Konstruktive Heuristiken

Konstruktive Heuristik: Ein Plan oder ein Schedule wird durch das *schrittweise Ein- oder Anfügen* einer Operation oder einer Menge von Operationen erzeugt:

- **Reihungsregeln (Dispatching Rules);**
- **Beam-Search Verfahren;**
- **Spezielle Verfahren unter Ausnutzung struktureller Eigenschaften des Problems.**

Einfache Reihungsregeln lassen sich wie folgt klassifizieren:

- **Statische Regeln** sind eine Funktion der Eingabedaten.
- **Dynamische Regeln** sind zeitabhängige Regeln, d.h. sie benutzen zusätzlich Daten des bisher erzeugten Schedules.

* **Lokale Regeln** nutzen Informationen über die zur Verfügung stehenden Maschinen bzw. Aufträge.
* **Globale Regeln** benutzen zusätzliche Informationen.

Reihungsregeln für Operationen

Kurz	Regel	Bedeutung
RANDOM (SIRO)	Service in random order	Die Operationen werden in zufälliger Reihenfolge eingeplant.
SPT	Shortest processing time first	Die Operation mit kleinster Bearbeitungszeit wird zuerst eingeplant.
LPT	Longest processing time first	Die Operation mit längster Bearbeitungs-zeit wird zuerst eingeplant.
ERD FIFO	Earliest release date first ($\hat{=}$ First in first out)	Die als Erstes zur Verfügung stehende Operation wird zuerst eingeplant.

Reihungsregeln für Aufträge

Kurz	Regel	Bedeutung
EDD	Earliest due date first	Der Auftrag mit dem kleinsten Due Date wird zuerst eingeplant.
MS	Minimum slack first	Der Auftrag, für den $\max\{d_i - p_i - t, 0\}$ angenommen wird, wird zuerst eingeplant.
WSPT	Weighted shortest processing time first	Der Auftrag mit dem größten Wert von w_i/p_i wird zuerst eingeplant.
CP	Critical path rule	Der Auftrag mit der größten Summe an Bearbeitungszeiten auf einem Weg zu einer Senke wird zuerst eingeplant (d.h. bei $p_i = 1$ der Auftrag mit größtem Vorwärtsrang)
LNS	Largest number of successors first	Der Auftrag mit den meisten Nachfolgern wird zuerst eingeplant.

Reihungsregeln für Aufträge

Kurz	Regel	Bedeutung
SST	Shortest set-up time first	Der Auftrag mit der kürzesten Rüstzeit wird zuerst eingeplant.
LAPT (m=2)	Longest alternate processing time first	Wenn eine Maschine frei wird, plane den Auftrag darauf an, der die längste Bearbeitungszeit auf der anderen Maschine hat.
SQ	Shortest queue first	Jeder eintreffende Auftrag wird der Maschine mit der kürzesten Warteschlange zugewiesen.
SQNO	Shortest queue at the next operation	Wenn eine Maschine frei wird, wird der Auftrag mit der kürzesten Warteschlange auf der nächsten Maschine in seiner technologischen Reihenfolge eingeplant.

Beam-Search-Verfahren

Beam-Search-Verfahren: Aus jedem *Branch & Bound Verfahren*, für das im Branching-Schritt ein An- oder Einfügen von Operationen verwendet wird, lässt sich sofort eine Konstruktionsverfahren entwickeln:
- Man verzweigt nur in die *Tiefe*;
- Man lässt auf jedem Level nur eine *maximale Anzahl* k von Knoten stehen,
  k ist die Beamweite (beam-width).
- Diese k Knoten können nach verschiedenen Strategien ausgewählt werden,
  z.B. alle Knoten sind Kinder verschiedener Eltern oder nicht.

Bemerkung: Man unterscheidet *Beam-Search-Anfügeverfahren* und *Beam-Search-Einfügeverfahren*

Hinweis: Beam-Search Verfahren haben im Allgemeinen eine *höhere Zeitkomplexität* als einfache Reihungsregeln.

Beam-Search-Anfügeverfahren für ein Flow-Shop Problem

Beispiel 1. Gegeben: $F \mid prmu \mid C_{max}$

Näherungslösung: organisatorische Reihenfolge der Aufträge $A_{i_1}, A_{i_2}, \ldots, A_{i_n}$

Anfügetechnik: Es bezeichne $C(i_1, i_2, \ldots, i_k)$ die *Gesamtbearbeitungszeit* der ersten k Aufträge. Fixiere im Schritt k den Auftrag A_{i_k} an Position k.

