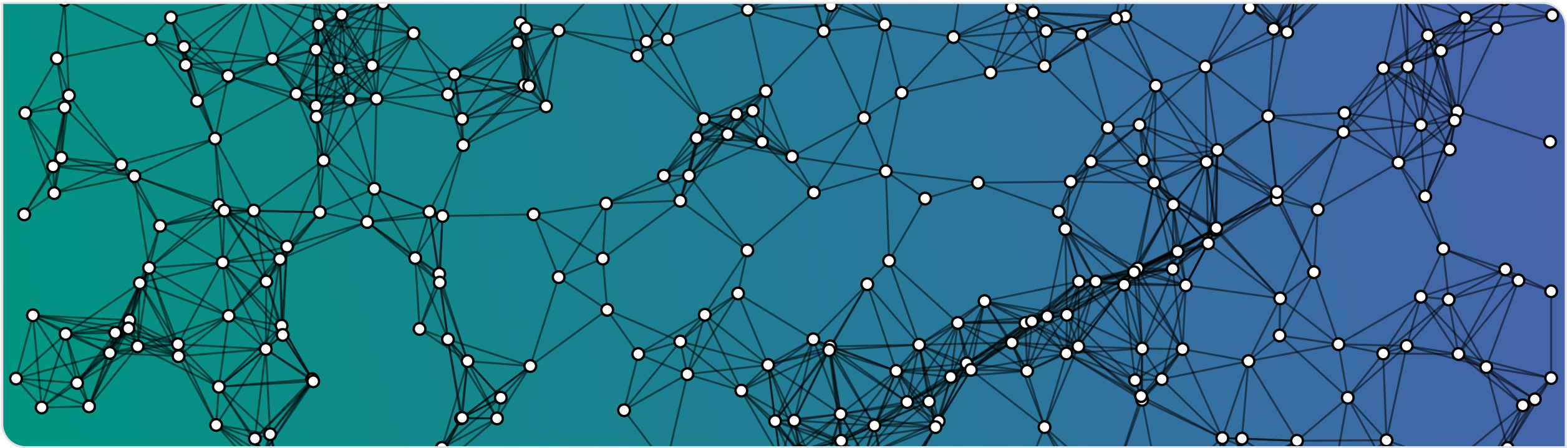
Karlsruher Institut für Technologie

Algorithmen 1

Generische Optimierungsmethoden



Beispiel: Ausgewogen und Billig

Problem

- Burger entsprechen nicht den offiziellen Ernährungsrichtlinien
- pro Gericht fehlen $0{,}5\,\mathrm{mg}$ Vitamin A, $15\,\mathrm{mg}$ Vitamin C, $4\,\mathrm{g}$ Ballaststoffe
- Ziel: Behebung dieses Problems bei möglichst geringen Kosten

	Karotten	Weißkohl	Gewürzgurken
Vitamin A (mg/kg)	35	0, 5	0, 5
Vitamin C (mg/kg)	60	300	10
Ballaststoffe (g/kg)	30	20	10
Preis (€/kg)	0, 75	0, 5	0, 15

. . . and when Rabbid said, "Honey or condensed milk with your bread?" he was so excited that he said, "Both," and then, so as not to seem greedy, he added, "But don't bother about the bread, please."
A. A. Milne, Winnie the Pooh

Lösung: x_1, x_2, x_3 repräsentieren die Menge an Karotten, Kohl und Gurken

$$
\begin{aligned}
\text{minimiere:} \quad & 0,75x_1 + 0,5x_2 + 0,15x_3 \\
\text{Nebenbedingungen:} \quad & 35x_1 + 0,5x_2 + 0,5x_3 \geq 0,5 \\
& 60x_1 + 300x_2 + 10x_3 \geq 15 \\
& 30x_1 + 20x_2 + 10x_3 \geq 4 \\
& x_i \geq 0
\end{aligned}
$$

- optimale Lösung:
  - $9{,}5\,\mathrm{g}$ Karotten
  - $38\,\mathrm{g}$ Kohl
  - $290\,\mathrm{g}$ Gurken

