A&W 2026

G-13 Timo Stucki, LFW E 13

Week 12

CodeX Network Tasks

(Min-Cut)

Smallest Enclosing Circle

Convex Hull

Infos

ETH zürich

Feel free to contact me:

- tistucki@student.ethz.ch
- Discord: timostucki

Material:

- timostucki.com

CodeX Network Tasks

The Great Harvest

Applications of Networks and Flows

Recap

Applications of Networks and Flows

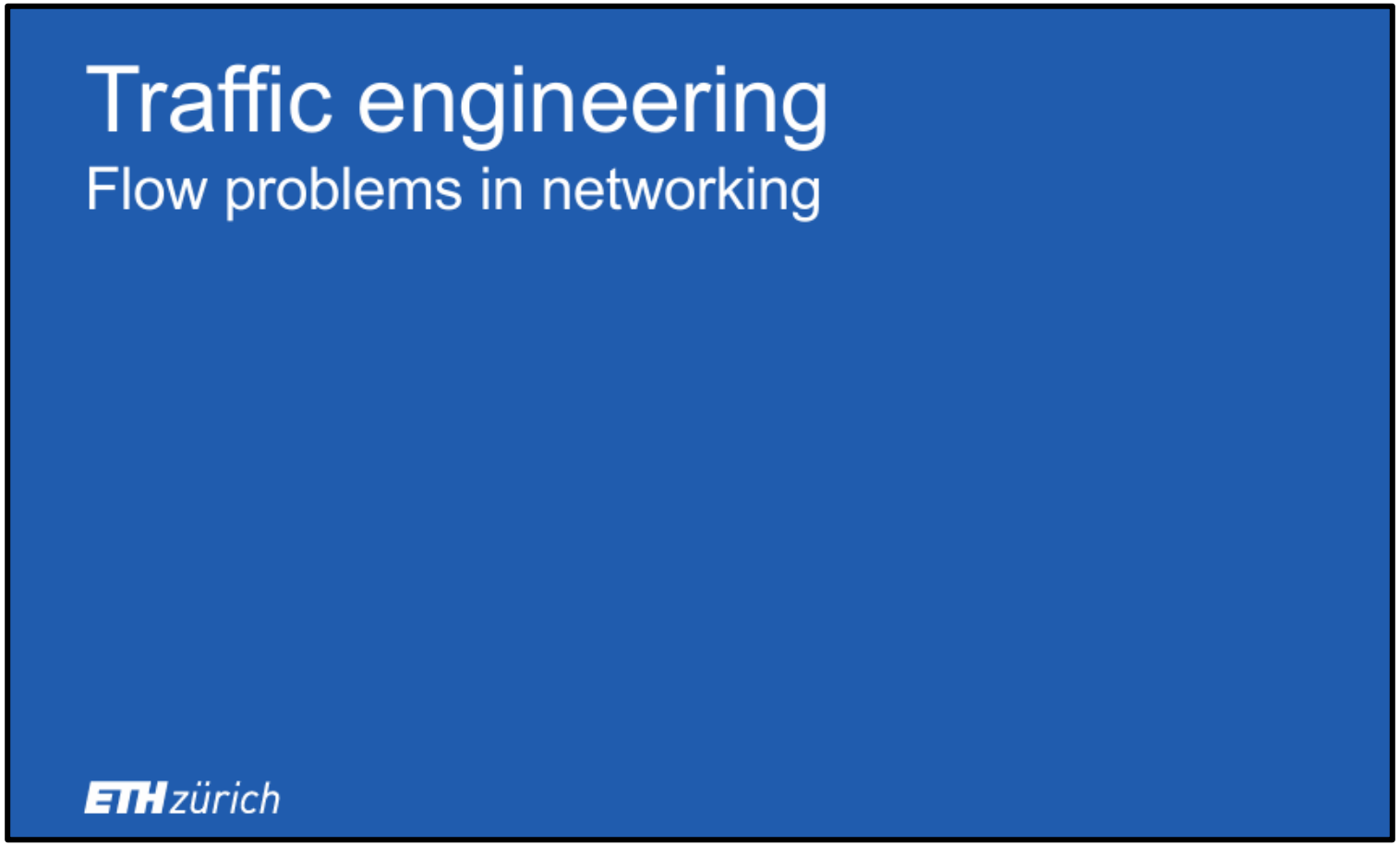
You have seen in lecture:

- Matchings, hier: bipartites maximum Matching in $O(mn)$ (geht besser in $O((m+n)\sqrt{n})$ [Hopcroft&Karp'73]).
- Schnitten zwischen Knoten u und v (Knoten-/Kantenzusammenhang).
- Kantendisjunkten Pfaden (auch knotendisjunkten Pfaden).
- Beweis Satz von Menger (aus Maxflow-Mincut).

Bildsegmentierung. Gegeben ein Bild (aus Pixeln mit Farbwerten), trenne Vordergrund von Hintergrund.

Applications of Networks and Flows

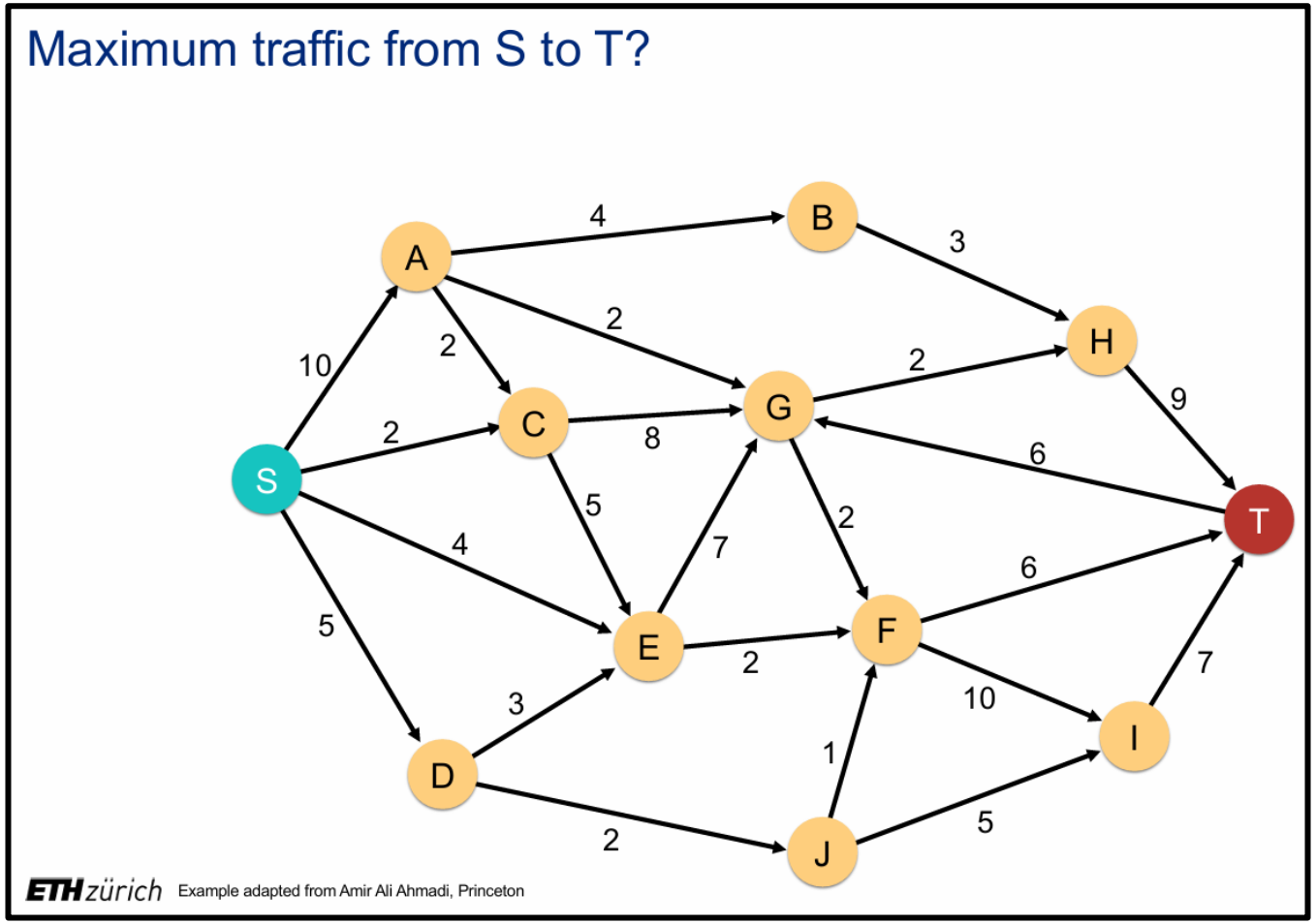
What **you** will 100% use it for:



“Computer Networks” lecture FS26

Applications of Networks and Flows

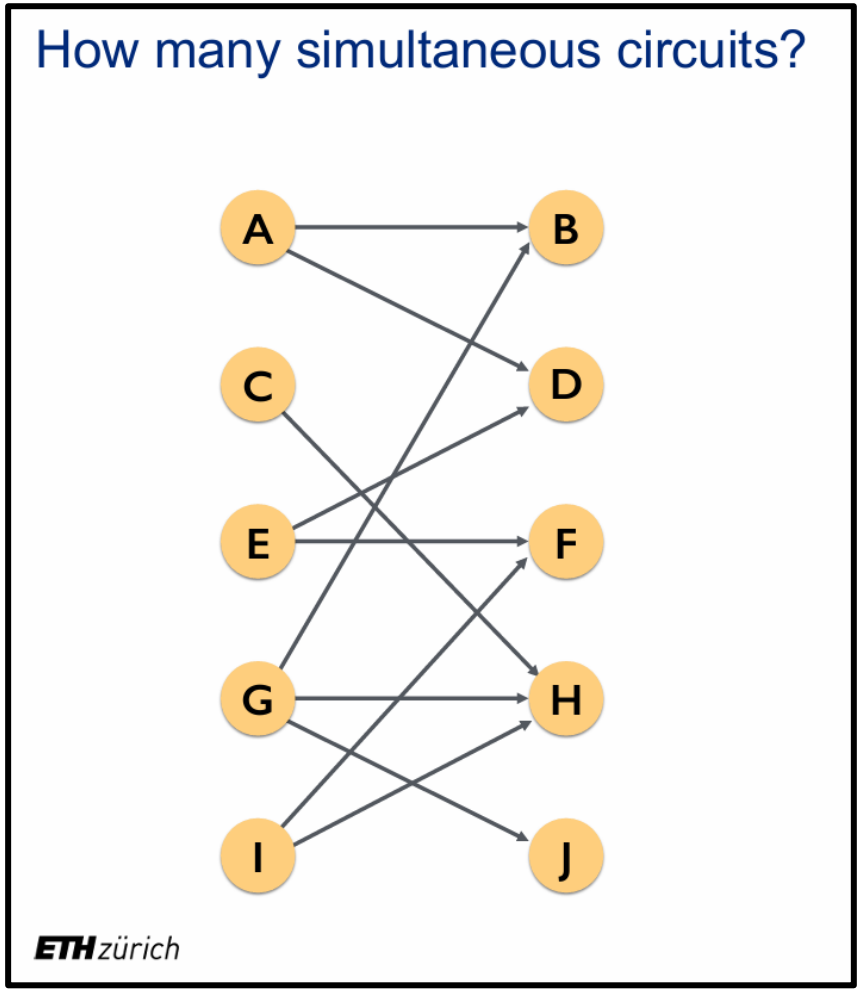
What **you** will 100% use it for:



"Computer Networks" lecture FS26

Applications of Networks and Flows

What **you** will 100% use it for:



"Computer Networks" lecture FS26

Exercise S12

“Integrality and Matching”

Lemma 3.15. Die maximale Grösse eines Matchings im bipartiten Graph G ist gleich dem Wert eines maximalen Flusses im Netzwerk N.

