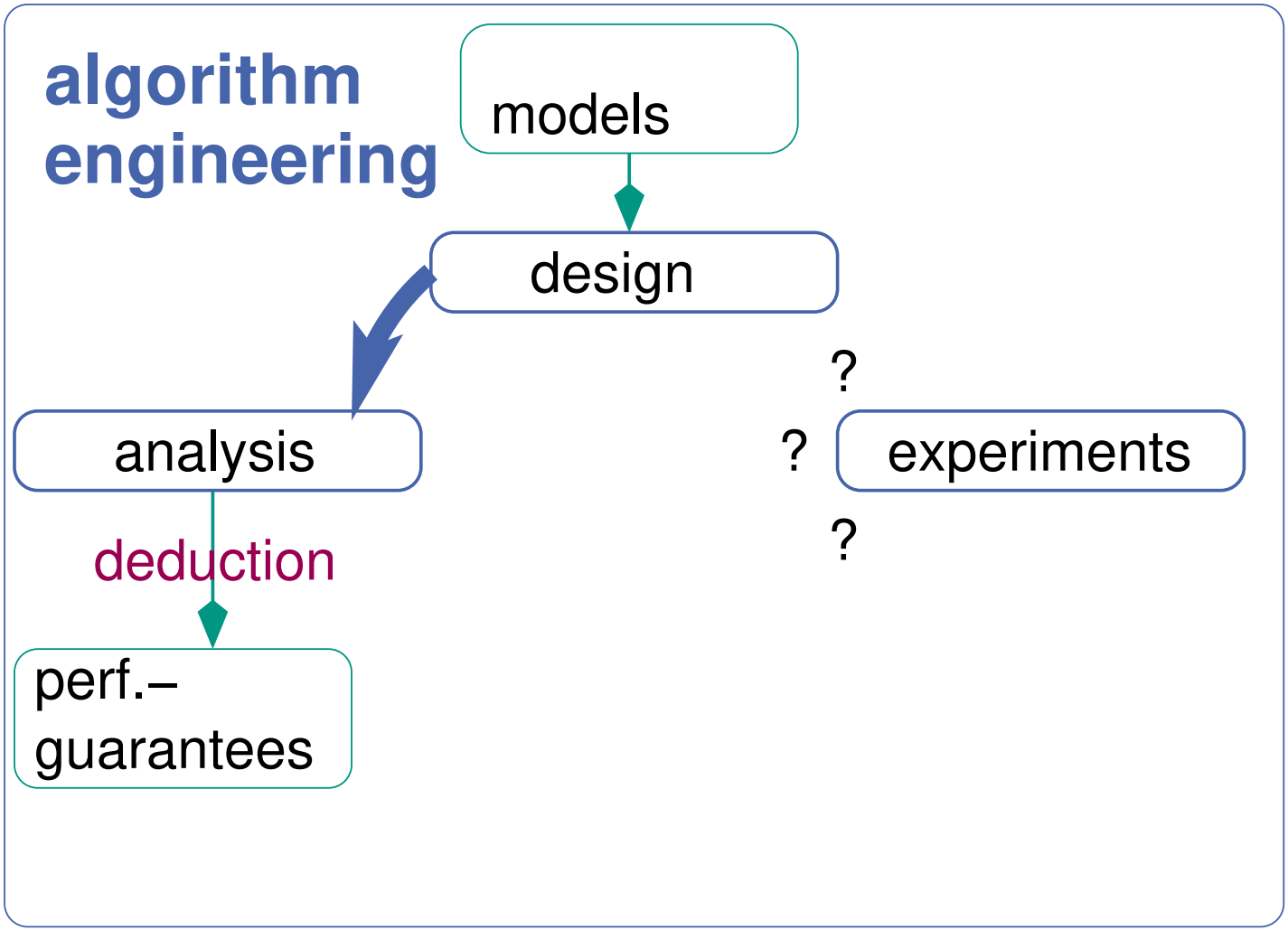
(Caricatured) Traditional Algorithm Theory



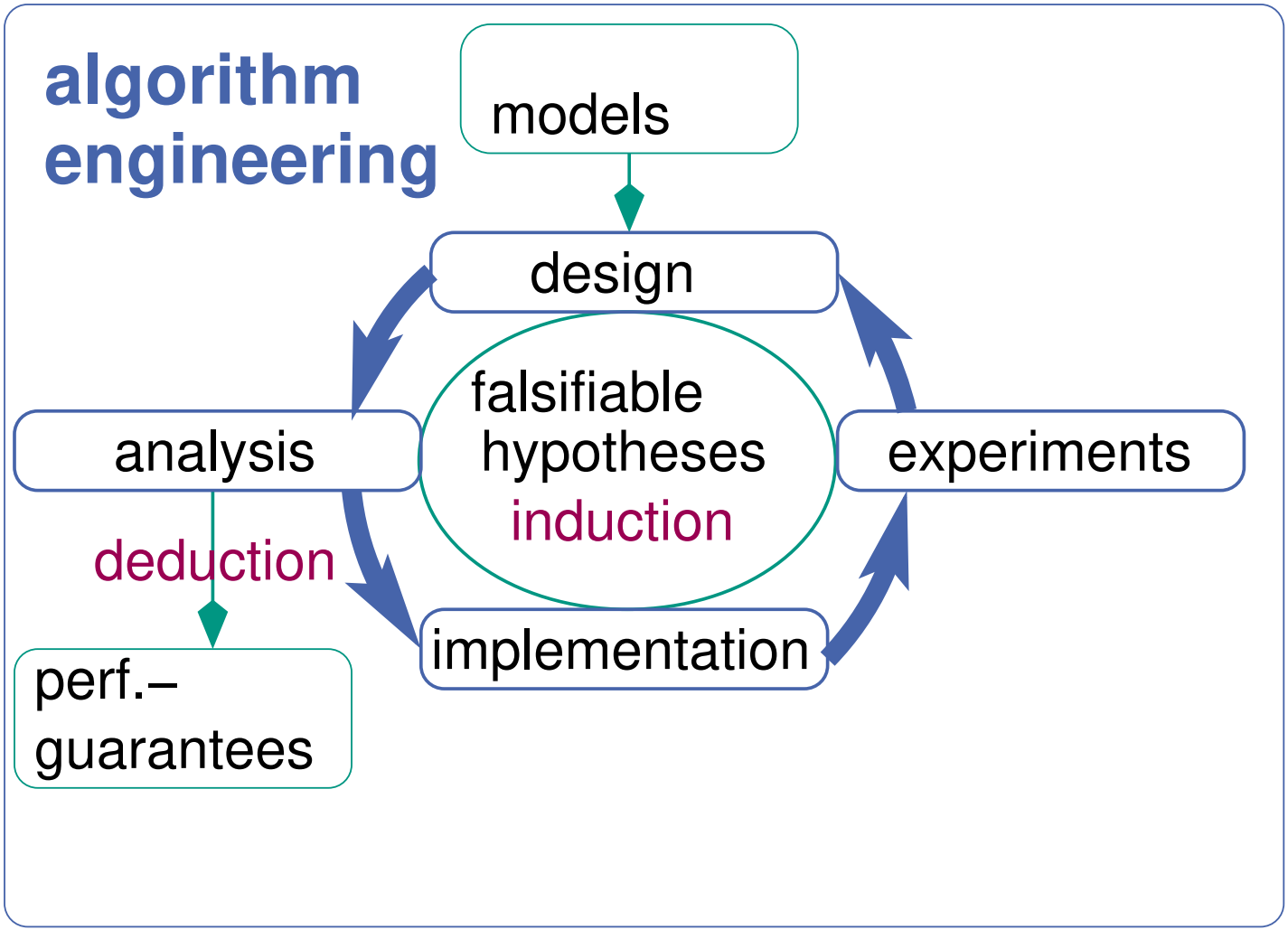
Gaps Between Theory & Practice

Theory	⟷	Practice
simple 	**appl. model**	 complex
simple 	**machine model**	 real
complex 	**algorithms**	FOR simple
advanced 	**data structures**	arrays,. . .
worst case max	**complexity measure**	 inputs
asympt. $O(\cdot)$	**efficiency**	42% constant factors