Lineare Programme

Trivia

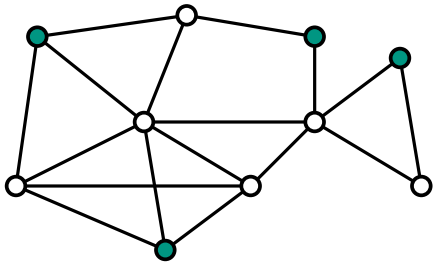
- wurden bereits in den 40er Jahren verwendet (und manuell gelöst)
- „Programm“ ist ein militärischer Begriff für verschiedene Arten von Plänen (z.B. Versorgungsplan, Verlegungsplan für Truppen etc.)
- erstes großes LP, das mit dem Simplex-Algorithmus gelöst wurde
  - optimiere Kosten für ausgewogene Ernährung
  - 77 Variablen, 9 Nebenbedingungen
  - Simplex-Methode (per Hand in 1947): 120 Personentage
- etwas später (mittels Computer): George Dantzig versucht seine eigene Ernährung zu optimieren
  - erster Versuch: mehrere Liter Essig pro Tag
  - zweiter Versuch: 200 Brühwürfel pro Tag
  - $\Rightarrow$ ein sinnvolles LP zu formulieren ist nicht immer trivial

Ein schwieriges Graphenproblem

Problem: Independent Set
Sei $G = (V, E)$ ein Graph. Finde eine möglichst große Knotenmenge $I \subseteq V$, sodass für jede Kante $\{u, v\} \in E$ höchstens einer der Knoten u oder v in I liegt.

Intuition

- die Kanten modellieren Konflikte
- finde möglichst große konfliktfreie Knotenmenge



Algorithmische Situation

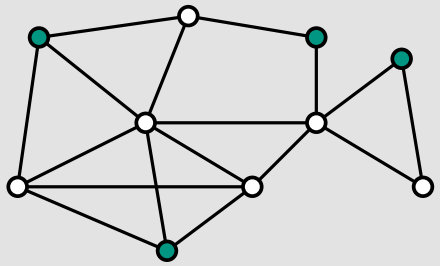
- Independent Set ist NP-schwer
  - wir kennen keinen Algorithmus mit polynomieller Laufzeit
  - wir gehen davon aus, dass es keinen solchen Algorithmus gibt
- man sollte trotzdem nicht alle Hoffnung verlieren
  - ggf. sind viele Instanzen gutartig $\rightarrow$ nur manchmal langsam
  - suboptimale Lösungen können ggf. effizient gefunden werden

Independent Set als lineares Programm

Die Variablen

- Knoten $v \in V \rightarrow$ Variable x_v
- Einschränkung: $x_v \in \{0, 1\}$
- Interpretation: $x_v = 1 \Leftrightarrow v \in I$

Problem: Independent Set
Sei $G = (V, E)$ ein Graph. Finde eine möglichst große Knotenmenge $I \subseteq V$, sodass für jede Kante $\{u, v\} \in E$ höchstens einer der Knoten u oder v in I liegt.



Optimierungsfunktion

- Ziel: wähle möglichst viele Knoten aus $\rightarrow$ setze viele Variablen auf 1
- maximiere: $\sum_{v \in V} x_v$

Nebenbedingungen

- für jede Kante: höchstens einer der Endpunkte ausgewählt
- für jede Kante $\{u, v\} \in E$ eine Nebenbedingung: $x_u + x_v \leq 1$

Anmerkung

- Forderung, dass x_v ganzzahlige Werte annimmt macht das Problem schwer

LPs und ILPs

Lineare Programme (LP)

- gegeben: lineare Optimierungsfunktion und lineare Nebenbedingungen
- finde optimale **reellwertige** Variablenbelegung unter Einhaltung der Nebenbedingungen
- Theorie: in polynomieller Zeit lösbar
- Praxis: hoch effiziente Implementierungen verfügbar (frei und kommerziell)

Ganzzahliges lineare Programme (ILP – Integer Linear Program)

- finde optimale **ganzzahlige** Variablenbelegung unter Einhaltung der Nebenbedingungen
- Theorie: NP-schwer (vermutlich nicht in polynomieller Zeit lösbar)
- Praxis: meist effiziente Implementierungen verfügbar (frei und kommerziell)

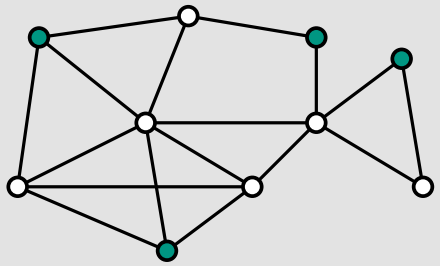
mächtiges generisches Werkzeug um auch schwierige Probleme zu lösen

Rohe Gewalt (Brute-Force)

Naiver Plan

- zähle alle Teilmengen $I \in V$ auf
- teste ob I ein Independent Set ist
- gib größtes Independent Set aus

Problem: Independent Set
Sei $G = (V, E)$ ein Graph. Finde eine möglichst große Knotenmenge $I \subseteq V$, sodass für jede Kante $\{u, v\} \in E$ höchstens einer der Knoten u oder v in I liegt.