Exercise S12.1 – *Integrality and Matching*

We have already seen Frobenius' Theorem: *Let $k \geq 1$. Every k-regular bipartite graph contains a perfect matching.* The proof we have seen uses Hall's theorem. In this exercise, we want to find another proof using our knowledge about flows.

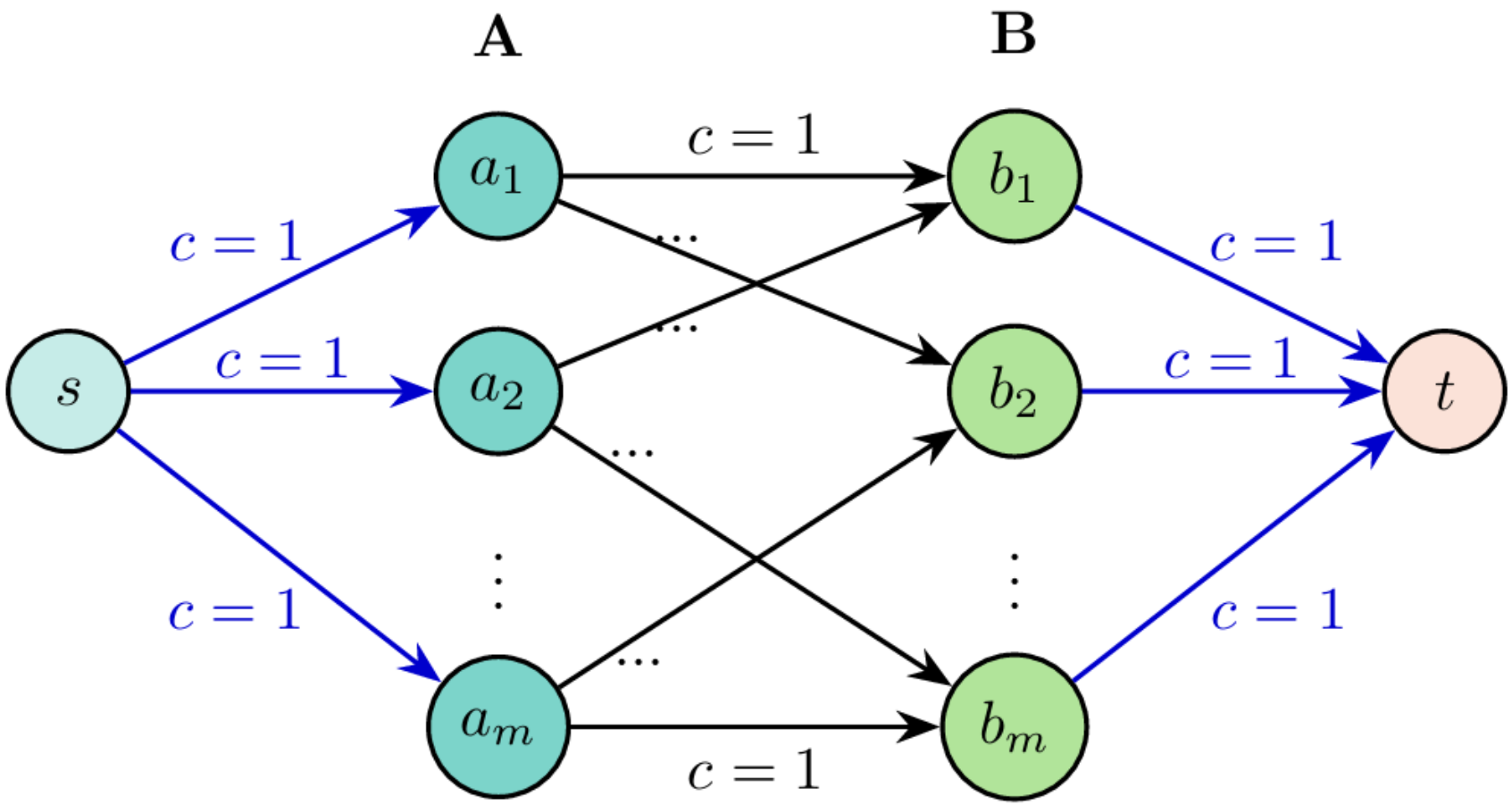
Let $G = (A \dot{\cup} B, E)$ be a k-regular graph for $k \geq 1$.

(a) Describe how to model bipartite matching on G as a flow problem in a network $N = (V, A, c, s, t)$.

(b) Construct explicitly a (not necessarily integer!) flow f on N with $\text{val}(f) = n$. Conclude that $\text{maxflow}(N) = n$.
Hint: you don't have to construct an integral flow. Make sure to define the flow value on all edges of your network N!

(c) Using the result from (b), show that there exists an integer valued flow f' with $\text{val}(f') = n$. Using f', prove that G contains a perfect matching M.

(a) We do the same construction as always: we direct all edges from A to B. Furthermore, we add a source s, and all edges (s, a) for $a \in A$, and we add a sink t and all edges (b, t) for $b \in B$. Each edge gets capacity 1. There is a one to one correspondence between matchings in G and *integral* flows of value $|A| = |B|$ in N. (See Lemma 3.15.)



Exercise S12.1 – *Integrality and Matching*

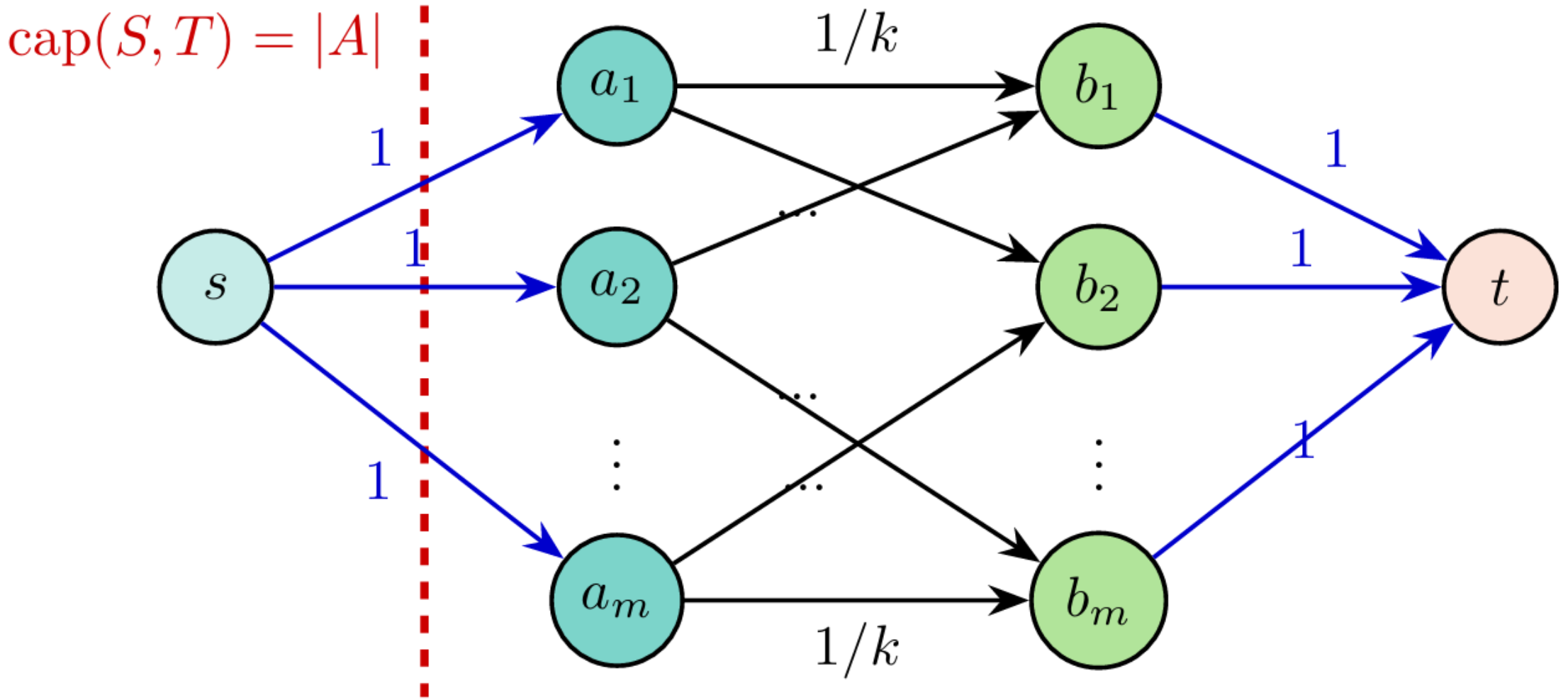
We have already seen Frobenius' Theorem: *Let $k \geq 1$. Every k-regular bipartite graph contains a perfect matching.* The proof we have seen uses Hall's theorem. In this exercise, we want to find another proof using our knowledge about flows.

Let $G = (A \dot{\cup} B, E)$ be a k-regular graph for $k \geq 1$.

(a) Describe how to model bipartite matching on G as a flow problem in a network $N = (V, A, c, s, t)$.

(b) Construct explicitly a (not necessarily integer!) flow f on N with $\text{val}(f) = n$. Conclude that $\text{maxflow}(N) = n$.
Hint: you don't have to construct an integral flow. Make sure to define the flow value on all edges of your network N!

(c) Using the result from (b), show that there exists an integer valued flow f' with $\text{val}(f') = n$. Using f', prove that G contains a perfect matching M.



Each vertex in A has k outgoing edges: $\sum f_{\text{out}} = k \cdot (1/k) = 1.$
Each vertex in B has k incoming edges: $\sum f_{\text{in}} = k \cdot (1/k) = 1.$

(b) For our flow f, we set $f(e) = 1/k$ for all edges between A and B. For the remaining edges, we set the flow to 1. Because all capacities are 1, they are satisfied. Flow conservation is also satisfied: each vertex in A has one incoming edge with flow value 1 and k outgoing edges with flow value $1/k$ each. For vertices in B it is exactly the other way around. Here we used that G is k-regular. The value of this flow equals $|A| = |B|$. The cut $cap(\{s\}, U \cup W \cup \{t\})$ has capacity $|A|$. Thus, f is a maximal flow.

Exercise S12.1 – *Integrality and Matching*

We have already seen Frobenius' Theorem: *Let $k \geq 1$. Every k-regular bipartite graph contains a perfect matching.* The proof we have seen uses Hall's theorem. In this exercise, we want to find another proof using our knowledge about flows.

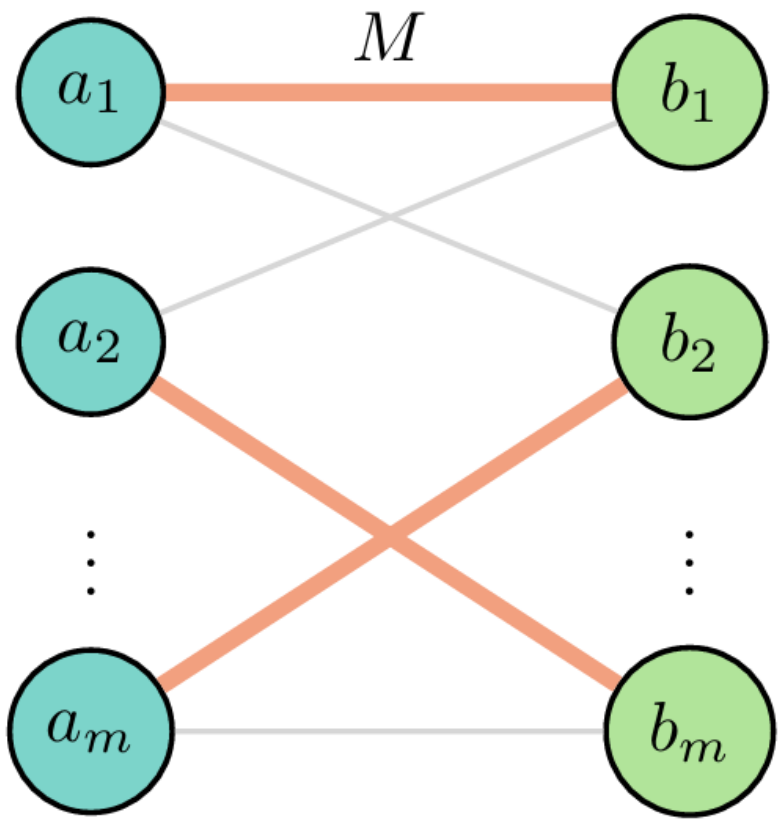
Let $G = (A \dot{\cup} B, E)$ be a k-regular graph for $k \geq 1$.