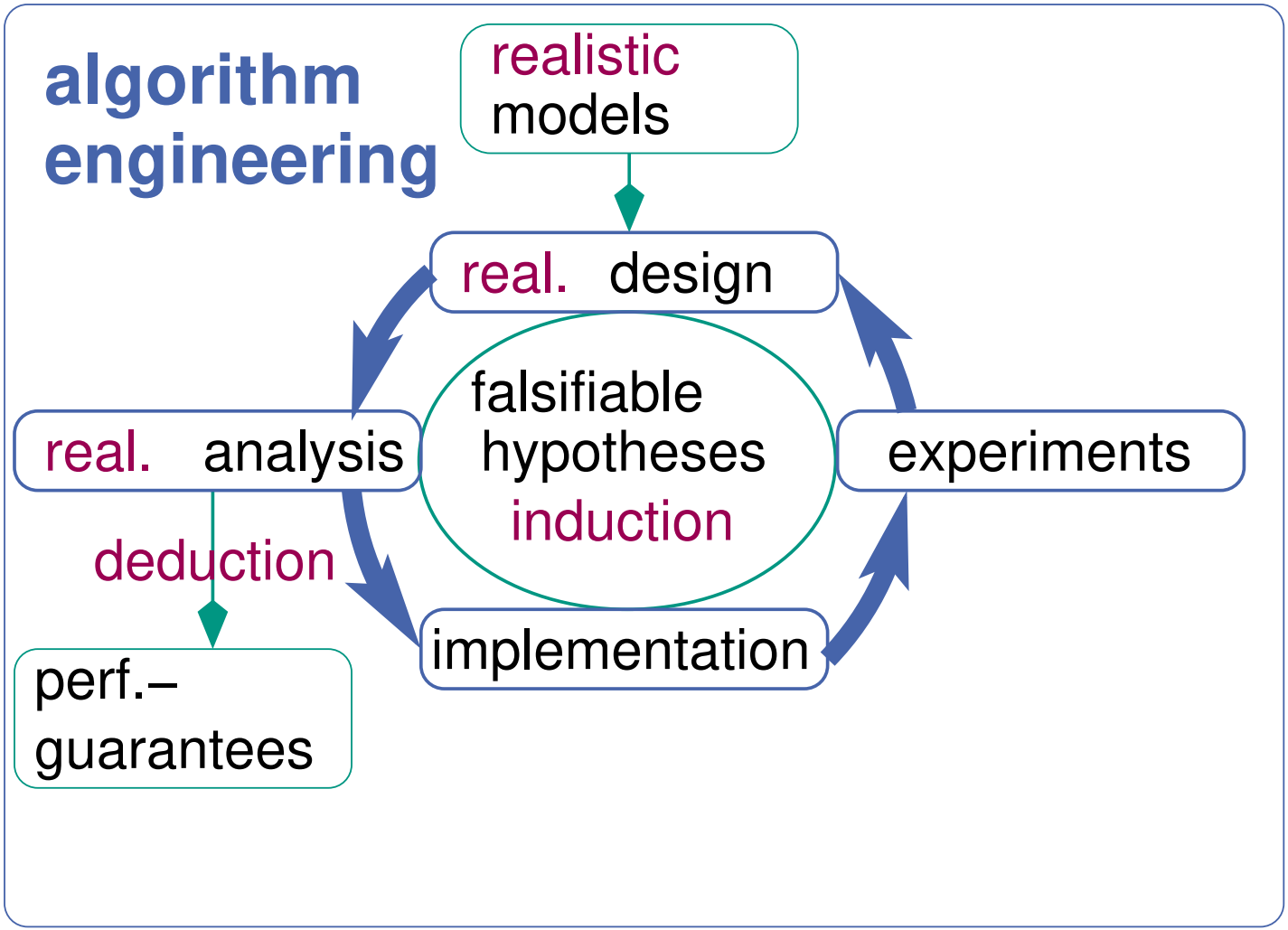
Algorithmics as Algorithm Engineering



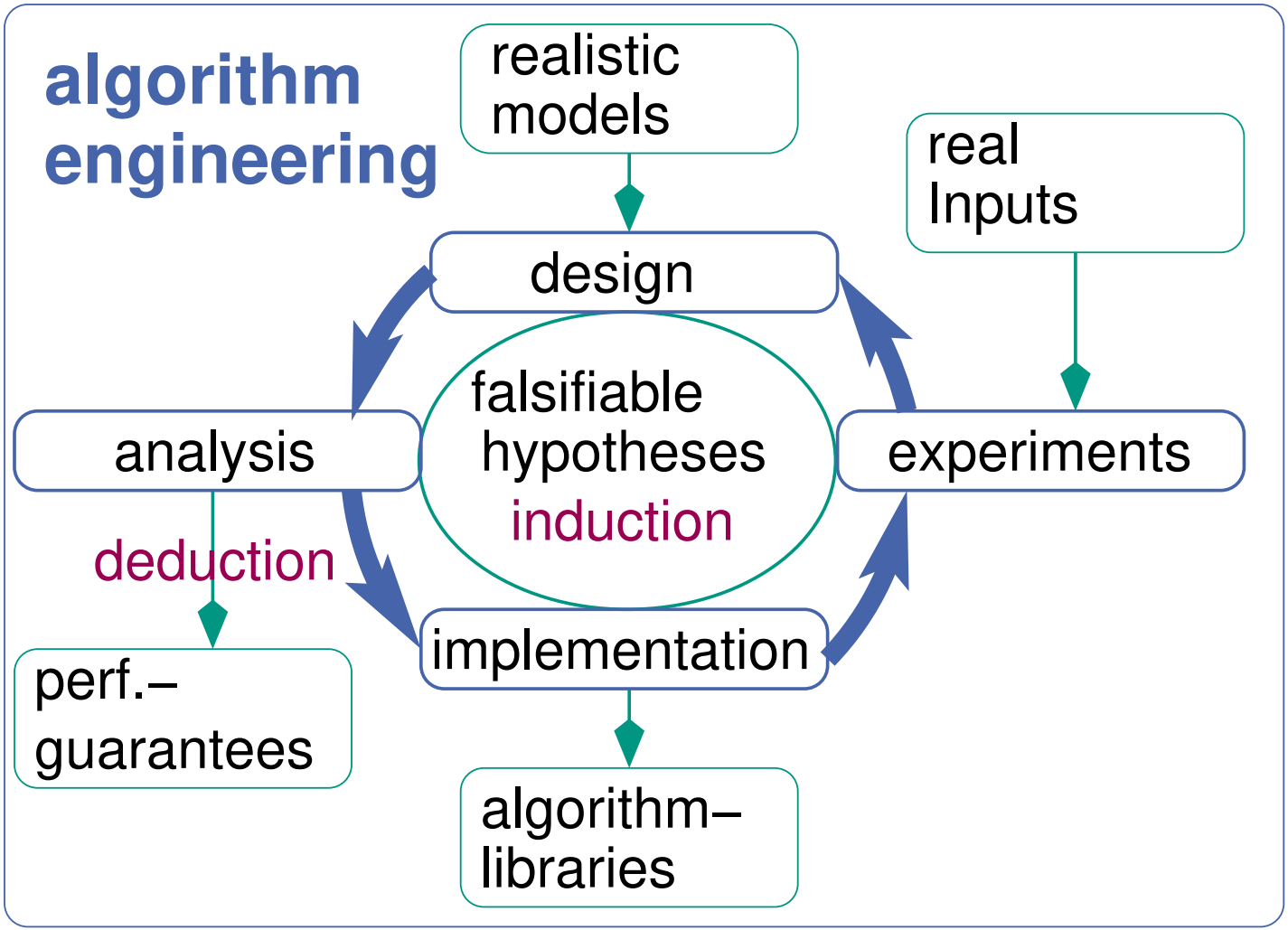
Algorithmics as Algorithm Engineering



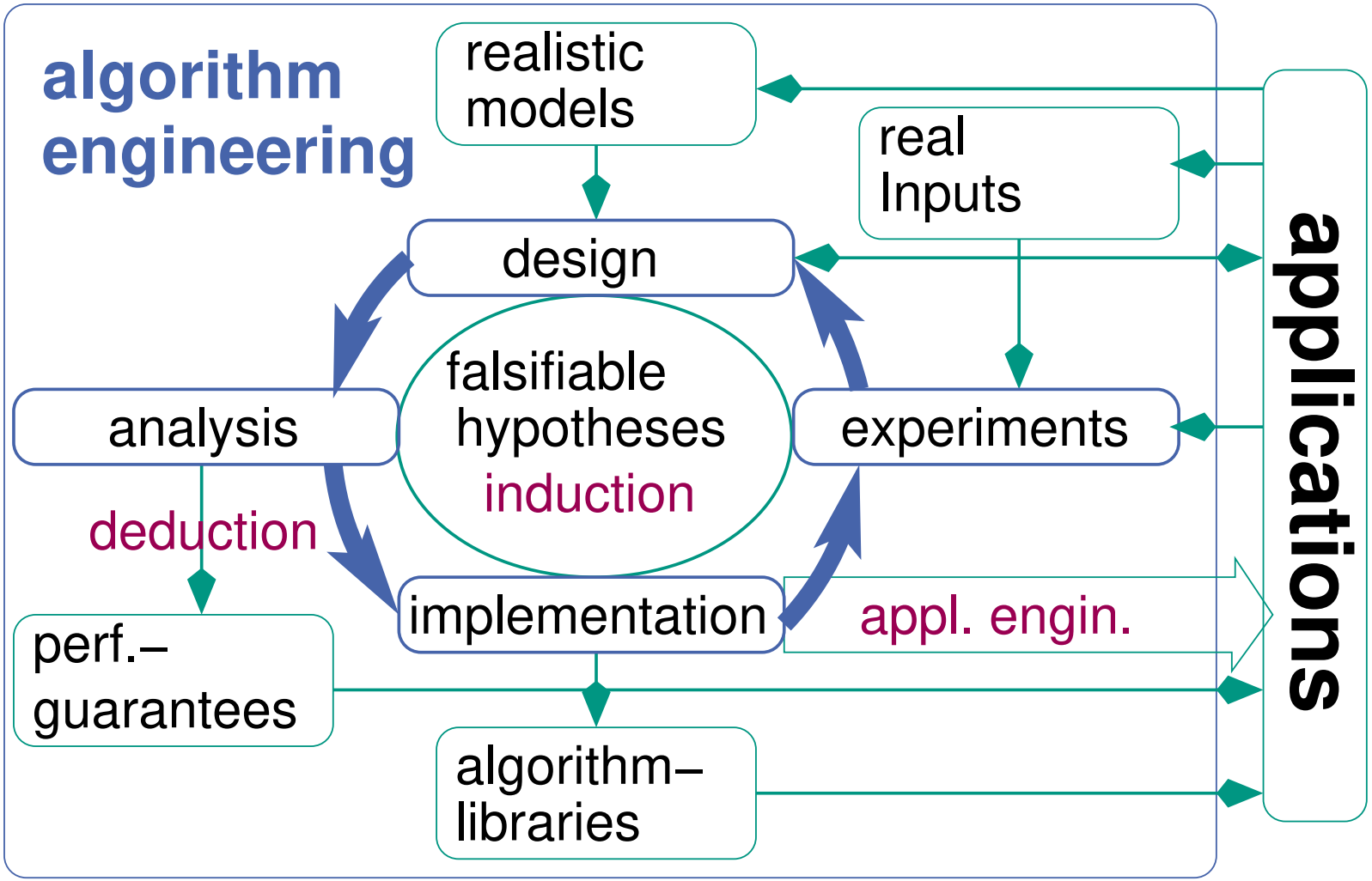
Algorithmics as Algorithm Engineering



Algorithmics as Algorithm Engineering

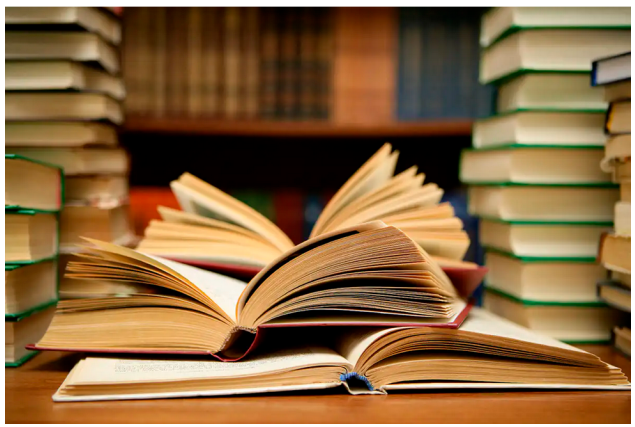


Algorithmics as Algorithm Engineering



Organization

- 8 LP, 6 SWS (adv), 2LP+4LPFK, 4 SWS (beginner)
- 240h workload (adv), 180h workload (beginner)
- The project takes place online
- We will have weekly meetings
- We can only take 2-3 students.
- The project will conclude with a 15min presentation