- Der Auftrag an erster Position ergibt sich aus

$$\sum_{j=1}^{m} p_{i_1 j} = \min_{i \in I} \left\{ \sum_{j=1}^{m} p_{ij} \right\} \quad \Longleftrightarrow \quad C(i_1) \leq C(i) \text{ für alle } i \in I.$$

- Wenn die organisatorische Reihenfolge mit $A_{i_1} \rightarrow A_{i_2} \ldots \rightarrow A_{i_{k-1}}$, $k - 1 < n$, schon festliegt, dann füge den Auftrag A_{i_k} an, für den gilt:

$$C(i_1, \ldots, i_{k-1}, i_k) = \min_{l \in I \setminus \{i_1, \ldots, i_{k-1}\}} \{C(i_1, \ldots, i_{k-1}, l)\}$$

Beam-Search-Einfügeverfahren für ein Flow-Shop Problem

Beispiel 2. Gegeben: $F \mid prmu \mid C_{max}$

Näherungslösung: organisatorische Reihenfolge der Aufträge $A_{i_1}, A_{i_2}, \ldots, A_{i_n}$

Einfügetechnik: Man gebe eine Reihenfolge der Aufträge vor, man ordne sie z.B. nach *nicht-wachsenden Summen* ihrer Bearbeitungszeiten auf den Maschinen an: $A_{i_1^*}, \ldots, A_{i_n^*}$. Fixiere in Schritt k die Position des k-ten Auftrags $A_{i_k^*}$ in der organisatorischen Reihenfolge der ersten k Aufträge:

- Falls die Teilreihenfolge $A_{i_1}, \ldots, A_{i_{k-1}}$, $k-1 < n$, ermittelt wurde, lässt sich der neue Auftrag $A_{i_k^*}$ auf allen Positionen r mit $1 \leq r \leq k$ einfügen. Schließlich wählt man die Position aus, für die die Gesamtbearbeitungszeit der k Aufträge am kleinsten ist, d.h. man bestimmt

$$\min \left\{ C(i_k^*, i_1, \ldots, i_{k-1}), C(i_1, i_k^*, i_2, \ldots, i_n), \ldots, C(i_1, \ldots, i_n, i_k^*) \right\}.$$

Spezielle Konstruktionsverfahren - Beispiel

Beispiel 3. $O \mid\mid C_{max}$ - Problem (o.B.d.A. sei $n = m$ und $SIJ = I \times J$):

Idee: Schränke die Menge aller Pläne auf die Menge aller rangminimalen Pläne ein.

S1: Setze $rk = 1$;
S2: Falls noch nicht alle Operationen einen Rang zugewiesen bekommen haben, dann:
{ Löse auf der Menge dieser Operationen ein Zuordnungsproblem mit der Zielfunktion f und weise allen Operationen der optimalen Zuordnung den Rang rk zu, setze $rk = rk + 1$ und gehe zu S2}, sonst: STOP.

Ein *Zuordnungsproblem* ist die Bestimmung eines *perfekten Matchings* M auf dem zugehörenden bipartiten Graphen $G^b = (I \times J, E)$ mit $(i,j) \in E \leftrightarrow (i,j) \in SIJ$ mit Kantengewichten p_{ij} unter

- Minimierung von $f = \Sigma_{(ij)\in M}\ p_{ij}$ oder
- Maximierung von $f = \Sigma_{(ij)\in M}\ p_{ij}$ oder
- Minimierung von $f = \max\{p_{ij} \mid (i,j) \in M\}$ oder
- Maximierung von $f = \min\{p_{ij} \mid (i,j) \in M\}$.

Iterative Heuristiken - Metaheuristiken

Idee: Bestimme eine *Startlösung* und versuche, diese Lösung schrittweise zu verbessern.

Problem: Bestimme x^* mit $f(x^*) = \min\{ f(x) \mid x \in S, S \neq \emptyset, \text{ und } S \text{ endlich } \}$!

Definition 1: Ein zu diesem Problem gehörender *Nachbarschaftsgraph* $G^N = (V, E)$ ist ein ungerichteter oder gerichteter Graph. Die Knoten entsprechen den Lösungen und zu jeder Lösung x ist eine Nachbarschaft $N(x) \subseteq S \setminus \{x\}$ gegeben mit:
y benachbart zu x genau dann, wenn $y \in N(x)$.
(Eine Nachbarschaft ist eine *Abbildung* von S in die Potenzmenge von S.)