(a) Describe how to model bipartite matching on G as a flow problem in a network $N = (V, A, c, s, t)$.

(b) Construct explicitly a (not necessarily integer!) flow f on N with $\text{val}(f) = n$. Conclude that $\text{maxflow}(N) = n$.
Hint: you don't have to construct an integral flow. Make sure to define the flow value on all edges of your network N!

(c) Using the result from (b), show that there exists an integer valued flow f' with $\text{val}(f') = n$. Using f', prove that G contains a perfect matching M.

(c) By Theorem 3.12, the maximum flow in N and the maximum integral flow in N have the same value. Thus, by (b), there is an integral flow of value $|A|$. By Lemma 3.15, G has a matching of size $|A| = |B|$. This is already a perfect matching.



Algorithms – Highlights

Algorithms – Highlights

Algorithms – Highlights

Randomised

Deterministic

Algorithms – Highlights

Randomised

Deterministic

Min-Cut

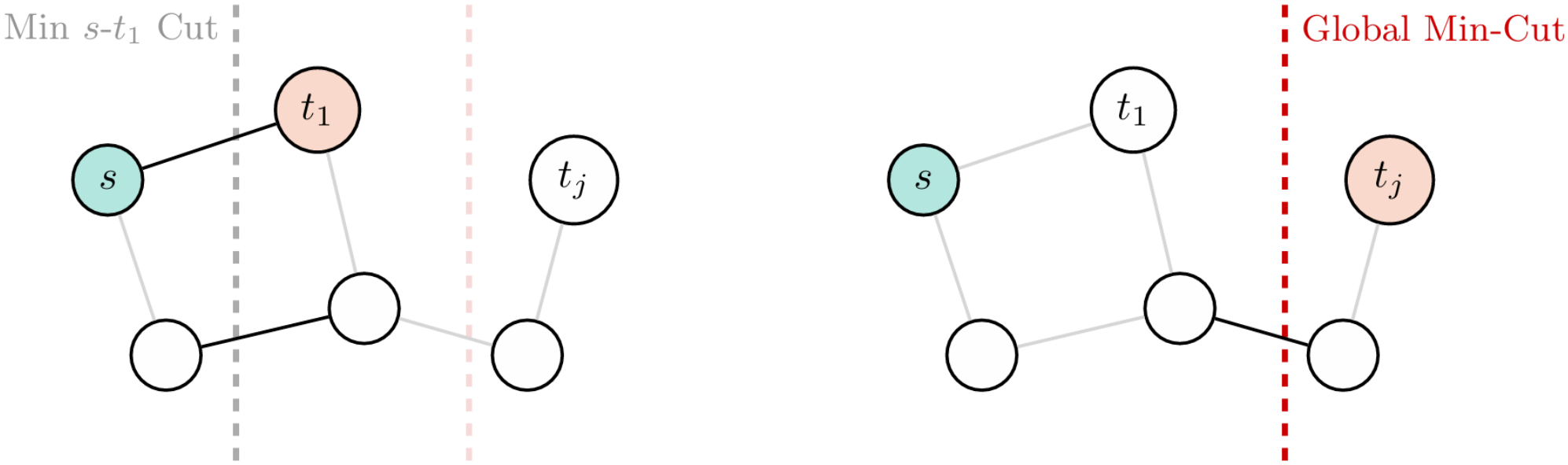
(Recap)

Min-Cut

8. The MIN-CUT problem can be solved by fixing one vertex s and computing minimum s-t cuts for all $t \in V \setminus \{s\}$.

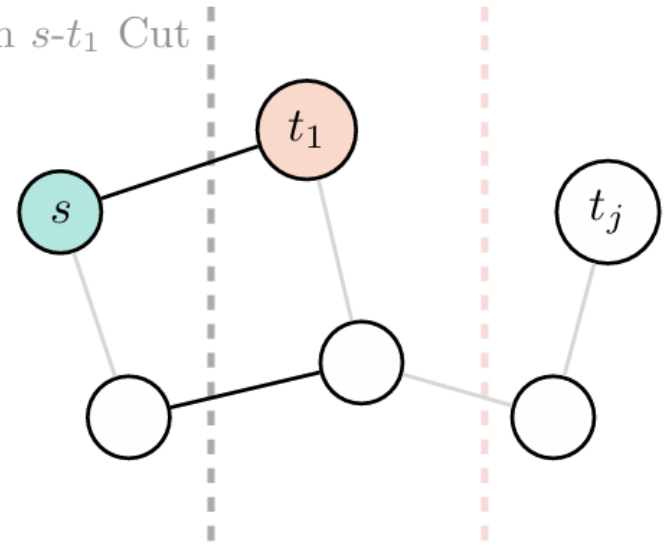
Answer: TRUE

Explanation. [Script page 192] Inelegant illustration:

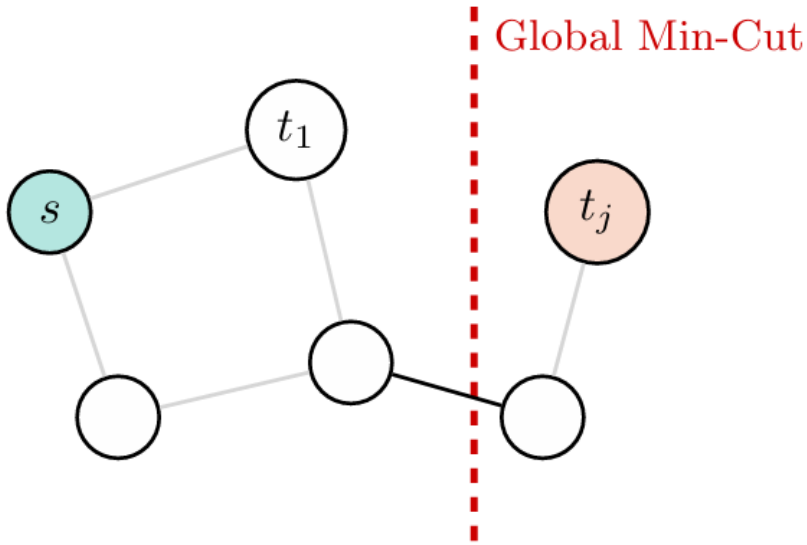


Min-Cut

Explanation. [Script page 192] Inelegant illustration:



Iteration 1: Trying with t_1
s, t_1 are on **different** sides of cut.
Cut size $= 2$
(we first need to try with all
other t's to see if this is $\mu(\mathbf{G})$)



Iteration j: Trying with t_j
s, t_1 are on **different** sides of cut.
Cut size $= 1 = \mu(\mathbf{G})$

Min-Cut

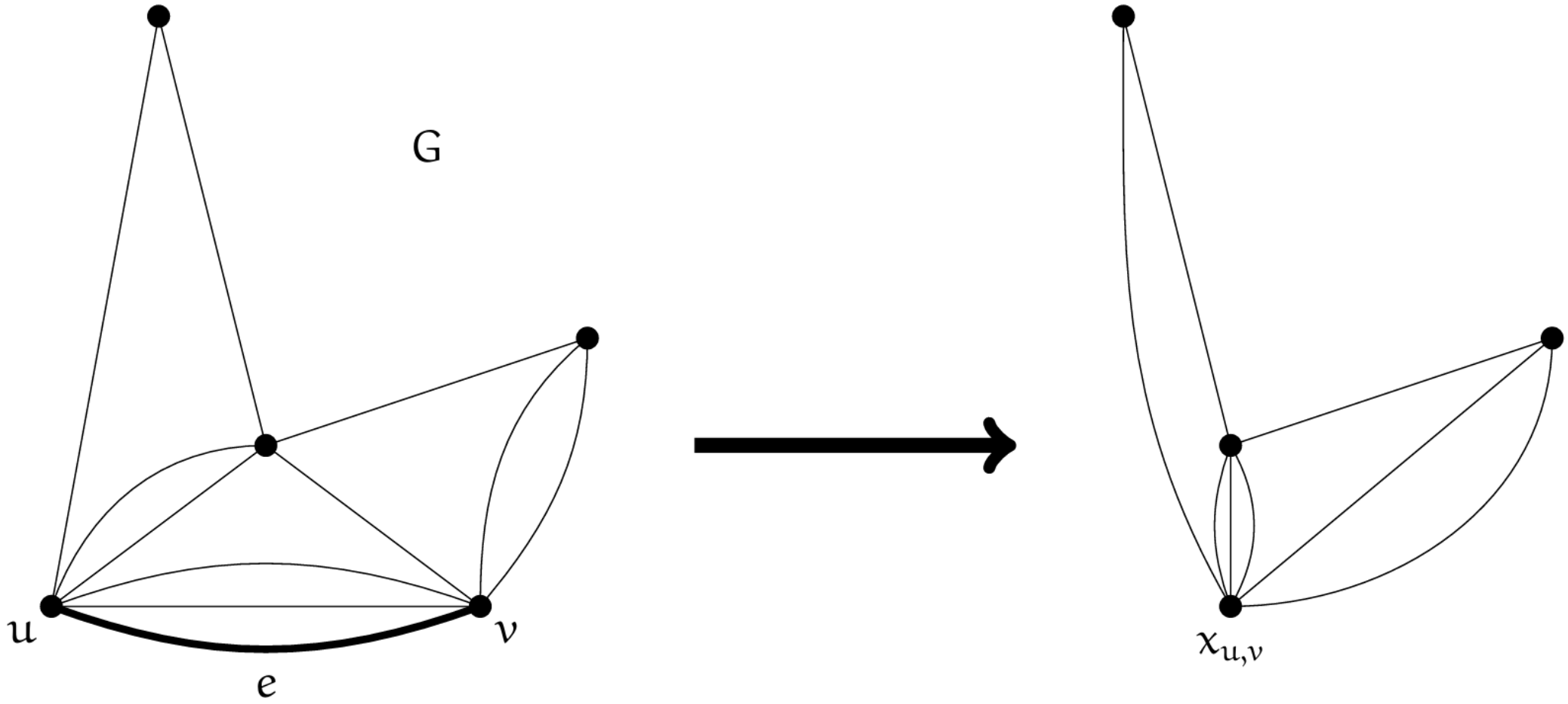
Runtime of finding maximum flow in $O(nm \log n) = O(n^3 \log n)$ using Dynamic Trees n-1 times:

$$O(n^4 \log n)$$

→ **Use randomised algorithm to be faster than this** 🎉 🎉 🎉

Min-Cut

Edge contraction:

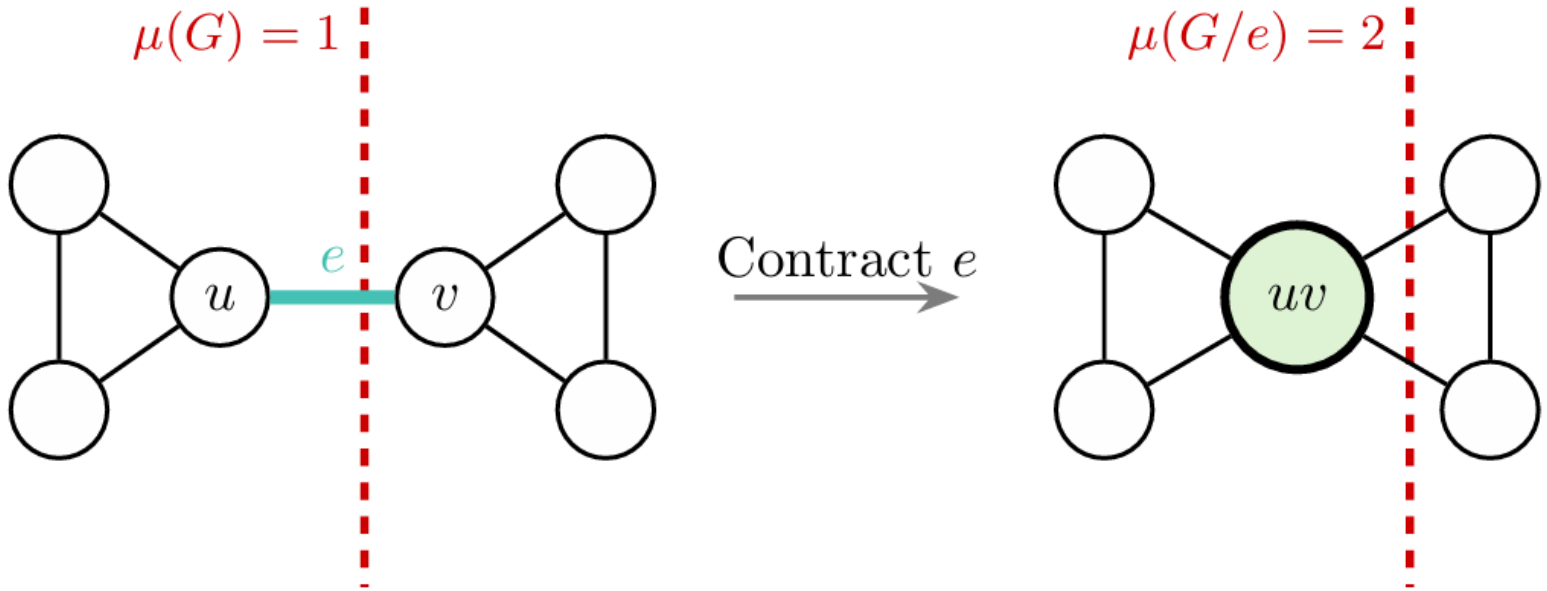


Min-Cut

9. If e is an edge of G, then always $\mu(G/e) \leq \mu(G)$.

Answer: FALSE

Explanation. Here $\mu(G)$ denotes the size of the minimum cut (edge connectivity) of the graph. **Counterexample:**



Original Graph G **Contracted Graph G/e**

Min-Cut

Lemma 3.20. Sei G ein Graph und e eine Kante in G. Dann gilt $\mu(G/e) \geq \mu(G)$ und falls es in G einen minimalen Schnitt C mit $e \notin C$ gibt, dann gilt $\mu(G/e) = \mu(G)$.

Min-Cut

CUT(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > 2$ **do**
2: $\quad e \leftarrow$ gleichverteilt zufällige Kante in G
3: $\quad G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G

Min-Cut

Lemma 3.21. Sei $G = (V, E)$ ein Multigraph mit n Knoten. Falls e gleichverteilt zufällig unter den Kanten in G gewählt wird, dann gilt

$$\Pr[\mu(G) = \mu(G/e)] \geq 1 - \frac{2}{n}.$$

Beweis. Sei C ein minimaler Schnitt in G und sei $k := |C| = \mu(G)$. Sicherlich ist der Grad jedes Knotens in G mindestens k, da die zu einem Knoten inzidenten Kanten immer einen Schnitt bilden. Es gilt daher $|E| = \frac{1}{2}\sum_{v \in V} \deg(v) \geq \frac{kn}{2}$. Wir erinnern uns an $e \notin C \Rightarrow \mu(G/e) = \mu(G)$ und somit

$$\Pr[\mu(G) = \mu(G/e)] \geq \Pr[e \notin C] = 1 - \frac{|C|}{|E|} \geq 1 - \frac{k}{kn/2} = 1 - \frac{2}{n},$$

was zu zeigen war. □

Min-Cut

$\hat{p}(G) :=$ Wahrscheinlichkeit, dass $\textsc{Cut}(G)$ den Wert $\mu(G)$ ausgibt und

$$\hat{p}(n) := \inf_{G=(V,E),\,|V|=n} \hat{p}(G) \ .$$

Man beachte, dass $\hat{p}(2) = 1$.

Lemma 3.22. Es gilt für alle $n \geq 3$

$$\hat{p}(n) \geq (1 - 2/n) \cdot \hat{p}(n-1).$$

Min-Cut

$$\hat{p}(n) \geq \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdots \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} \cdot \underbrace{\hat{p}(2)}_{=1} = \frac{2}{n(n-1)}$$

Lemma 3.23. Es gilt $\hat{p}(n) \geq \frac{2}{n(n-1)} = 1/\binom{n}{2}$ für alle $n \geq 2$.

Min-Cut

CUT(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > 2$ **do**
2: $\quad e \leftarrow$ gleichverteilt zufällige Kante in G
3: $\quad G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G

Min-Cut

Satz 3.24. Für den Algorithmus der $\lambda\binom{n}{2}$-maligen Wiederholung von $\textsc{Cut}(G)$ gilt:

(1) Der Algorithmus hat eine Laufzeit von $O(\lambda n^4)$.

(2) Der kleinste angetroffene Wert ist mit einer Wahrscheinlichkeit von mindestens $1 - e^{-\lambda}$ gleich $\mu(G)$.

Min-Cut

Runtime of finding maximum flow in $O(nm \log n) = O(n^3 \log n)$ using Dynamic Trees n-1 times:

$$O(n^4 \log n)$$

→ **Use randomised algorithm to be faster than this** 🎉 🎉 🎉

Bootstrapping

$z(t) = O(t^4 \log t)$ Zeit mit $p^*(t) = 1$, oder nach Satz 3.24 (mit $\lambda = 1$) in $z(t) = O(t^4)$ Zeit mit $p^*(t) = 1 - e^{-1}$).

CUT1(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > t$ **do**
2: $e \leftarrow$ gleichverteilt zufällige Kante in G
3: $G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G ▷ in Zeit $O(z(t))$

Min-Cut

Bootstrapping

$z(t) = O(t^4 \log t)$ Zeit mit $p^*(t) = 1$, oder nach Satz 3.24 (mit $\lambda = 1$) in $z(t) = O(t^4)$ Zeit mit $p^*(t) = 1 - e^{-1}$).

CUT1(G) — G zusammenhängender Multigraph

1: **while** $|V(G)| > t$ **do**
2: $e \leftarrow$ gleichverteilt zufällige Kante in G
3: $G \leftarrow G/e$
4: **return** Grösse des eindeutigen Schnitts in G ▷ in Zeit $O(z(t))$

Min-Cut

Now we repeat this process, and eventually:

> Wo führt das hin? Es konvergiert zu einem Verfahren mit einer Laufzeit von $O(n^2 \text{poly}(\log n))$, man kann sich das wie einen „Grenzwertalgorithmus“ vorstellen.

(Details in lecture/script on page 197)

We achieve **$O(n^2 \text{poly}(\log n))$** runtime 🎉 🎉 🎉

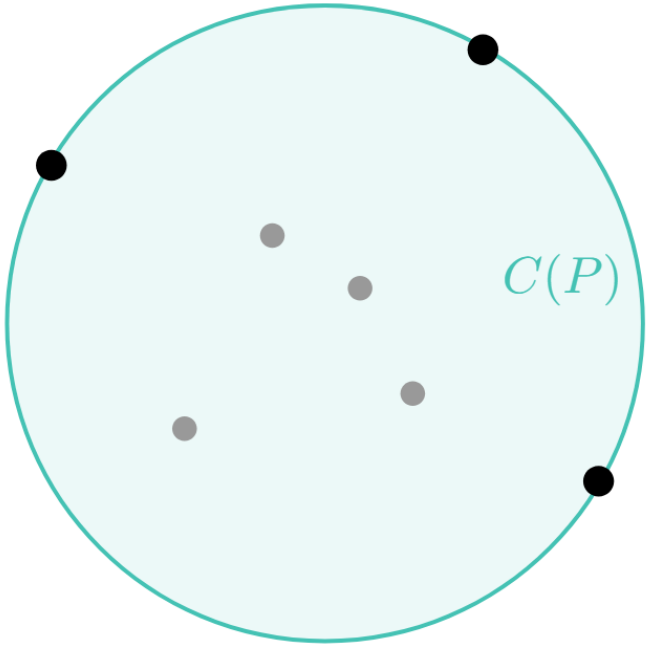
Key: Cut(G) always looses the most success probability in the last few steps and by iterating over these separately (bootstrapping), we don’t have to redo the whole process at every failure in the last steps.

Smallest Enclosing Circle

short recap

Smallest Enclosing Circle

Lemma 3.25. Für jede (endliche) Punktemenge P im $\mathbb{R}^2$ gibt es einen eindeutigen kleinsten umschliessenden Kreis C(P).

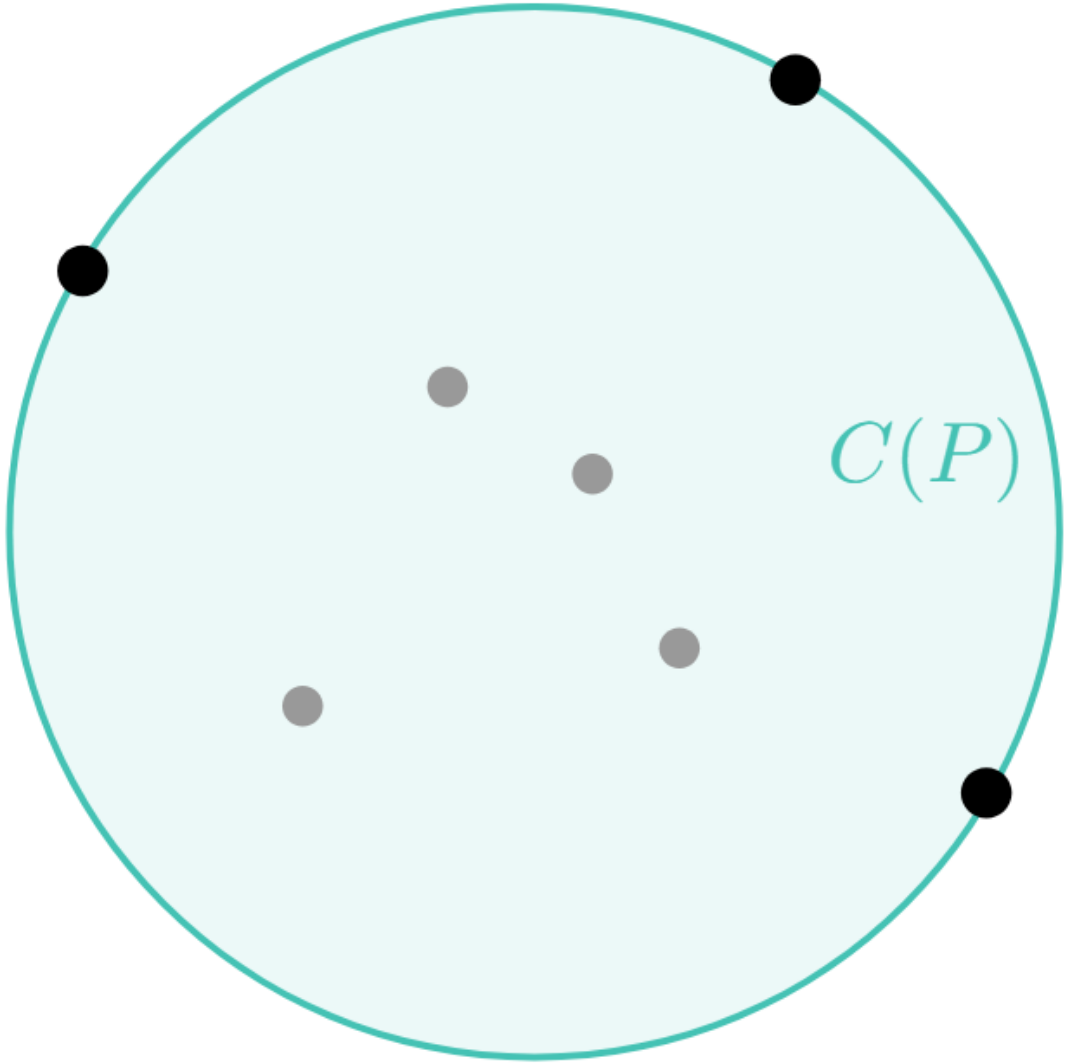


Smallest Enclosing Circle

Lemma 3.25. Für jede (endliche) Punktemenge P im $\mathbb{R}^2$ gibt es einen eindeutigen kleinsten umschliessenden Kreis $C(P)$.