- You already need programming very good skills in C++ or C. Algorithms will be programmed in that language. If you feel uncomfortable in C++ don't take the course.



Organization

The course teaches you how to do algorithm engineering RESEARCH.

- The programming projects are research oriented, i.e. you will work on a topic that may turns out to become a publication.
- We will (often) work with (international) colleagues
- You have to hand in a written report in the end.
- The programming project is meant as a preparation for a thesis project! If you finish the project successfully then, you can and *should* continue to work on the topic for your final thesis project

Organization

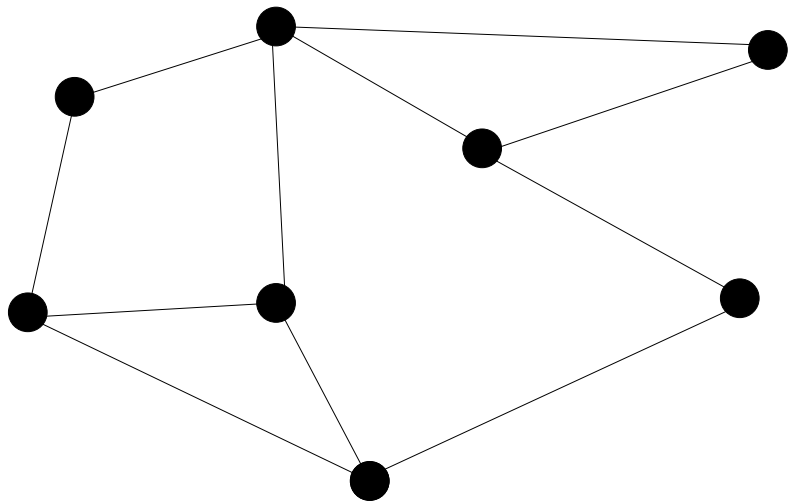
Send me your transcripts
and
a list of topics that you like!