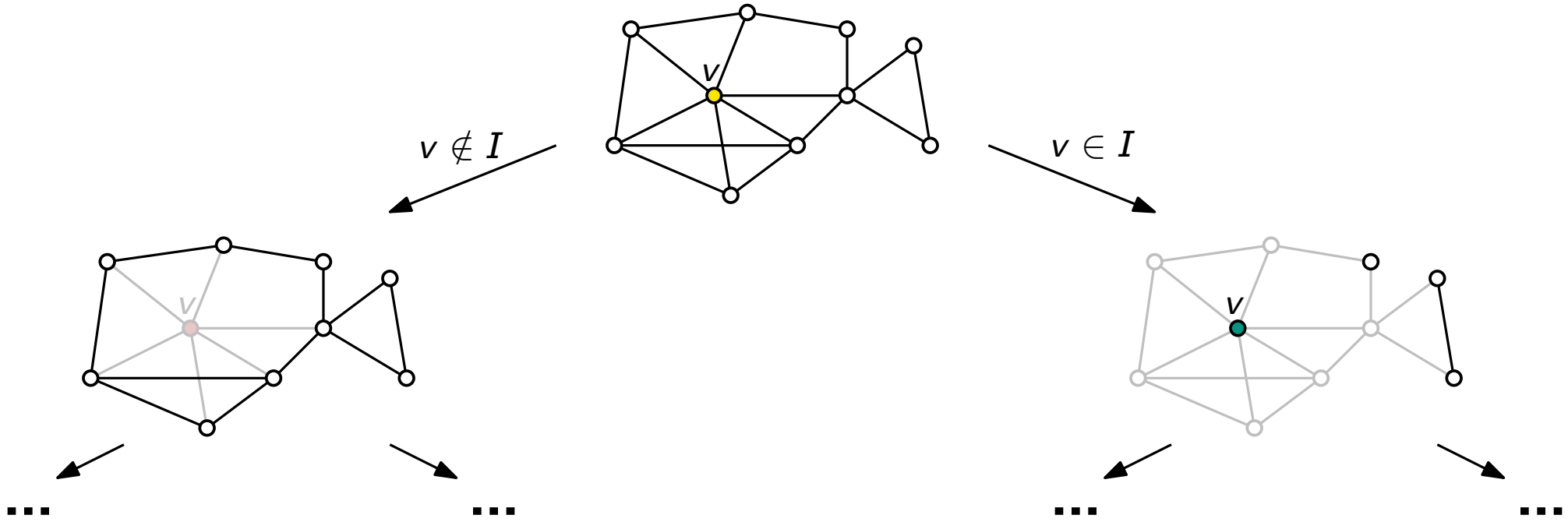
Beobachtung

- Kante $\{u, v\} \in E \rightarrow$ falls $u \in I$ und $v \in I$, dann ist I kein Independent Set
- wir könnten uns eigentlich alle Teilmengen sparen, bei denen $u \in I$ und $v \in I$

Geschickteres Brute-Force: Branching

- finde eine einzelne Entscheidung und probiere alle möglichen Fälle aus
- hier: für einen Knoten $v \in V$ entweder $v \notin I$ oder $v \in I$
- $v \notin I \rightarrow$ lösche v aus dem Graphen und löse die restliche Instanz
- $v \in I \rightarrow$ Nachbarn von v sind alle nicht in $I \rightarrow$ lösche $N(v)$ aus dem Graphen und löse die restliche Instanz

Branching: Beispiel



Beobachtung

- im Zweig $v \in I$ wurde der Graph deutlich einfacher
- je mehr Nachbarn v hat, desto mehr Knoten werden im Zweig $v \in I$ gelöscht
- sinnvolle Heuristik: wähle hochgradigen Knoten v zum branchen

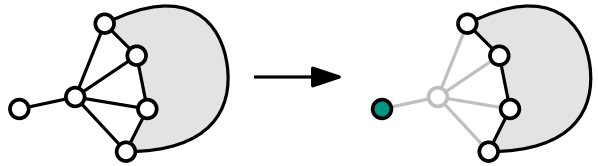
Branch-and-Bound

Wie können wir Brute Force schneller machen?