Lemma 3.26. Für jede (endliche) Punktemenge P im $\mathbb{R}^2$ mit $|P| \geq 3$ gibt es eine Teilmenge $Q \subseteq P$, so dass $|Q| = 3$ und $C(Q) = C(P)$.

Smallest Enclosing Circle



Smallest Enclosing Circle

CompleteEnumeration(P)

1: **for all** $Q \subseteq P$ mit $|Q| = 3$ **do**
2: bestimme $C(Q)$
3: **if** $P \subseteq C^{\bullet}(Q)$ **then**
4: **return** $C(Q)$

Smallest Enclosing Circle

CompleteEnumeration(P)

1: **for all** $Q \subseteq P$ mit $|Q| = 3$ **do**
2: bestimme $C(Q)$
3: **if** $P \subseteq C^{\bullet}(Q)$ **then** ← check in O(n)
4: **return** $C(Q)$

Runtime: $O(n^4)$

Smallest Enclosing Circle

RANDOMISED_PRIMITIVEVERSION(P)

```
1: repeat forever
2:     wähle Q ⊆ P mit |Q| = 3 zufällig und gleichverteilt
3:     bestimme C(Q)
4:     if P ⊆ C•(Q) then
5:         return C(Q)
```

Smallest Enclosing Circle

```
RANDOMISED_PRIMITIVEVERSION(P)
1: repeat forever
2:     wähle Q ⊆ P mit |Q| = 3 zufällig und gleichverteilt
3:     bestimme C(Q)
4:     if P ⊆ C•(Q) then ← check in O(n)
5:         return C(Q)
```

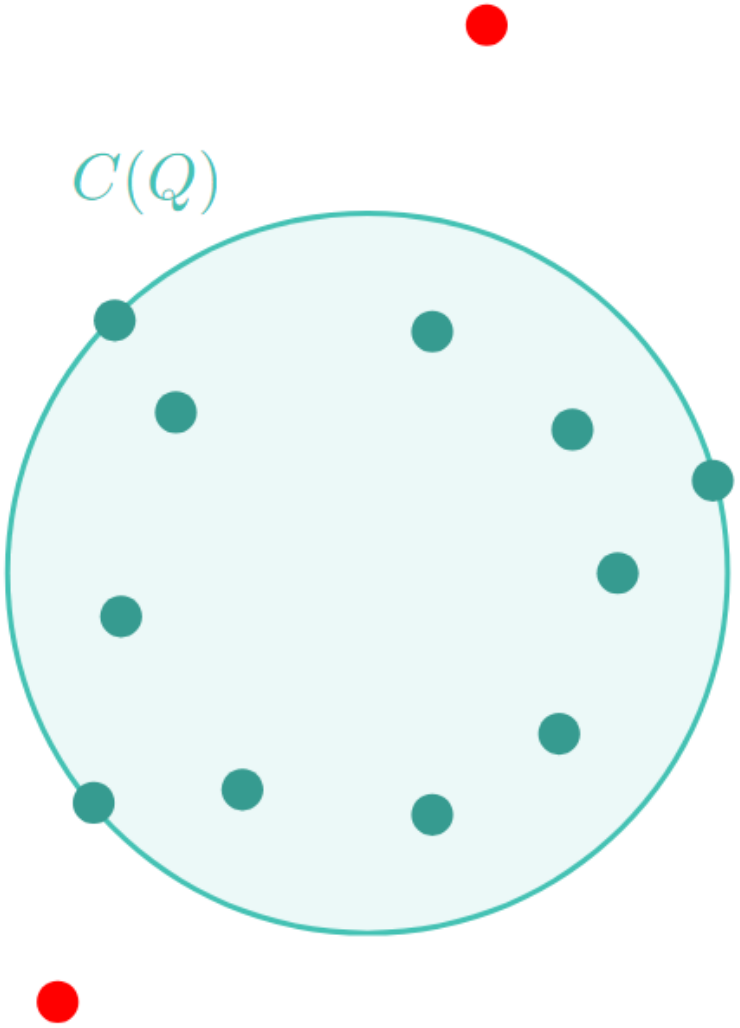
Expected runtime: $O(n^4)$

Smallest Enclosing Circle

RANDOMISED_CLEVERVERSION(P)

1: **repeat forever**
2: wähle $Q \subseteq P$ mit $|Q| = 11$ zufällig und gleichverteilt
3: bestimme $C(Q)$
4: **if** $P \subseteq C^{\bullet}(Q)$ **then**
5: **return** $C(Q)$
6: verdoppele alle Punkte von P ausserhalb von $C(Q)$

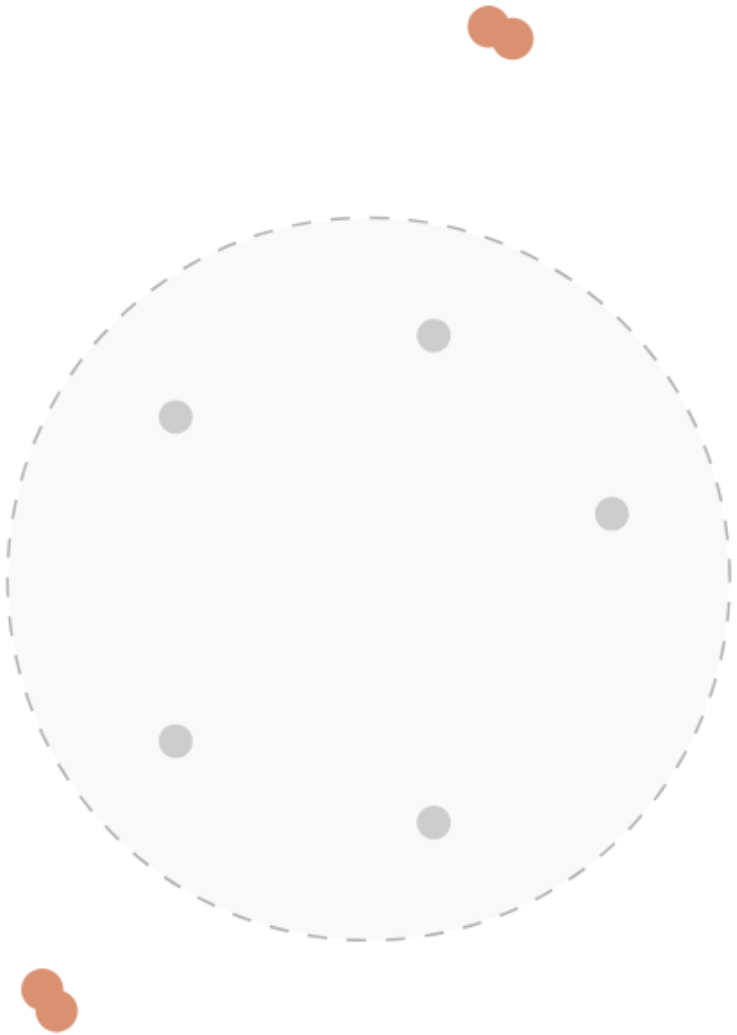
Smallest Enclosing Circle



$p_i \notin C(Q)$

"Sampling Step" ($|Q| = 11$)
Blue dots denote sampled subset Q.
Red dots denote outliers violating the condition $P \subseteq C^{\bullet}(Q)$.

Smallest Enclosing Circle



×2 "weight"

Every point outside $C(Q)$ is doubled, making it more likely that they are chosen for Q in the next iteration(s).

Smallest Enclosing Circle

```
RANDOMISED_CLEVERVERSION(P)
1: repeat forever
2:     wähle Q ⊆ P mit |Q| = 11 zufällig und gleichverteilt
3:     bestimme C(Q)                          ← in O(1) !
4:     if P ⊆ C•(Q) then                      ← in O(n)
5:         return C(Q)
6:     verdoppele alle Punkte von P ausserhalb von C(Q)
```

Expected # iterations: O(logn) 🎉 🎉 🎉

Smallest Enclosing Circle

RANDOMISED_CLEVERVERSION(P)

1: **repeat forever**
2: wähle $Q \subseteq P$ mit $|Q| = 11$ zufällig und gleichverteilt
3: bestimme $C(Q)$
4: **if** $P \subseteq C^{\bullet}(Q)$ **then**
5: **return** $C(Q)$
6: verdoppele alle Punkte von P ausserhalb von $C(Q)$

Satz 3.29. Algorithmus RANDOMIZED_CLEVERVERSION berechnet den kleinsten umschliessenden Kreis von P in erwarteter Zeit $O(n \log n)$.

Smallest Enclosing Circle

```
RANDOMISED_CLEVERVERSION(P)
1: repeat forever
2:     wähle Q ⊆ P mit |Q| = 11 zufällig und gleichverteilt
3:     bestimme C(Q)
4:     if P ⊆ C•(Q) then
5:         return C(Q)
6:     verdoppele alle Punkte von P ausserhalb von C(Q)
```

Why |Q|=11? allows us to prove expected # iterations is in O(logn)
(could also use any number > 11, theoretically)

Consider the lecture and script to further recap the proofs.

Smallest Enclosing Circle

RANDOMISED_CLEVERVERSION(P)

```
1: repeat forever
2:     wähle Q ⊆ P mit |Q| = 11 zufällig und gleichverteilt
3:     bestimme C(Q)
4:     if P ⊆ C•(Q) then
5:         return C(Q)
6:     verdoppele alle Punkte von P ausserhalb von C(Q)
```

Which type of randomised algorithm is this?

Convex Hull

short recap

Convex Hull

Sei $d \in \mathbb{N}$.