Topics

Dynamic Matching

Graphs Change over Time

Graph subject to **update** operations

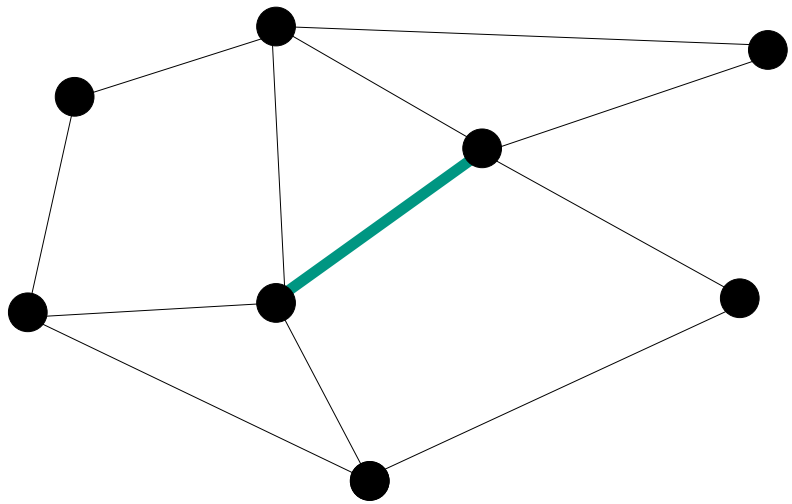


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

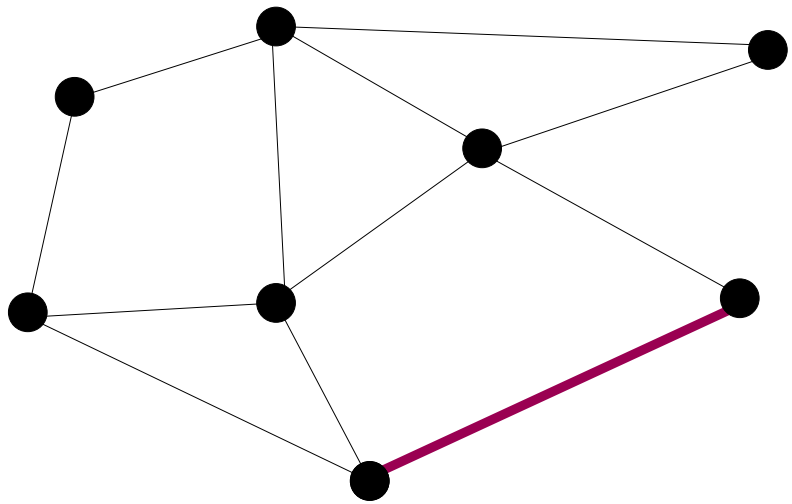


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

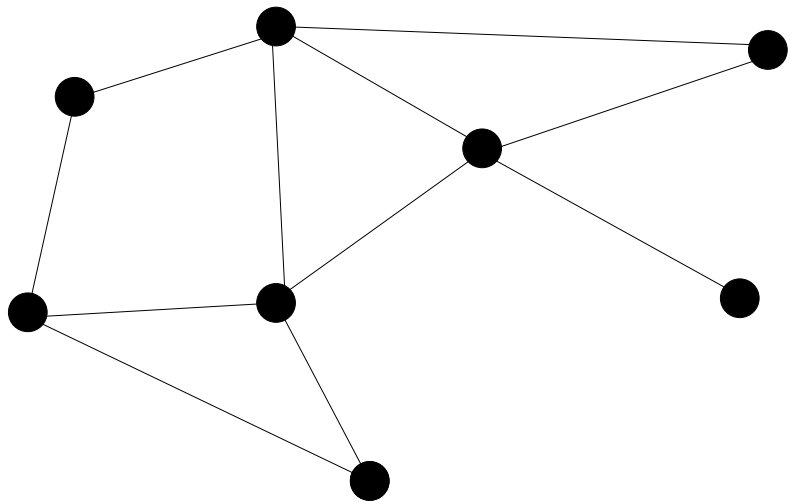


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

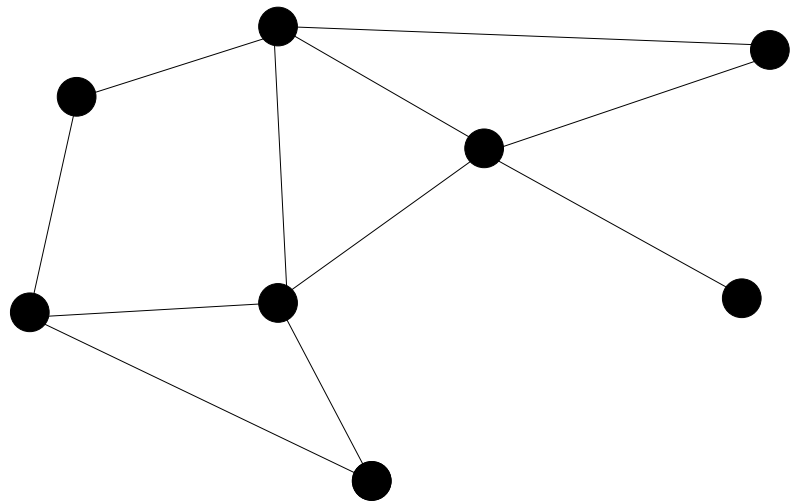


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function

Graphs Change over Time

Graph subject to **update** operations

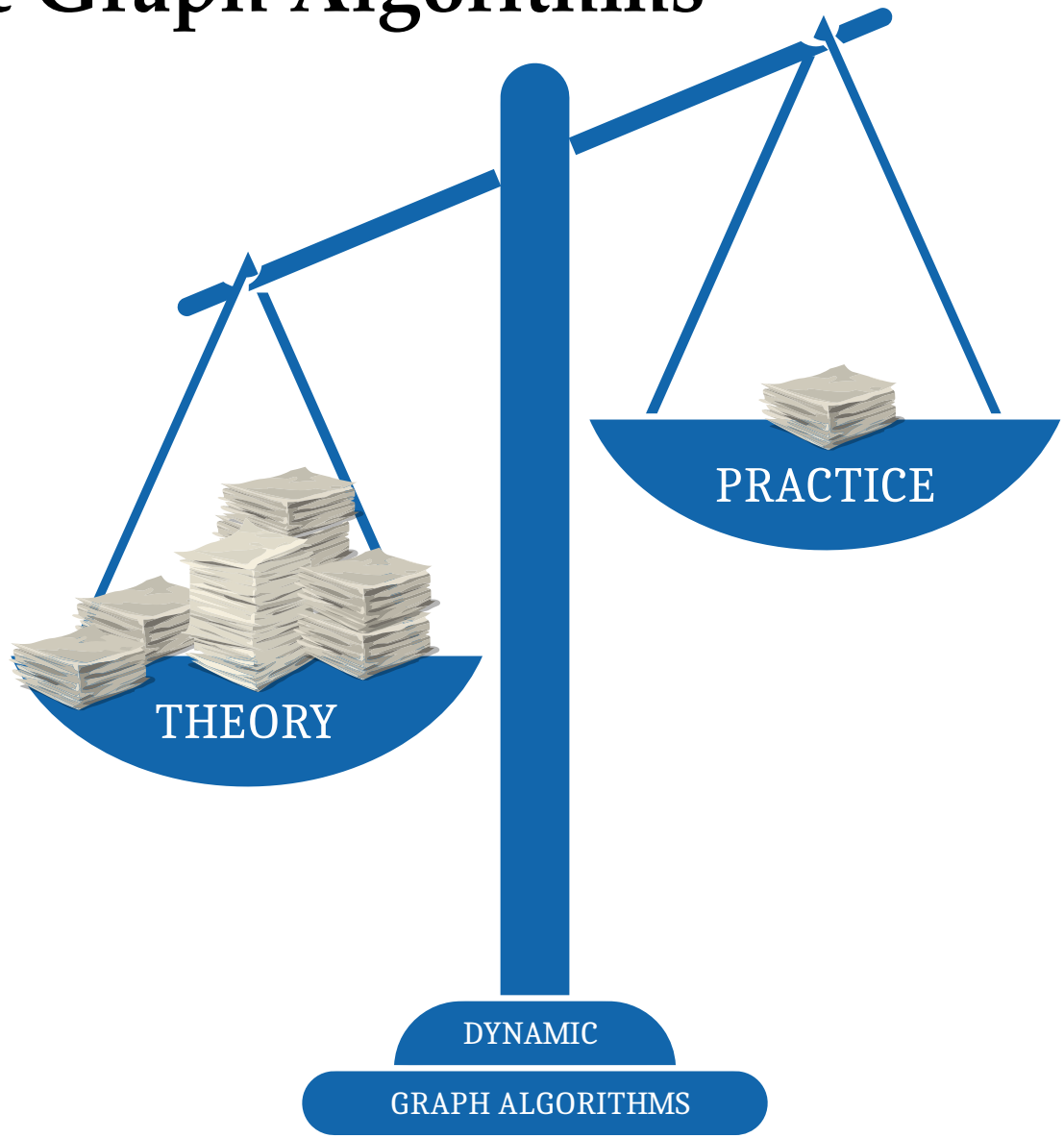


Typical updates: insert(u,v), delete(u,v), set_weight(u,v,ω)

+ additional *init* function
+ additional *query* function
$\Rightarrow$ every static algorithm gives you a *naive* dynamic algorithm

Dynamic Graph Algorithms

Context



Definitions

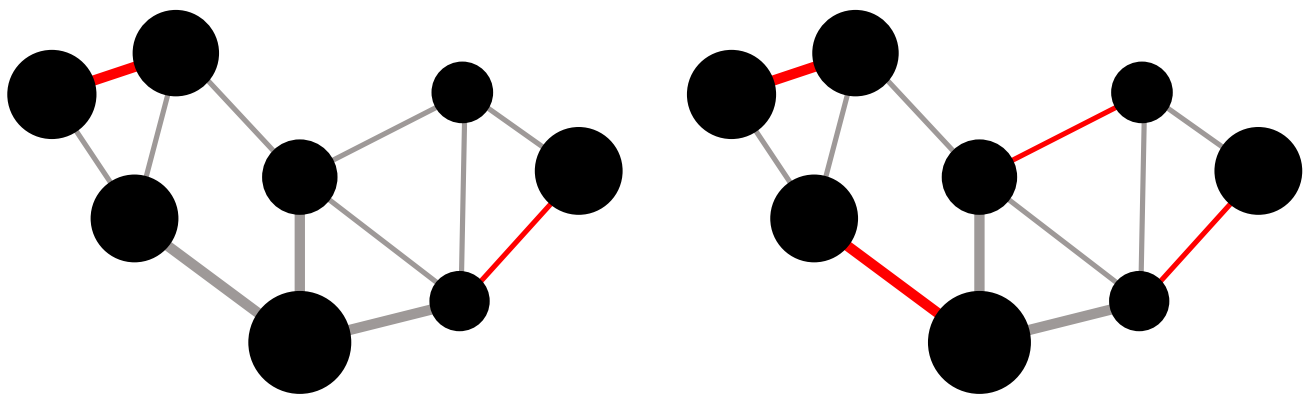
Definition (Matching)

A matching $\mathcal{M} \subseteq E$ is a set of edges not sharing any end point.
I.e. $G = (V, \mathcal{M})$ has maximum degree one.

A matching is maximal iff no edge can be added to the matching.

The weight of a matching is $\sum_{e \in \mathcal{M}} \omega(e)$.

A maximum weight matching has the largest weight possible weight.



Random Walks

One Step at a Time:

- starting from a free vertex u, match a random edge (u, v)
- if v was free, we are done
- if it was matched with say w, unmatch (v, w)
- and continue random walk at w

Insertion/Deletion:

- start random walk of length $2/\epsilon - 1$ from free vertices
- on insert, if both endpoints are free, directly match it
- Δ-settling: for each vertex check all neighbors to find free vertex

Theorem

The random walk based algorithm maintains a $(1+\epsilon)$-approximate maximum matching if the length of the walk is $2/\epsilon - 1$ and the walks are repeated $\Delta^{2/\epsilon-1} \log n$ times.