- geschicktes Branching (gerade gesehen)
- treffe „offensichtliche“ Entscheidungen $\rightarrow$ Reduktionsregel
- aktueller Zweig „offensichtlich“ schlecht $\rightarrow$ Zweig abschneiden (Pruning)

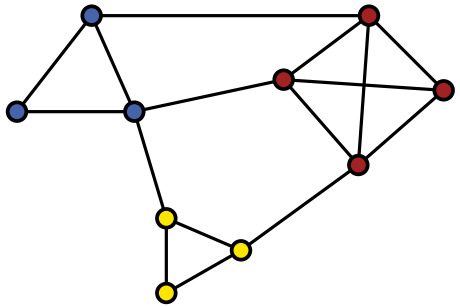
Reduktionsregel (Beispiel)

- $\deg(v) \leq 1 \Rightarrow$ es gibt ein maximales Independen Set I mit $v \in I$



Pruning (Beispiel)

- lower bound (global): wir kennen schon eine Lösung der Größe 17
- upper bound (Situation im aktuellen Zweig):
  - haben bisher 14 Knoten ausgewählt
  - übriger Graph kann mit 3 Cliquen überdeckt werden
  - Lösung in diesem Zweig kann nicht besser als 17 werde



Zwischenstand

Exakte Lösungen, ggf. langsam

- Branch-and-Bound: Framework für Brute-Force Algorithmen
  - auszufüllen: branching, Reduktionsregeln, Pruning mit upper/lower Bounds
  - Freiheitsgrade füllen ist sehr Problemspezifisch
- Formulierung als (I)LP oder auch NLP (nicht-lineares Programm)
  - Übersetzung in ein (I)LP ist problemspezifisch und nicht immer leicht
  - tatsächlicher Algorithmus: nutze stark optimierte Bibliothek

Heuristische Ansätze: opfere Optimalität für Laufzeit

- kennen wir schon: Greedy (funktioniert oft gut; meist nicht optimal)
- gleich: problemunabhängige Metaheuristiken

Wikipedia zu Metaheuristiken:
While the field also features high-quality research, many of the publications have been of poor quality; flaws include vagueness, lack of conceptual elaboration, poor experiments, and ignorance of previous literature.

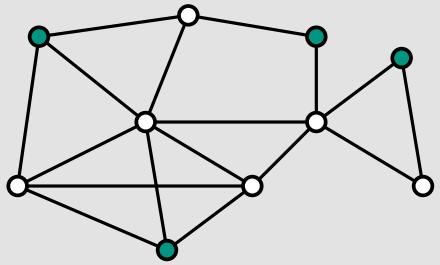
Allgemeines Optimierungsproblem

Problem: generisches Maximierungsproblem
Gegeben sei eine Funktion $f : X \to \mathbb{R}$. Finde $x \in X$ mit $f(x) \geq f(y)$ für alle $y \in X$.

Beispiel: Independent Set

- Möglichkeit 1: X komplett unabhängig vom eigentlichen Problem
  - $X = 2^V$ ist die Potenzmenge von V (z.B. codiert als Bitvektor $\{0,1\}^n$)
  - $f(x) = \begin{cases} |x| & \text{wenn } x \text{ ein Independent Set ist} \\ 0 & \text{sonst} \end{cases}$
- Möglichkeit 2: X enthält nur gültige Teilmengen $\to$ problemspezifisch
  - $X \subseteq 2^V$ ist die Menge aller Independent Sets
  - $f(x) = |x|$

Problem: **Independent Set**
Sei $G = (V, E)$ ein Graph. Finde eine möglichst große Knotenmenge $I \subseteq V$, sodass für jede Kante $\{u, v\} \in E$ höchstens einer der Knoten u oder v in I liegt.