- Für $v_0, v_1 \in \mathbb{R}^d$ sei

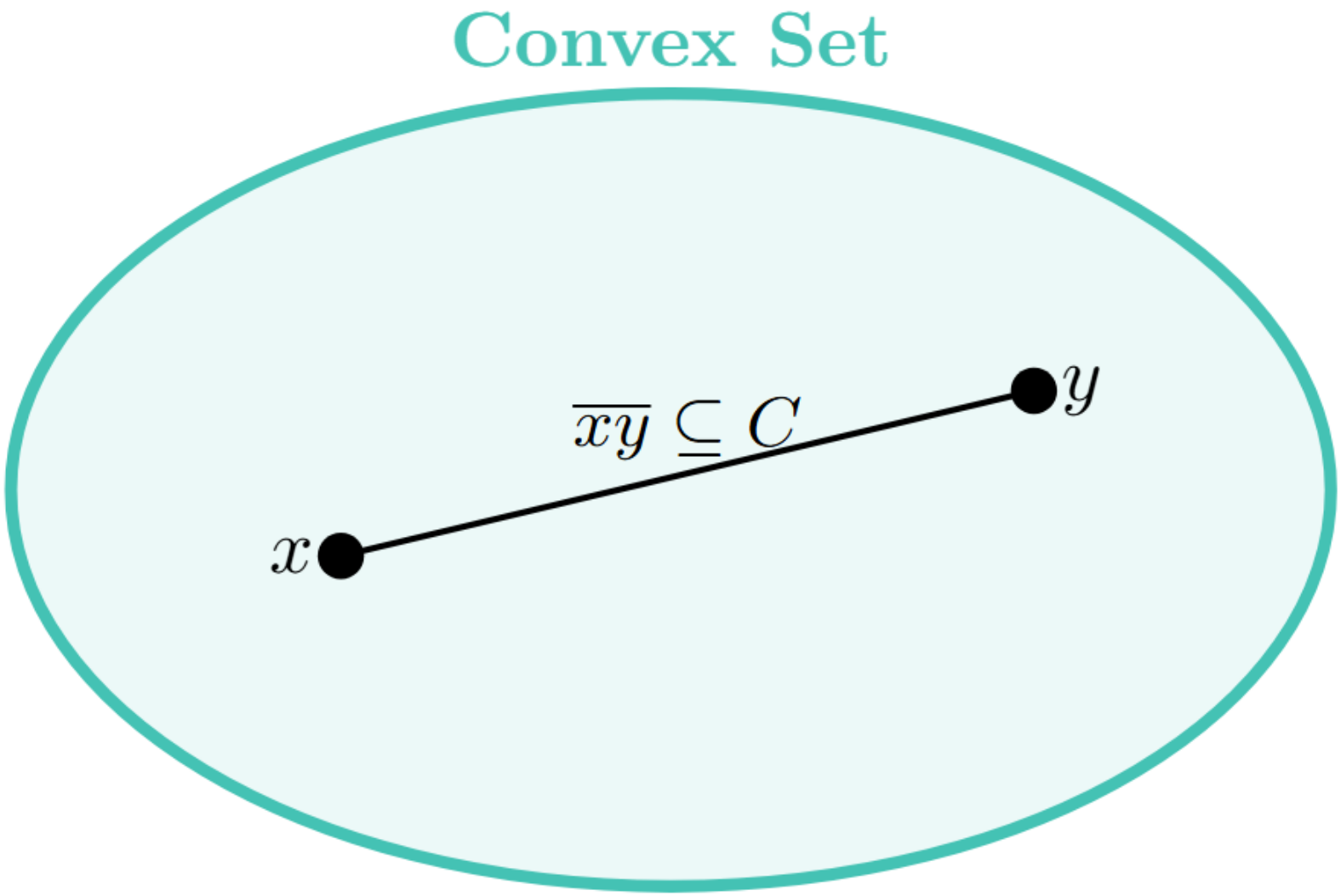
$$\overline{v_0 v_1} := \{(1-\lambda)v_0 + \lambda v_1 \mid \lambda \in \mathbb{R}, 0 \leq \lambda \leq 1\} ,$$

  das v_0 und v_1 verbindende **Liniensegment**.

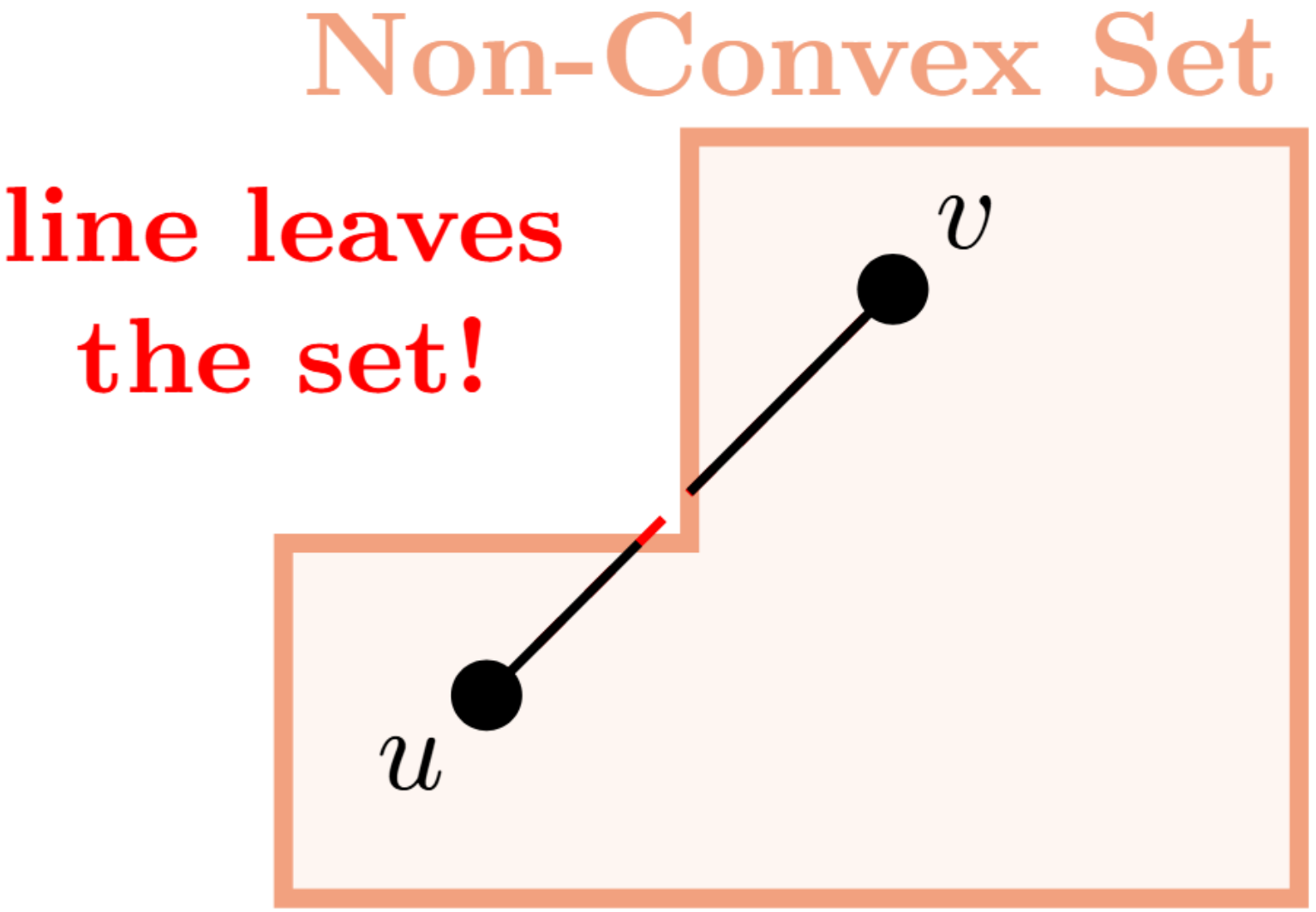
- Eine Menge $C \subseteq \mathbb{R}^d$ heisst **konvex**, falls

$$\forall v_0, v_1 \in C: \ \overline{v_0 v_1} \subseteq C .$$

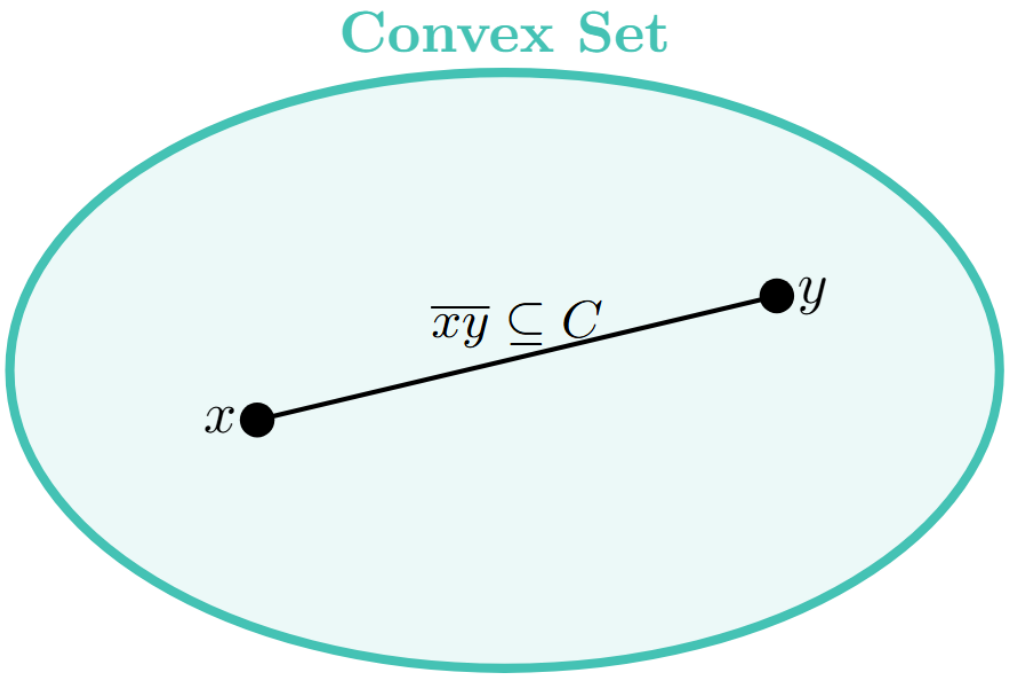
Convex Hull

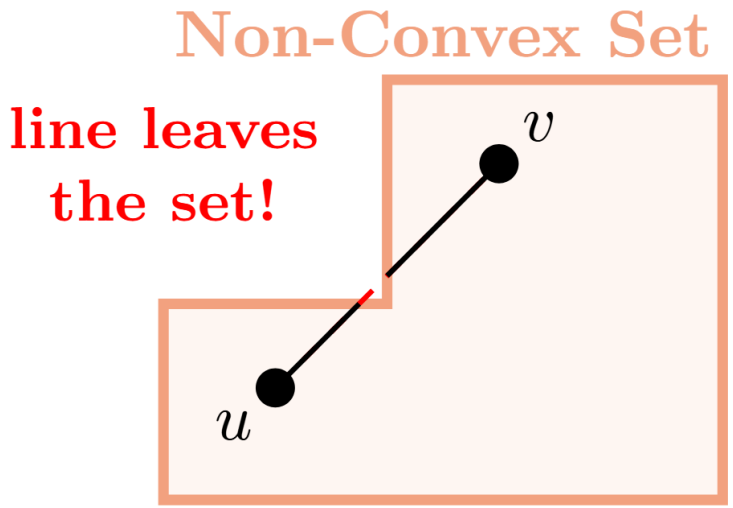


Convex Hull



Convex Hull


Convex Set
$\overline{xy} \subseteq C$
x
y


Non-Convex Set
line leaves
the set!
v
u

Convex Hull

Definition 3.33. Sei $S \subseteq \mathbb{R}^d$, $d \in \mathbb{N}$. Die *konvexe Hülle*, $\mathrm{conv}(S)$, *von* S ist der Schnitt aller konvexen Mengen, die S enthalten, d.h.