We want to transfer this to edge weighted graphs.

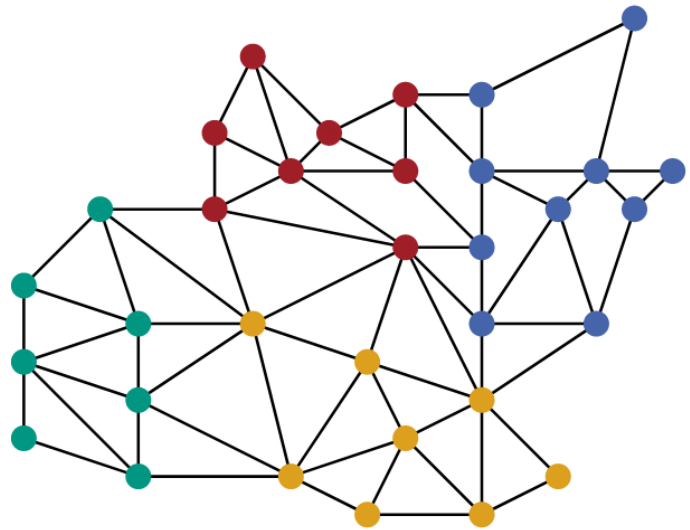
Another Option

Matchings in Hypergraphs (with Felix Joos)

Graph $G = (V, E)$

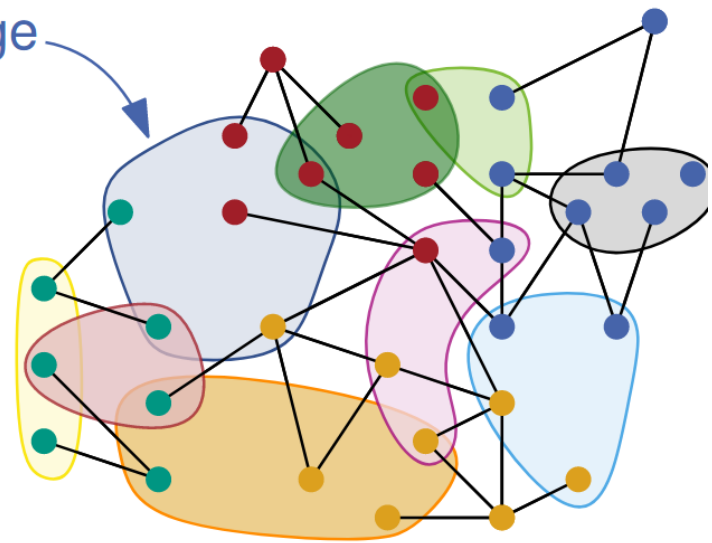
vertices — edges

- Models relationships between objects
- Dyadic (2-ary) relationships



Hypergraph $H = (V, E)$

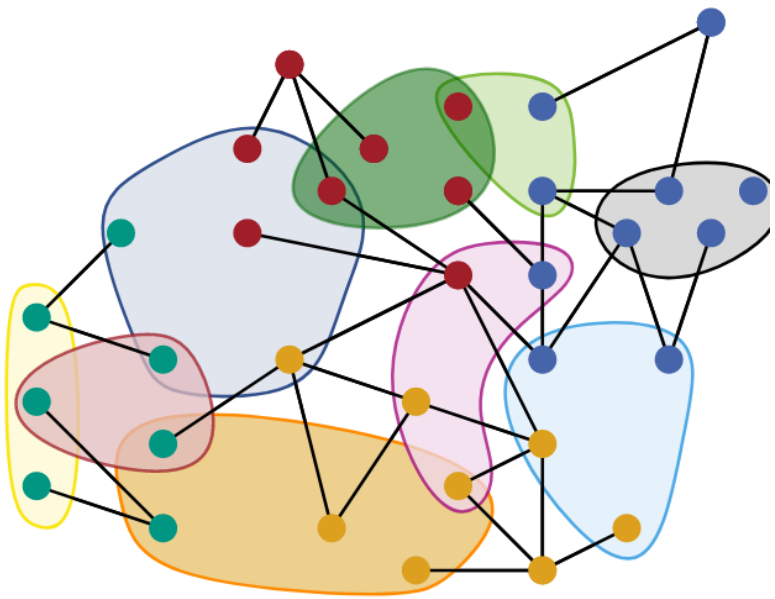
- Generalization of a graph
  $\Rightarrow$ hyperedges connect ≥ 2 nodes
- Arbitrary (d-ary) relationships
- Edge set $E \subseteq \mathcal{P}(V) \setminus \emptyset$



Problem Statements

Hypergraph matching: As before, only with hyperedges
Widely Open:

- Engineer heuristic algorithms that perform well in practice
- Engineer branch-and-bound algorithms for problem
- Engineer reductions for the problem
- THEORY: how long are augmenting paths on average? (graphs)