Wichtig: *Zusammenhang* und *Durchmesser* (maximaler Abstand zwischen zwei Knoten) von G^N

Bemerkung: Die Nachbarschaft wird *problembezogen* definiert.
Symmetrische Nachbarschaften werden durch einen *ungerichteten* Graphen beschrieben.
Zu jedem $x \in S$ sollte die Anzahl der Nachbarn *polynomial beschränkt* sein.

Ausgehend von der Startlösung sucht man auf G^N gezielt nach besseren Lösungen.

Nachbarschaften für Metaheuristiken

Eine Lösung sei gegeben durch eine *lineare Ordnung (Permutation)* der Operationen:

Name	Bedeutung	Beschreibung
API	adjacent pairwise interchange critical-API	Zwei benachbarte Operationen werden vertauscht. Zusätzlich wird dabei der kritische Weg 'zerstört'.
PI	pairwise interchange critical-PI	Zwei beliebige Operationen werden vertauscht. Zusätzlich wird dabei der kritische Weg 'zerstört'.
Shift	Shift-Nachbarschaft critical-Shift	Eine Operation wird nach rechts oder links verschoben. Zsätzlich wird dabei der kritische Weg 'zerstört'.
Block	Blockaustausch critical-Block	Zwei benachbarte Blöcke von Operationen werden getauscht. Zusätzlich wird dabei der kritische Weg 'zerstört'.

Merke: - Nachbarschaften sind nicht redundanzfrei;
- die critical-Nachbarschaften reduzieren jeweils die Menge der Nachbarn;
- bei Anwendung auf partielle Ordnungen muss Zulässigkeit gewährleistet werden.

Einfache lokale Suche: Iterative Improvement

Algorithmus Iterative Improvement

Eingabe: Zielfunktion, Nachbarschaft;
Ausgabe: lokales Optimum in $\tilde{x}$.
BEGIN
  Bestimme eine Startlösung $\tilde{x}$;
  REPEAT
    Bestimme die *beste* Lösung x' aus $N(\tilde{x})$, d. h. $\forall\, x \in N(\tilde{x}) : f(x') \leq f(x)$;
    IF $f(x') < f(\tilde{x})$ **THEN** $\tilde{x} := x'$;
  UNTIL $f(x') \geq f(\tilde{x})$;
END.

Bemerkung: Der Algorithmus arbeitet nach dem Prinzip der *besten Verbesserung*. Oft wird diese Variante abgewandelt: Sowie man eine bessere Lösung in der Nachbarschaft von $\tilde{x}$ gefunden hat, geht man zu dieser über (Prinzip der *ersten Verbesserung*).

Threshold Algorithmen

Idee: Man akzeptiert auch kleine Verschlechterungen der Zielfunktion, damit man nicht vorzeitig in einem *lokalen Optimum* auf G^N stagniert.

Algorithmus Threshold

Eingabe: Zielfunktion, Nachbarschaft, Folge $\{t_k\}$ von Thresholds (Schwellwerten);
Ausgabe: beste gefundene Lösung $\tilde{x}$.
BEGIN Bestimme eine Startlösung $\tilde{x}$; $k := 0$;
  REPEAT
    Bestimme eine Lösung x' aus $N(\tilde{x})$;
    IF $f(x') - f(\tilde{x}) < t_k$ **THEN** $\tilde{x} := x'$;
    $k := k + 1$;
  UNTIL STOP;
END.

Bemerkung: Je nach Wahl der Folge $\{t_k\}$ entstehen die Verfahren *Iterative Improvement* (für alle k ist $t_k = 0$), *Threshold Accepting* und *Simulated Annealing*.

Simulated Annealing (1)

Idee: Man akzeptiert eine schlechtere Lösung mit einer Wahrscheinlichkeit, die im Laufe des Verfahrens gegen Null konvergiert. Der Übergang $x \to y \in N(x)$ im Schritt k wird mit folgender Wahrscheinlichkeit akzeptiert:

$$P(y \text{ wird akzeptiert}) = \begin{cases} 1, & \text{wenn } f(y) \leq f(x); \\ e^{-\frac{f(y)-f(x)}{c_k}}, & \text{sonst.} \end{cases}$$

Damit wird die Lösung y akzeptiert, wenn $e^{-\frac{f(y)-f(x)}{c_k}} \geq \rho$ gilt, wobei ρ eine *gleichverteilte Zufallszahl* im Intervall $[0, 1]$ ist.

$c_k > 0$ ist ein *Kontrollparameter* ('Temperatur'), der im Laufe des Algorithmus gegen Null konvergiert.