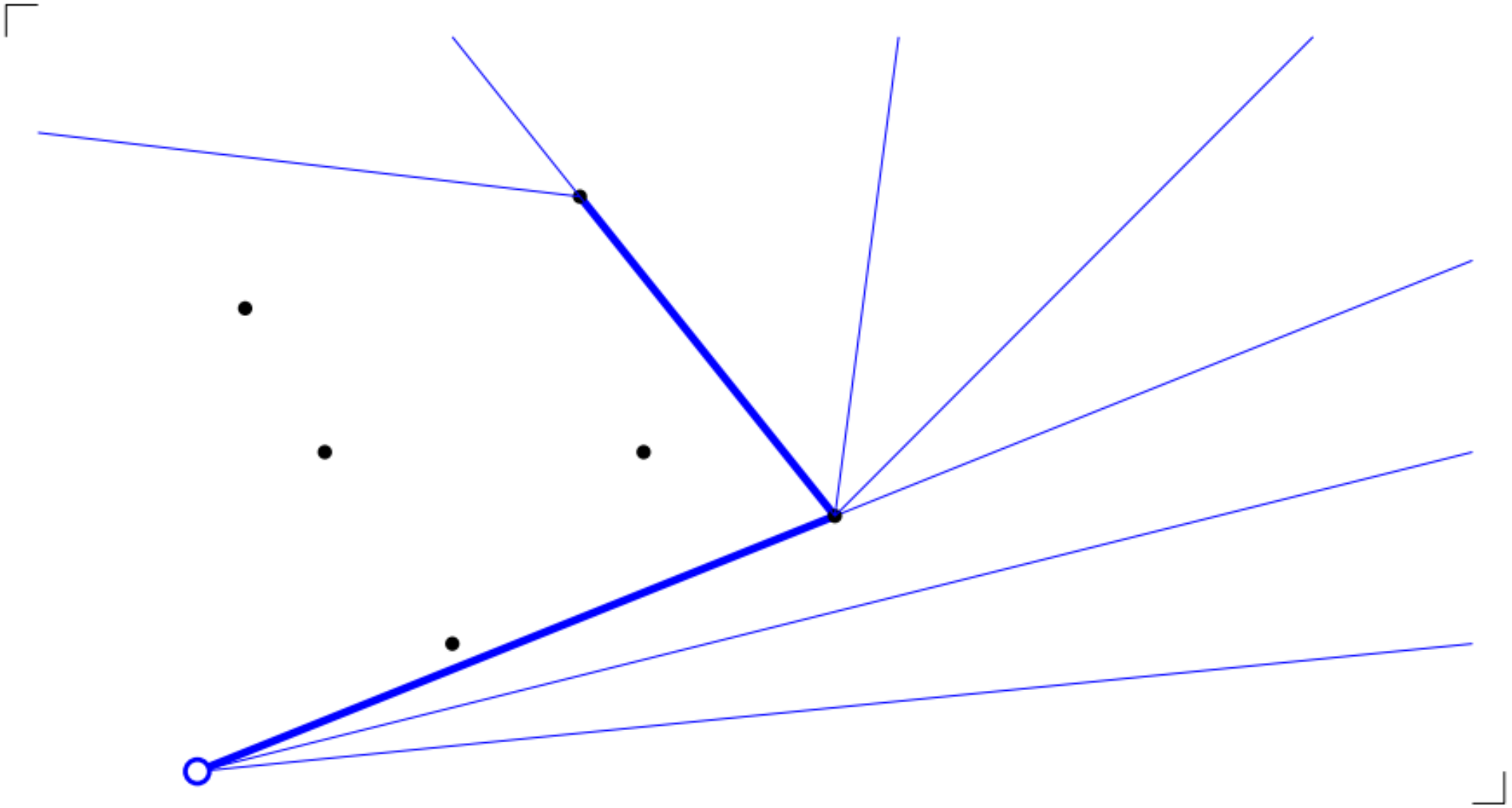
$$\mathrm{conv}(S) := \bigcap_{S \subseteq C \subseteq \mathbb{R}^d,\, C \text{ konvex}} C \,.$$

Convex Hull

Vereinfachende Annahme: **Allgemeine Lage**, d.h. keine 3 Punkte auf einer gemeinsamen Geraden, keine 2 Pkt gleiche x-Koordinate.

Jarvis Wrap

Jarvis Wrap



Jarvis Wrap

JarvisWrap(P)

1: $h \leftarrow 0$
2: $p_{now} \leftarrow$ Punkt in P mit kleinster x-Koordinate
3: **repeat**
4: $\quad q_h \leftarrow p_{now}$
5: $\quad p_{now} \leftarrow \text{FindNext}(q_h)$
6: $\quad h \leftarrow h + 1$
7: **until** $p_{now} = q_0$
8: **return** $(q_0, q_1, \ldots, q_{h-1})$

Jarvis Wrap

JarvisWrap(P)

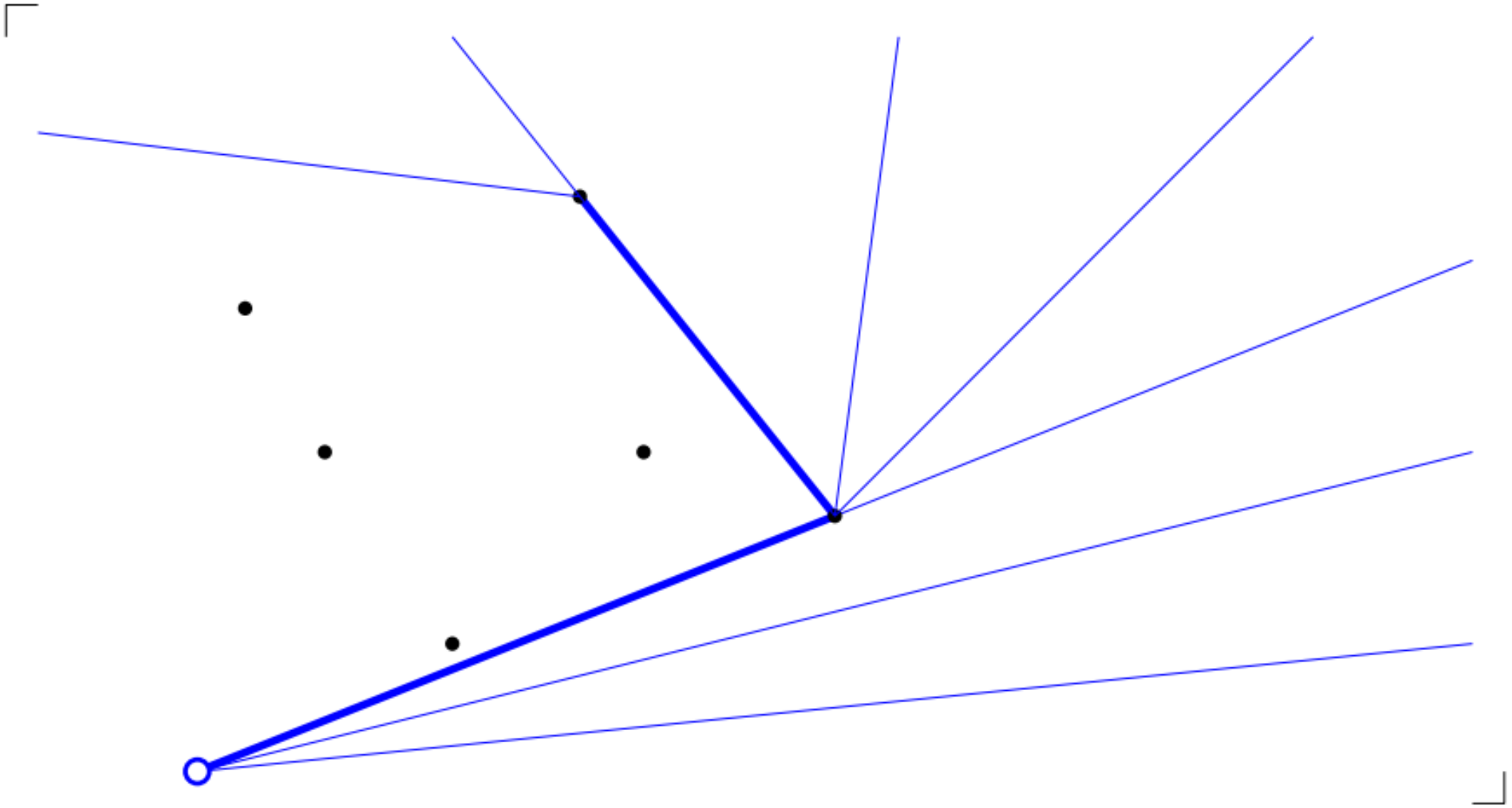
1: $h \leftarrow 0$
2: $p_{now} \leftarrow$ Punkt in P mit kleinster x-Koordinate
3: **repeat**
4: $\quad q_h \leftarrow p_{now}$
5: $\quad p_{now} \leftarrow \text{FindNext}(q_h)$
6: $\quad h \leftarrow h + 1$
7: **until** $p_{now} = q_0$
8: **return** $(q_0, q_1, \ldots, q_{h-1})$ ← corners

Jarvis Wrap

FindNext(q)

1: Wähle $p_0 \in P \setminus \{q\}$ beliebig
2: $q_{next} \leftarrow p_0$
3: **for all** $p \in P \setminus \{q, p_0\}$ **do**
4: **if** p rechts von $q q_{next}$ **then** $q_{next} \leftarrow p$
5: **return** q_{next}

Jarvis Wrap



Jarvis Wrap

```
JarvisWrap(P)
1: h ← 0
2: p_now ← Punkt in P mit kleinster x-Koordinate
3: repeat
4:     q_h ← p_now
5:     p_now ← FindNext(q_h)
6:     h ← h + 1
7: until p_now = q_0
8: return (q_0, q_1, ..., q_{h-1})
```

Satz 3.37. Gegeben eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus JarvisWrap die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.

Jarvis Wrap

```
JarvisWrap(P)
1: h ← 0
2: p_now ← Punkt in P mit kleinster x-Koordinate
3: repeat
4:     q_h ← p_now
5:     p_now ← FindNext(q_h)
6:     h ← h + 1
7: until p_now = q_0
8: return (q_0, q_1, ..., q_{h-1})
```

Satz 3.37. Gegeben eine Menge P von n Punkten in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus JarvisWrap die konvexe Hülle in Zeit $O(nh)$, wobei h die Anzahl der Ecken der konvexen Hülle von P ist.

Jarvis Wrap

Kollinearitäten (3 Punkte auf Gerade), **gleiche x-Koord.**, ...

- Anfangspunkt q_0 als den Punkt mit lexikographisch kleinster Koordinate (unter allen mit kleinster x-Koordinate, den mit kleinster y-Koordinate). (Adaption der x-Ordnung der Punkte)
- Der Test "p rechts von q, q_{next}" muss ersetzt werden durch (p rechts von q, q_{next}) oder (p auf der Geraden durch q, q_{next} und $|qp| > |qq_{\text{next}}|$). (Adaption der Ordnung $\prec_q$)
- In der Regel können wir nicht einmal annehmen, dass die Punkte verschieden sind (z.B. gegeben in einem Feld)!

Software ohne Berücksichtigung dieser Fälle hat geringen Nutzen!

Local Repair

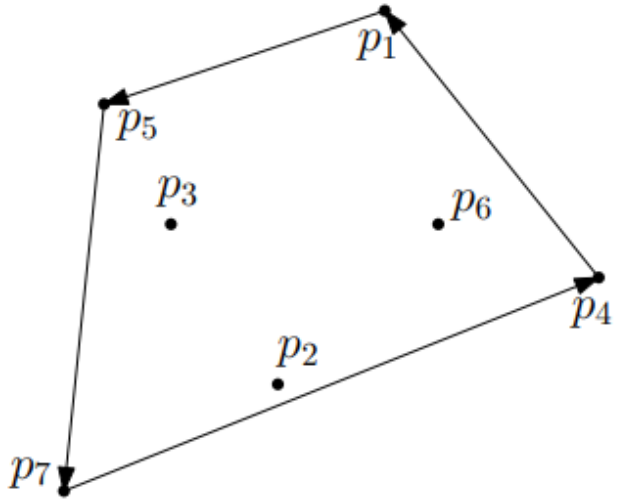
Vereinfachende Annahme: **Allgemeine Lage**, d.h. keine 3 Punkte auf einer gemeinsamen Geraden, keine 2 Pkt gleiche x-Koordinate.