Lokale Suche

Grundsätzliches Vorgehen

- starte mit irgendeiner Lösung $x \in X$
- iteriere solange wie gewünscht:
  - wähle Lösung $y \in X$ die ähnlich ist zu x ⟵ das lohnt sich ggf. problemspezifisch zu machen
  - falls $f(y) > f(x) \rightarrow$ ersetze x durch y

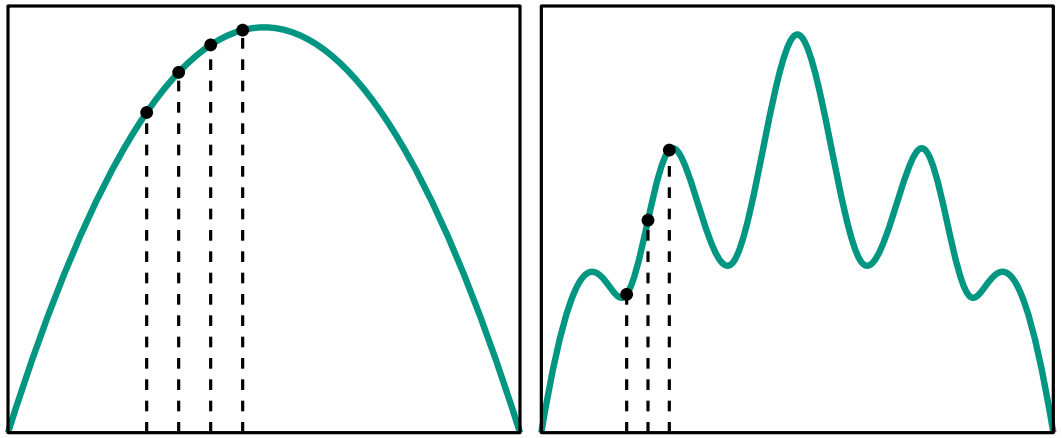
Mögliche konkrete Umsetzung (sehr generisch)

- Annahme: $X = \{0,1\}^n$
- Start: wähle $x \in X$ uniform zufällig
- wiederhole, bis x sich k mal in Folge nicht ändert:
  - setze $y = x$ und ändere ein zufälliges Bit von y
  - falls $f(y) > f(x) \rightarrow$ ersetze x durch y

Umgang mit Lokalen Optima

Problem

- lokale Suche hängt leicht in lokalen Optima fest



Lösungsansätze

- Neustart mit anderer initialen Lösung
- erlaube auch kleine Verschlechterungen mit einer gewissen Wahrscheinlichkeit
- erlaube größere Veränderungsschritte

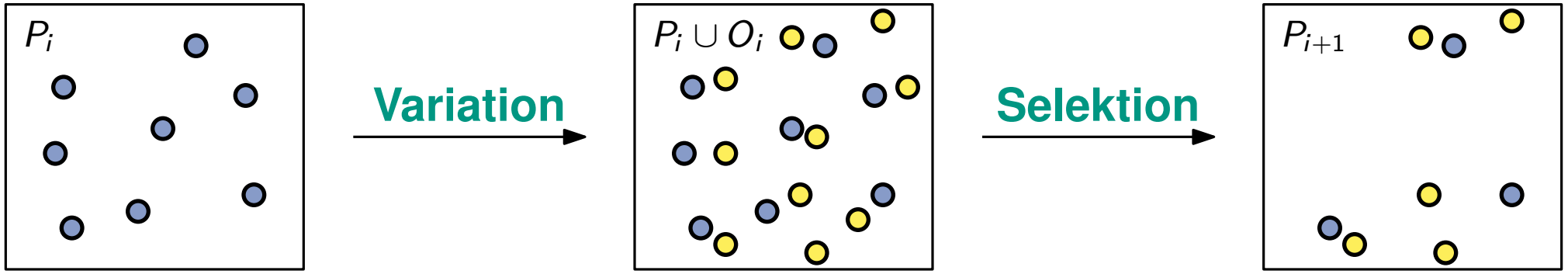
Gleich: evolutionärer Algorithmus

- ein möglicher Ansatz die lokale Suche zu erweitern
- setzt die verschiedenen Ansätze zur Vermeidung lokaler Optima um
- es gibt eine Vielzahl Varianten und verwandter Algorithmen

Evolutionärer Algorithmus

Grundsätzliches Vorgehen

- starte mit einer Menge $P_1 = \{x_1^1, \ldots, x_n^1\} \subset X$ von Lösungen
- iteriere so lange wie gewünscht:
  - erzeuge neue Lösungen $O_i = \{y_1^i, \ldots, y_m^i\}$ aus der aktuellen Lösungsmenge P_i
  - wähle neue Lösungsmenge $P_{i+1} = \{x_1^{i+1}, \ldots, x_n^{i+1}\} \subset P_i \cup O_i$ basierend f