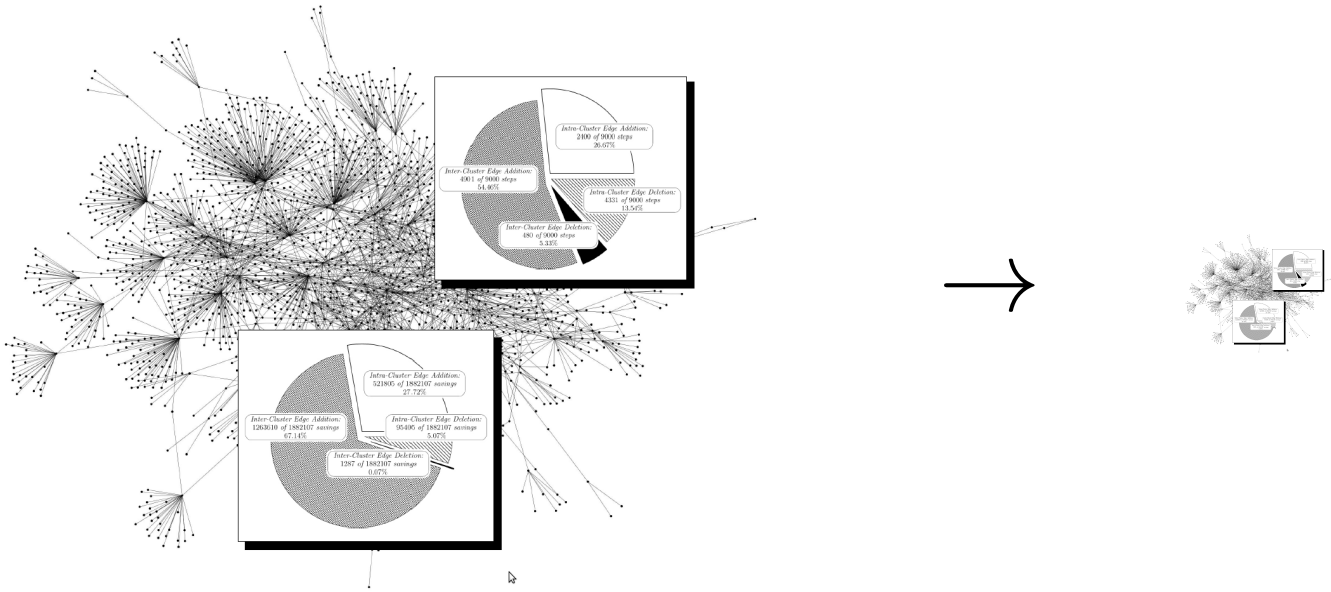


Weighted Independent Sets Meets Machine Learning

Kernelization

General Idea

Reductions:
rules to decrease graph size, while maintaining optimality

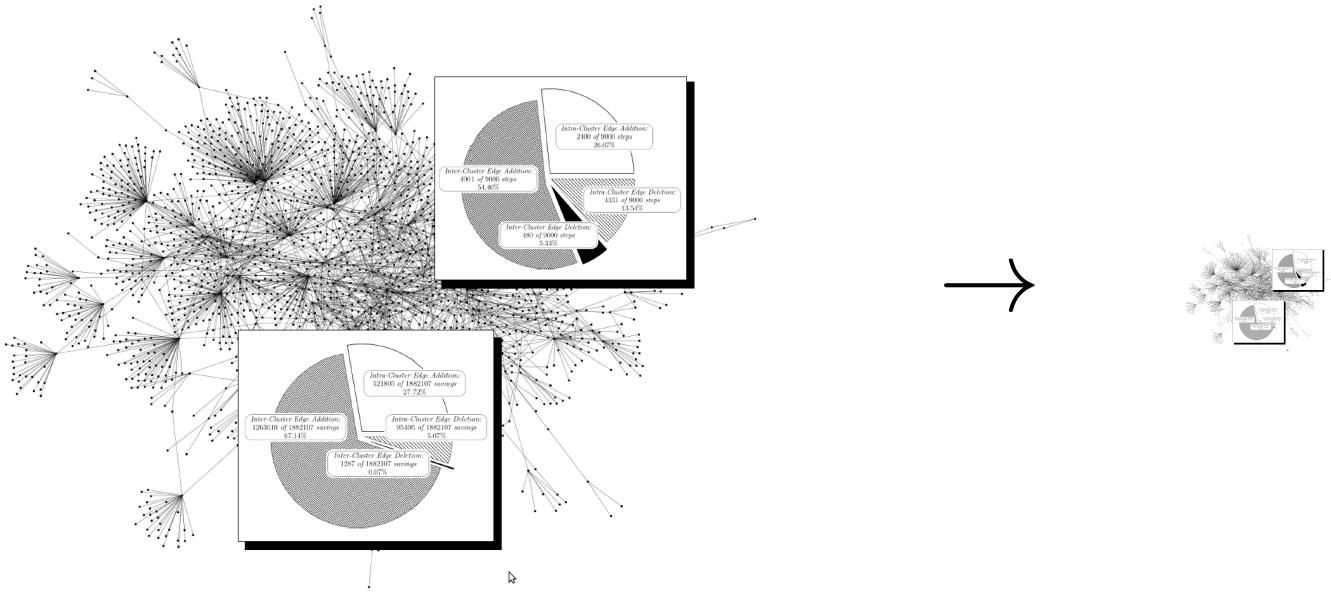


solve problem on problem kernel
$\rightarrow$ obtain solution on input graph

Kernelization

General Idea

Reductions:
rules to decrease graph size, while maintaining optimality

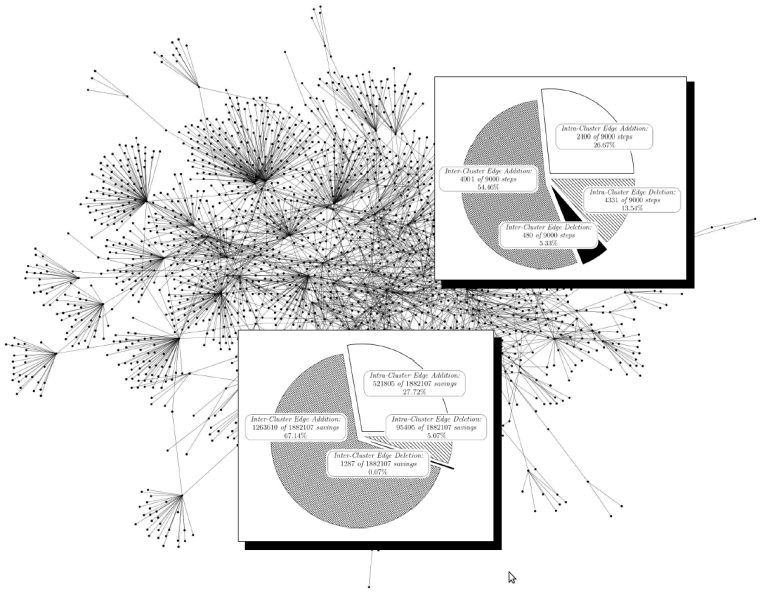


solve problem on problem kernel (using a heuristic)
$\rightarrow$ obtain solution on input graph quickly

Kernelization

General Idea

Reductions:
rules to decrease graph size, while maintaining optimality



solve problem on problem kernel
→ obtain solution on input graph

Independent Sets

- evolutionary [SEA'15]
- reduction + evolutionary [ALX'16]
- online reductions + LS [SEA'16]
- shared-mem parallel [ALX'18]
- weighted exact [ALX'19]

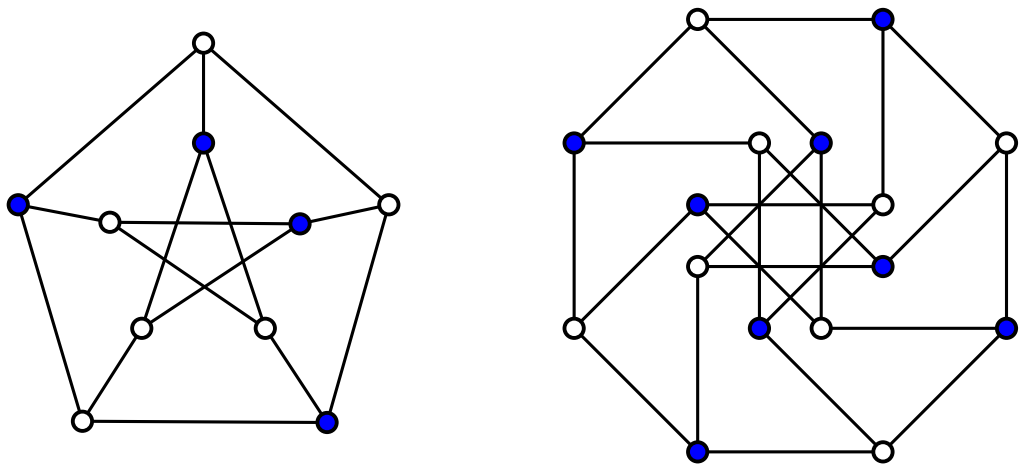
Definitions

Independent Set:

- subset $S \subseteq V$ such that there are no adjacent nodes in S

Maximum Independent Set (MIS):

- maximum cardinality set S
- maximum weight IS ($\max_S \sum_{v \in S} c(v)$)



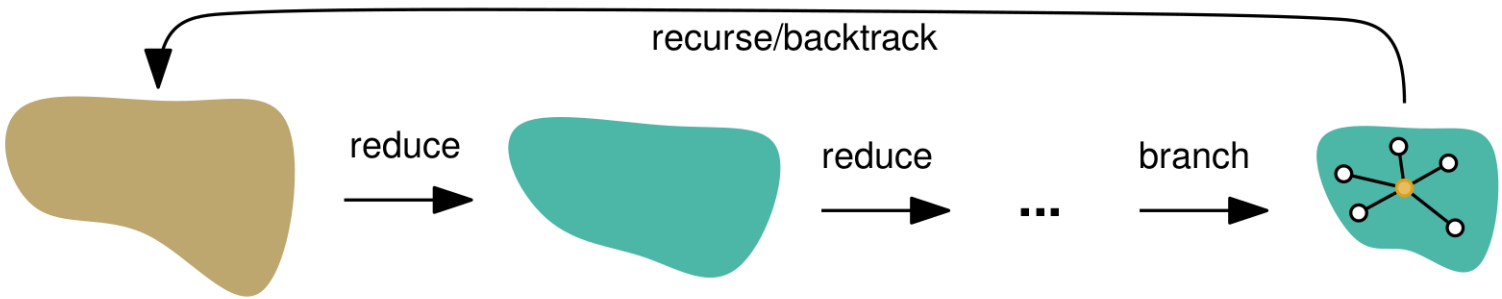
- finding a MIS is **NP-hard** and hard to approximate

Exact Algorithms

Independent Sets

Exact solution using **branch and bound**:

- Modify and remove subgraphs during recursions $\rightarrow$ **reductions**
- Branch when the graph can no longer be reduced



$\rightarrow$ Running time $O(1.211^n)$ [Akiba, Iwata'16]

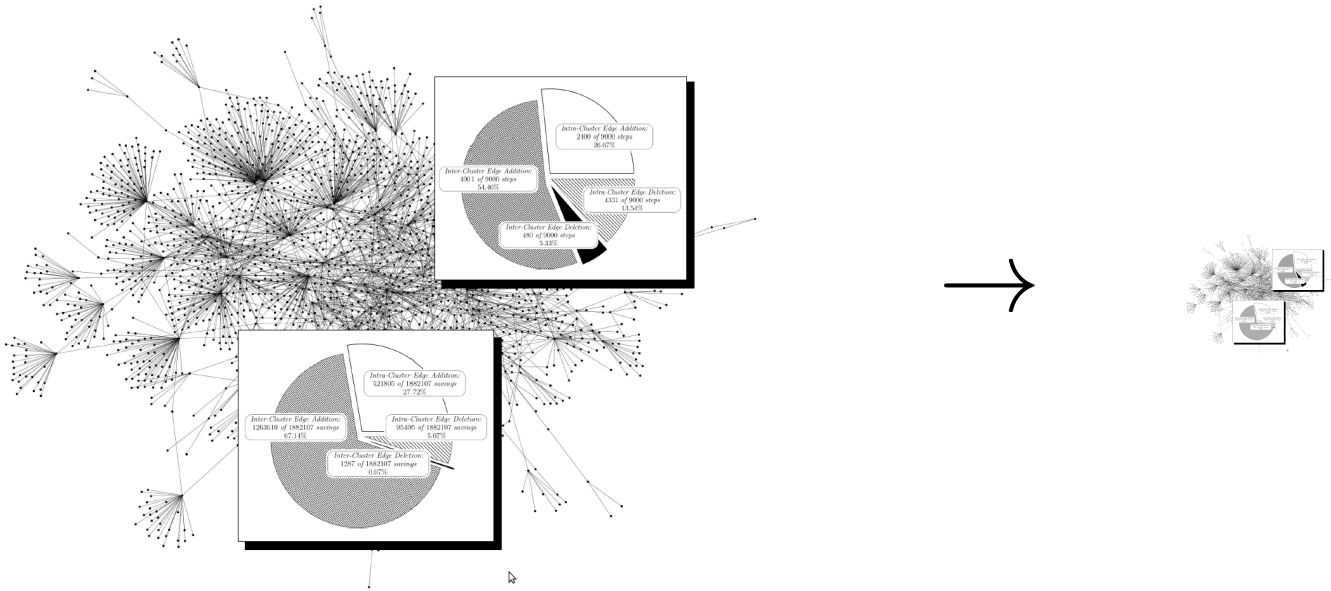
Works well, except when it doesn't:

- Many networks with **small kernels** are easy to solve
- Similar instances with **large kernels** remain unsolved

Kernelization

[Akiba, Iwata'15]

Reductions:
rules to decrease graph size, while maintaining optimality



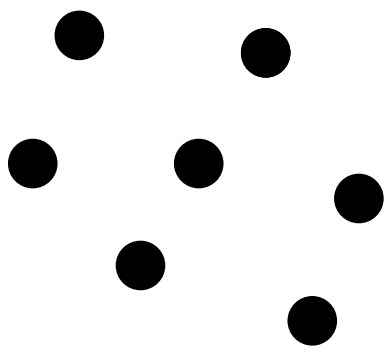
solve problem on problem kernel (using EA)
$\rightarrow$ obtain solution on input graph

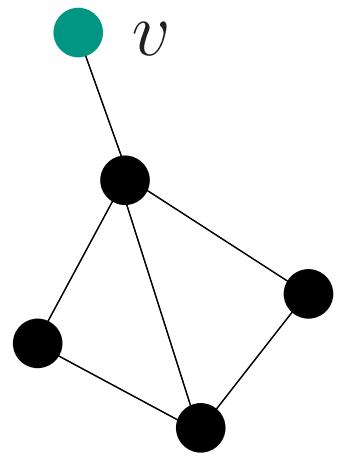
Kernelization

[Akiba, Iwata'15]

Reductions:
rules to decrease graph size, while maintaining optimality

Example:
remove degree 0 or 1 vertices recursively





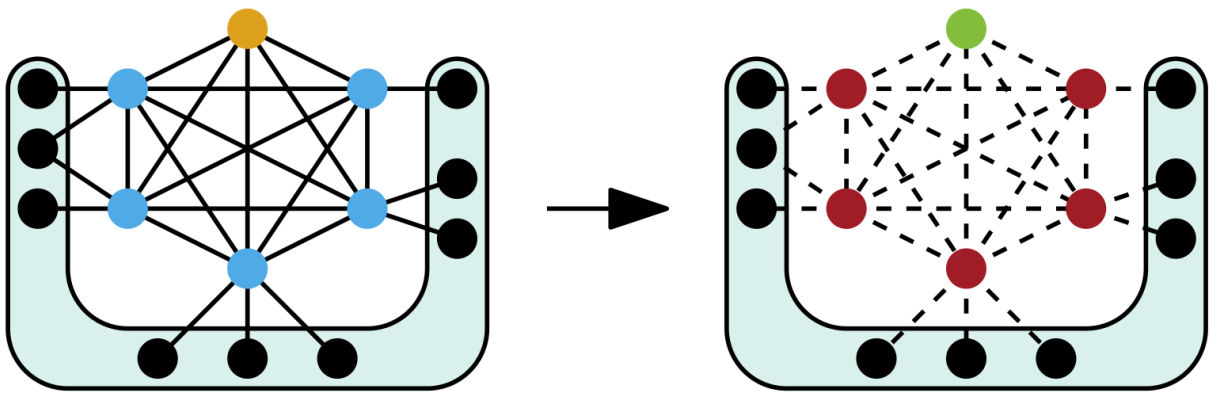
there is always a MIS that contains v
if neighbor of v in MIS choose v instead; otherwise add v

Kernelization

[Akiba, Iwata'15]

Reductions:
rules to decrease graph size, while maintaining optimality

Example:
isolated clique reduction

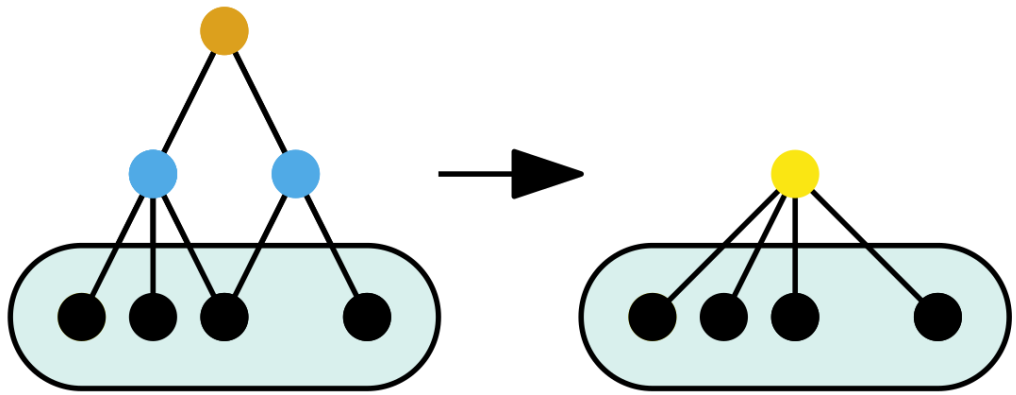


Kernelization

[Akiba, Iwata'15]

Reductions:
rules to decrease graph size, while maintaining optimality

Example:
vertex folding



Kernelization

[Akiba, Iwata'15]

Reductions:
rules to decrease graph size, while maintaining optimality

Example:
linear programming relaxation for MIS

$$\max \sum_{v \in V} x_v \text{ s.t.}$$
$$\forall (u,v) \in E : x_u + x_v \leq 1$$
$$\forall v \in V : x_v \geq 0$$

- $\exists$ half integral solution (i.e., using only values 0, 1/2, and 1)
- solve via maximum bipartite matching (also parallel, stay tuned)
- vertices with value 1, MUST be in the MIS

Kernelization

[Akiba, Iwata'15]

Reductions:
rules to decrease graph size, while maintaining optimality

Example:
linear programming relaxation for MIS

$$\max \sum_{v \in V} x_v \text{ s.t.}$$
$$\forall (u,v) \in E : x_u + x_v \leq 1$$
$$\forall v \in V : x_v \geq 0$$

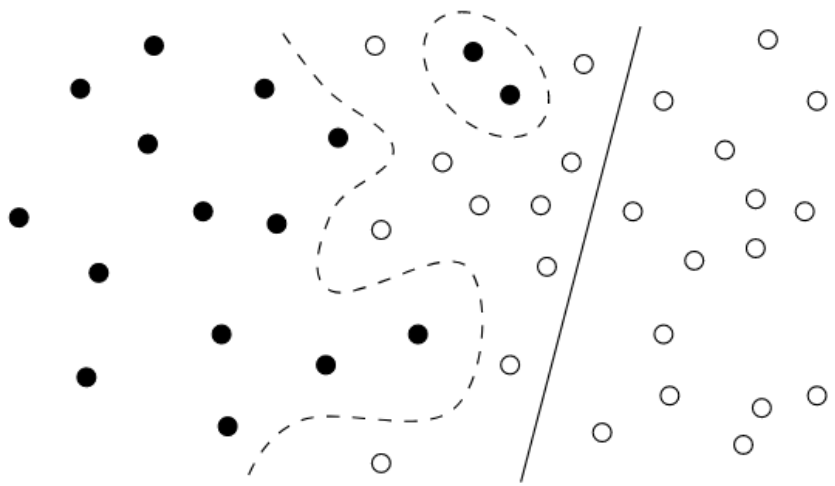
- $\exists$ half integral solution (i.e., using only values 0, 1/2, and 1)
- solve via maximum bipartite matching (also parallel, stay tuned)
- vertices with value 1, MUST be in the MIS

more reductions used in practice $\rightarrow$ [ALENEX'16]