Local Repair

Ein Paar $qr \in P^2$, $q \neq r$, heisst **Randkante** von P, falls alle Punkte in $P \setminus \{q, r\}$ links von q, r liegen.

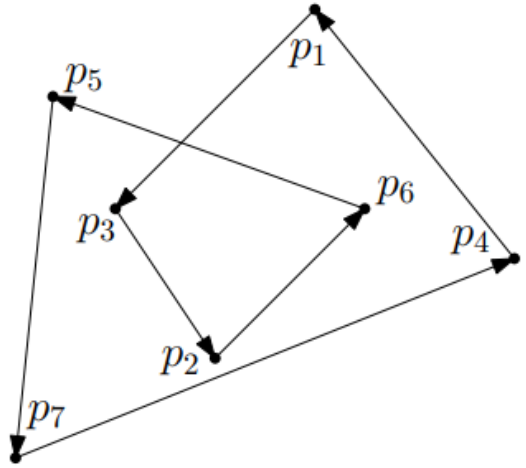
Lokale Prüfung in Polygon $(q_0, q_1, \ldots, q_{h-1})$.

$$\forall i, 1 \leq i \leq h: \; q_{i+1} \text{ links von } q_{i-1}, q_i$$



Local Repair

- $\{q_0, q_1, \ldots, q_{h-1}\}$ muss nicht Teilmenge von P sein.
- Das Polygon muss nicht alle anderen Punkte im Inneren haben.
- Das Polygon kann sich selbst kreuzen.



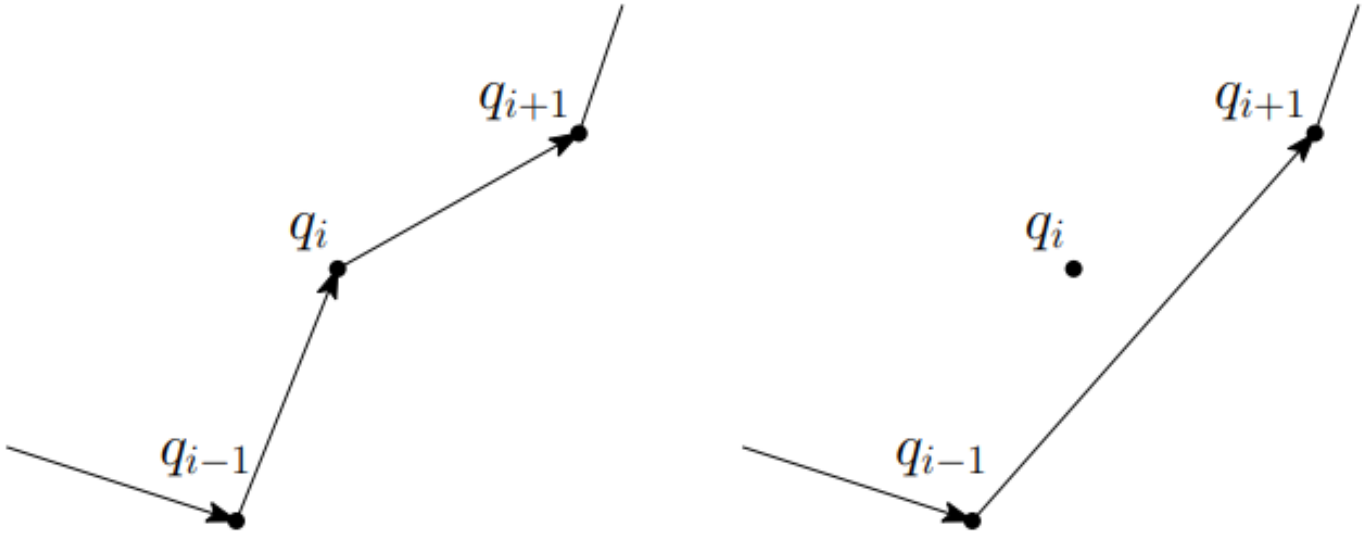
Idee: Wir beginnen mit einem nicht notwendigerweise konvexen Polygon, dass die Bedingungen oben nicht verletzt, und „**konvexifizieren**“ es sukzessive („**lokal Verbessern**“).

Local Repair

Gegeben $(q_0, q_1, \ldots, q_{k-1})$, falls

q_{i+1} rechts von q_{i-1}, q_i liegt

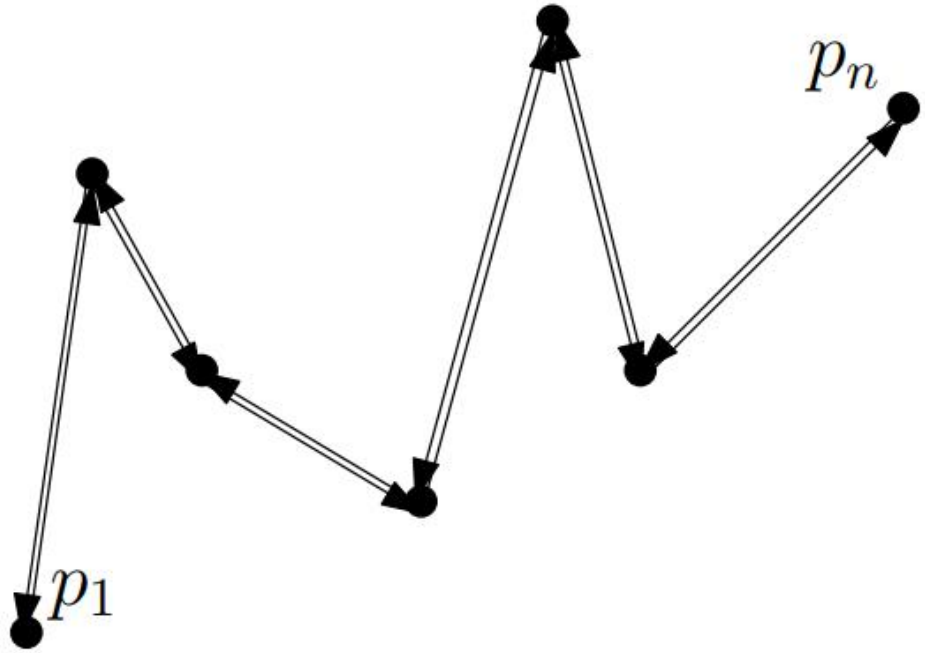
dann entferne q_i aus der Folge.



Local Repair

Sortiere P aufsteigend nach x-Koordinate: $(p_1, p_2, \ldots, p_n)$, und betrachte das Polygon

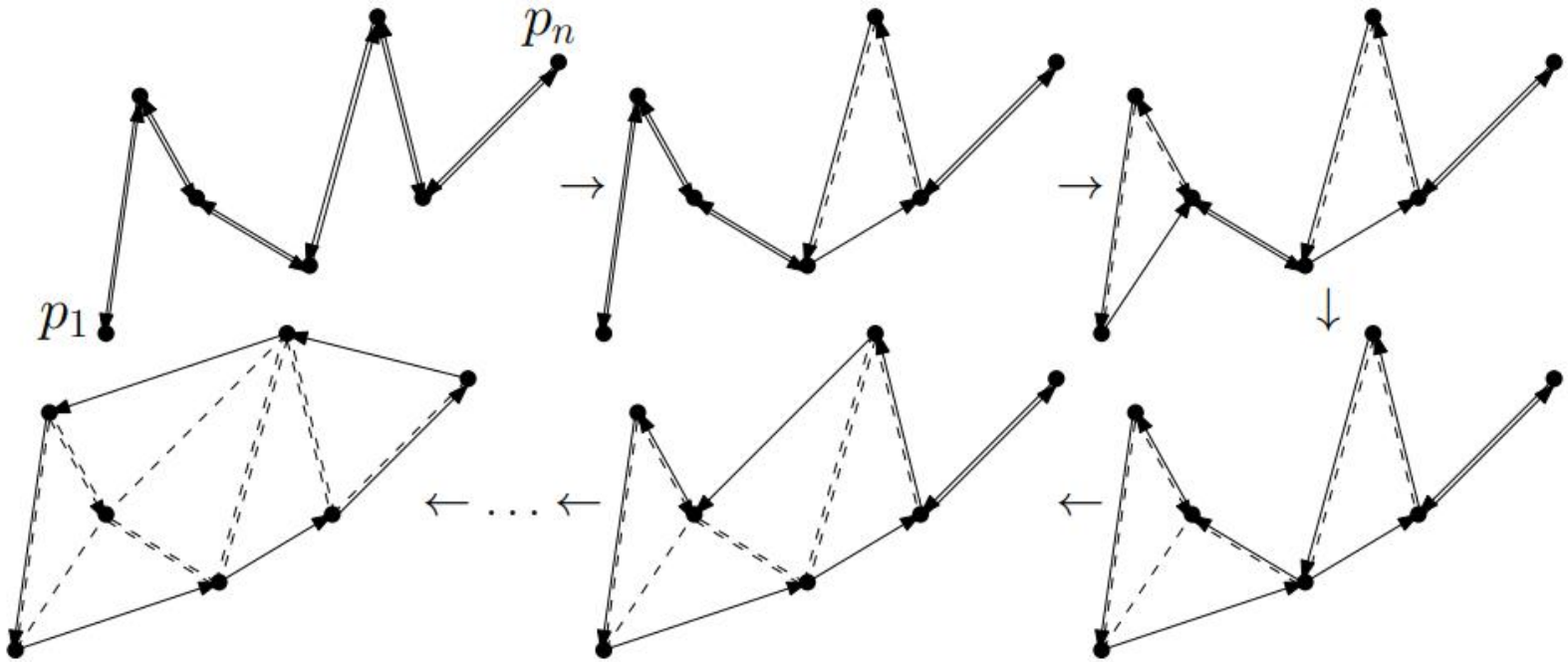
$$(p_1, p_2, \ldots, p_{n-1}, p_n, p_{n-1}, \ldots, p_2)$$



Local Repair

Sortiere P aufsteigend nach x-Koordinate: $(p_1, p_2, \ldots, p_n)$, und betrachte das Polygon

$$(p_1, p_2, \ldots, p_{n-1}, p_n, p_{n-1}, \ldots, p_2)$$



Local Repair

LocalRepair$(p_1, p_2, \ldots, p_n)$ — $(p_1, p_2, \ldots, p_n)$ sortiert

```
1:  q_0 ← p_1; h ← 0
2:  for i ← 2 to n do                ▷ unterer Rand, links nach rechts
3:      while h > 0 und p_i rechts von q_{h−1}, q_h do
4:          h ← h − 1
5:      h ← h + 1; q_h ← p_i
6:              ▷ (q_0, ..., q_h) untere konvexe Hülle von {p_1, ..., p_i}
7:  h' ← h
8:  for i ← n − 1 downto 1 do        ▷ oberer Rand, rechts nach links
9:      while h > h' und p_i rechts von q_{h−1}, q_h do
10:         h ← h − 1
11:     h ← h + 1; q_h ← p_i
12: return (q_0, q_1, ..., q_{h−1})
```

Local Repair

Satz 3.42. Gegeben eine Folge $p_1, p_2, \ldots, p_n$ nach x-Koordinate sortierter Punkte in allgemeiner Lage in $\mathbb{R}^2$, berechnet der Algorithmus LocalRepair die konvexe Hülle von $\{p_1, p_2, \ldots, p_n\}$ in Zeit $O(n)$.

Das heisst, inklusive vorbereitendes Sortieren haben wir einen $O(n \log n)$ Algorithmus, der asymptotisch schneller als JarvisWrap ist, es sei denn $h = o(\log n)$.