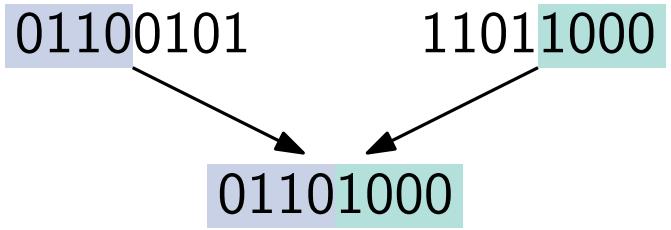
Aus der Evolution entliehene Begrifflichkeiten

- P_i ist die $i$te **Population**, f ist die **Fitnessfunktion**
- die x_j^i sind die **Eltern (parents)**, die y_j^i der **Nachwuchs (offspring)**

Variation: Rekombination und Mutation

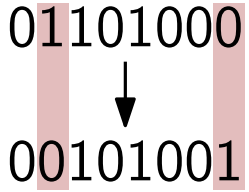
Rekombination

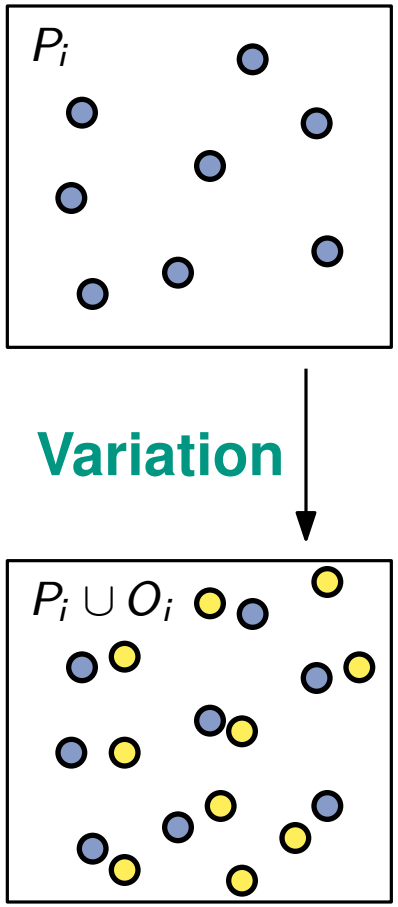
- wähle zwei (oder mehr) Lösungen $x_1^i, x_2^i \in P_i$
- kombiniere x_1^i und x_2^i auf irgendeine Art
- Beispiel:



Mutation

- ähnlich wie bei der lokalen Suche
- ändere eine (z.B. durch Rekombination erzeugte) Lösung etwas ab
- Beispiel:





Evolutionäre Algorithmen: Anmerkung

Lokale Minima und Vergleich zur lokalen Suche

- Mutation $\rightarrow$ wie bei der lokale Suche
- größere Population $\rightarrow$ startet lokale Suche im Prinzip an mehreren Stellen
- Rekombination $\rightarrow$ größere Sprünge möglich (nicht nur lokale Veränderungen)
- trotzdem: wir können natürlich nicht garantieren, dass wir das Optimum finden

Generisch vs. problemspezifisch

- Erinnerung: zwei mögliche Definitionen für die Lösungsmenge X

• $X = 2^V$ ist die Potenzmenge von V
  • jeder Bitvektor kommt in Betracht
  • generische Rekombination und Mutation möglich
	Generisch
	Pro: sehr leicht umzusetzen
	Con: funktioniert nicht immer gut

• $X \subseteq 2^V$ ist die Menge aller Independent Sets
  • nur manche Bitvektoren sind gültig
  • Rekombination und Mutation problemspezifische
	Problemspezifisch
	Con: mehr Arbeit
	Pro: funktioniert meist deutlich besser

Zusammenfassung

Lineare Programmierung

- mächtige Modellierungssprache um verschiedene andere Probleme auszudrücken
- noch mächtiger: ganzzahlige Variablen
- mächtige Bibliotheken zum Lösen $\rightarrow$ man muss selbst keinen Algorithmus bauen

Branch-and-Bound (Brute-Force)

- allgemeines Framework um geschickt alle Lösungen durchzuprobieren
- problemspezifisch: branching, Reduktionsregeln, Pruning mit upper/lower Bounds
- langsam im worst-case, aber in der Praxis oft gut

Metaheuristiken: Lokale Suche, Evolutionäre Algorithmen

- sehr generisch $\rightarrow$ für alle Probleme nutzbar, geringer Arbeitsaufwand
- keine (kaum) Garantien für Qualität und Laufzeit
- problemspezifische Anpassungen $\rightarrow$ bessere Qualität und Laufzeit