Oft wird ein *geometrisches Abkühlungsschema* genutzt:
$c_{k+1} = \lambda c_k$, mit $0 < \lambda < 1$ und λ nahe 1, $k = 0, 1, \ldots$
mit nicht zu großem Startwert $c_0 > 0$.

Simulated Annealing (2)

Algorithmus Simulated Annealing

Eingabe: Zielfunktion, Nachbarschaft, λ, c_0;
Ausgabe: beste gefundene Lösung x^*.
BEGIN Bestimme eine Startlösung x; $k := 0$; $x^* := x$; $best := f(x)$
  REPEAT
    Bestimme eine zufällige Lösung $y \in N(x)$;
    IF $f(y) < best$ **THEN** $(x^* := y \wedge best := f(y))$;
    Bestimme eine Zufallszahl $\rho \in [0, 1]$;
    IF $\rho \leq \min\left\{1, \exp(-\frac{f(y)-f(x)}{c_k})\right\}$ **THEN** $x := y$;
    $c_{k+1} := \lambda c_k$; $k := k + 1$;
  UNTIL STOP;
END.

Tabu Search

Idee: Man möchte bei der Suche nicht zu schon einmal betrachteten Lösungen zurückkommen, daher: Einführung einer Tabu-Liste TL von Lösungen bzw. von Eigenschaften von Lösungen, die nicht wieder besucht werden sollen.

Algorithmus Tabu Search

Eingabe: Zielfunktion, Nachbarschaft, Tabu-Eigenschaft, Länge der Tabu-Liste TL;
Ausgabe: beste gefundene Lösung x^*.
BEGIN Bestimme eine Startlösung x; $TL = \emptyset$; $x^* := x$; $best := f(x)$
  REPEAT
    Bestimme
    $Cand(x) = \{y \in N(x) \mid y$ ist nicht tabu $\vee\ y$ erfüllt ein Aspiration-Kriterium$\}$;
    Wähle ein $y \in Cand(x)$; Aktualisiere die Tabu-Liste TL;
    IF $f(y) < best$ **THEN** $(x^* := y \wedge best := f(y))$;
    $x := y$;
  UNTIL STOP;
END.

Genetische Algorithmen

Idee: Nach *Darwin's Evolutionstheorie* ('survival of the fittest') überleben die der Umwelt am besten angepassten Individuen.

Eine *Population* ist eine Menge von *Individuen* (Lösungen), wobei jedes Individuum mit einer *Fitness* (z.B. Zielfunktionswert bei Maximierungsproblem bzw. reziproker Zielfunktionswert bei Minimierungsproblem) ausgestattet ist. Eine Ausgangsmenge von Individuen bildet eine *Generation*. Beim Übergang zur nächsten Generation sterben Individuen, andere vererben ihre Eigenschaften nach den Gesetzen der Natur (z.B. *Crossover* und *Mutation* der Gene).

Bemerkung: Ein genetischer Algorithmus ist eine Metaheuristik mit einer Menge von Startlösungen, die iterativ von Generation zu Generation verändert wird, um bessere Lösungen zu erhalten.

Genetische Algorithmen werden oft allgemein beschrieben, wobei Crossover und Mutation der Gene problembezogen definiert werden und ihre Anwendung zur Erzeugung von Nachkommen über Wahrscheinlichkeitsparameter gesteuert wird. Oft wird nach Erzeugung der Nachkommen versucht, diese Lösungen iterativ weiter zu verbessern.

Genetische Algorithmen

Algorithmus Basisschema eines genetischen Algorithmus

Eingabe: Zielfunktion, Parameter des Algorithmus wie z.B. Populationsgröße k, Anzahl der zu erzeugenden Generationen; Wahrscheinlichkeiten für die Anwendung eines Crossovers bzw. einer Mutation;
Ausgabe: beste gefundene Lösung x^*.

BEGIN Bestimme eine Menge von k Lösungen durch verschiedene konstruktive Näherungsverfahren;
Bestimme z.B. durch lokale Suche zu jeder Lösung eine mindestens gleich gute;
Die erhaltene Menge bildet die Startpopulation von Individuen;
REPEAT
Bilde aus der aktuellen Population durch Anwendung genetischer Operatoren (z.B. mittels Crossover zwischen zwei Individuen, Mutation in einem Individuum) die Nachkommen;
Bestimme ggf. aus den neuen Individuen wiederum durch lokale Suche bessere Lösungen;
Bestimme eine neue Population gemäß der Fitness der Individuen);
Speichere dabei die beste gefundene Lösung x^*;
UNTIL STOP;
END.