Machine Learning in Optimization

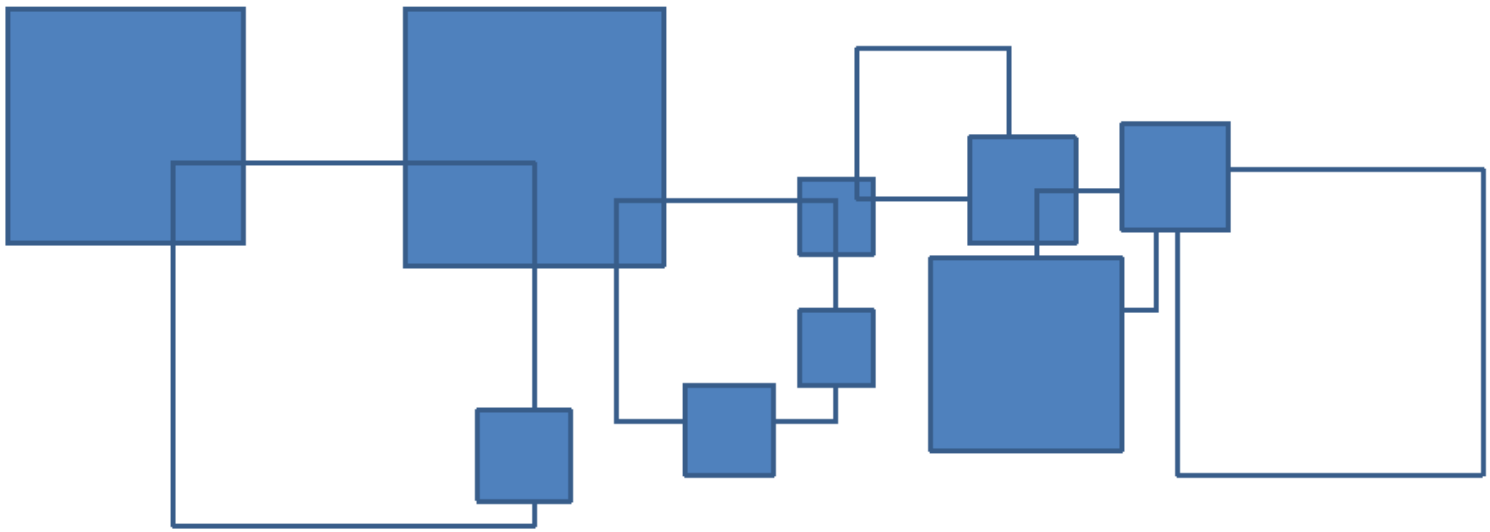
Rough Project Idea

- Given a set of instances that you can solve,
  i.e. you have a solution to your problem
- Using ML (linear regression) learn a function $f : V \to \{0, 1\}$
  $\rightsquigarrow$ predict if a vertex is in MIS or not
- In a sense, we learn new reductions
- Project: apply function to input, afterward optimal algorithm, also transfer this to the weighted case
- Another option: shared-memory parallel weighted MIS local search
- Another option: memetic algorithm for the weighted MIS problem



Dynamic Independent Set of Rectangles

Independet Set of Rectangles



Input: Set of axis-parallel squares in $\mathbb{R}^2$

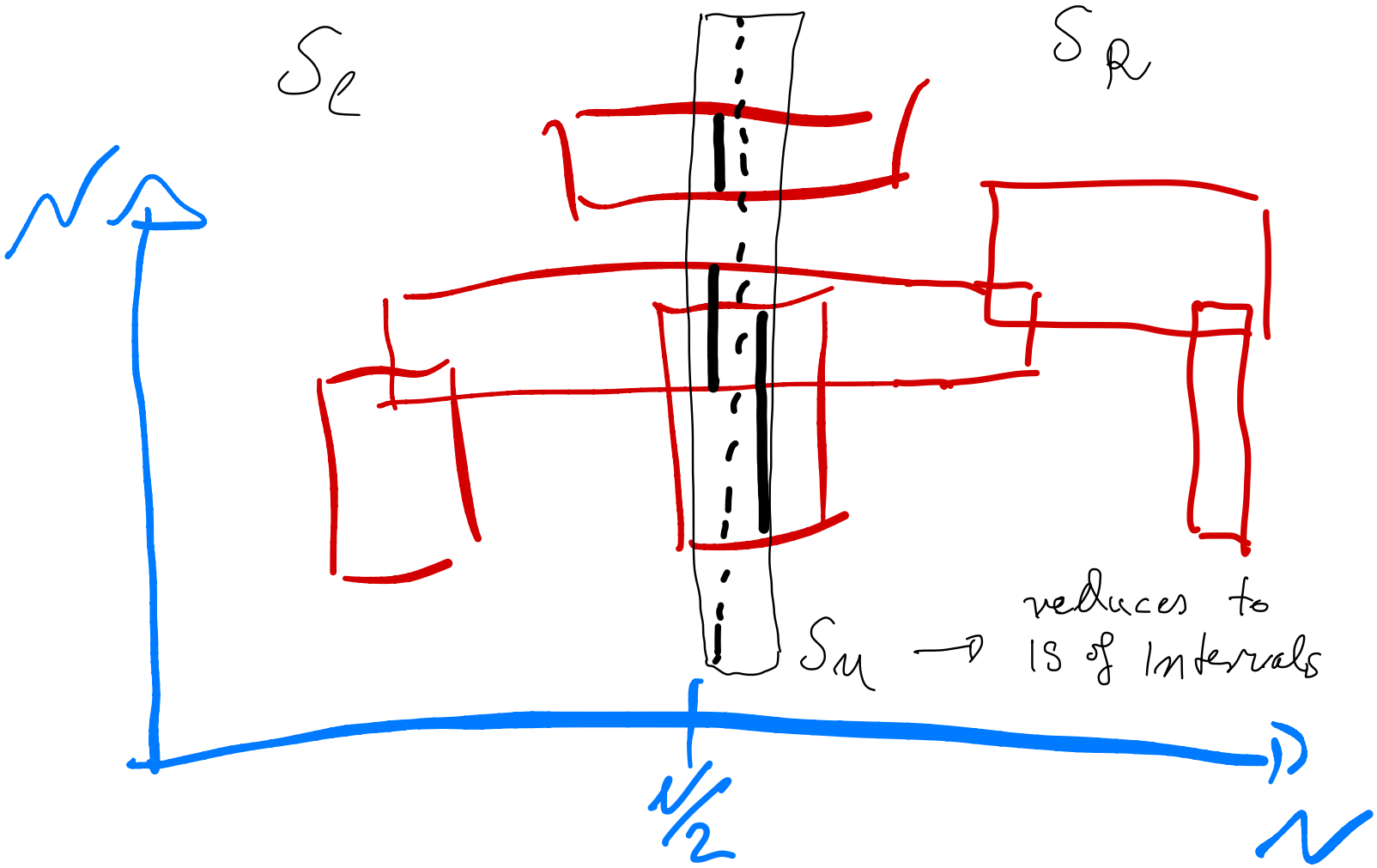
Goal: Find a maximum set of **non-overlapping** squares

NP-hard

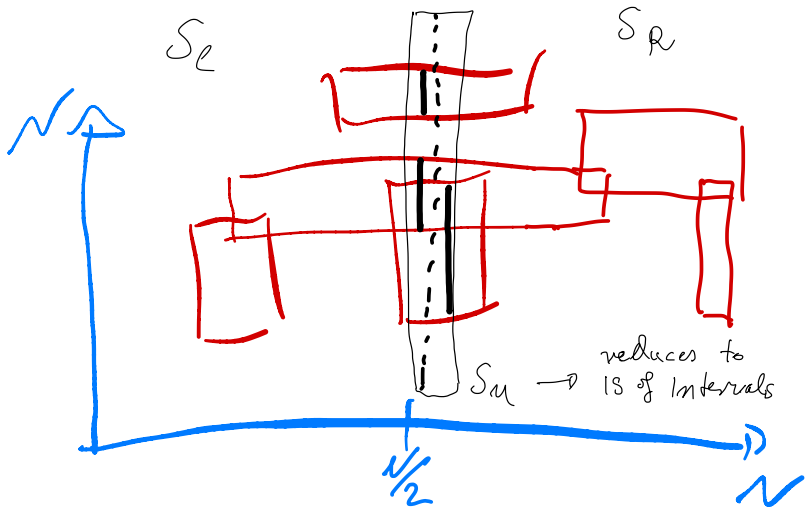
$(1+\epsilon)$-approximation

[Erlebach et al., 2001]
[Chan, 2003]

Algorithm – Rough Idea



Algorithm – Rough Idea



Recurse on $S_l, S_r \rightarrow \log n$ levels
S_m reduces to independent set of intervalls
Dynamic $\rightarrow$ find correct S_m and update solution

[Dynamic Geometric Independent Set:]
`https://arxiv.org/abs/2003.02605`

[Dynamic Map Labeling:]
`https://arxiv.org/abs/2002.07611`