Aufgabenblatt 1 – Musterlösung

Aufgabe 1: Aufgabenblatt effizient bearbeiten

a) Formulieren Sie das Problem als Suchproblem.

R = {(A,1,2), (A,2,8), (A,3,6), (A,4,2), (A,5,4), (B,1,1), (B,2,4), (B,4,1), (B,5,2), (C,2,2), (C,5,1)} beschreibt alle möglichen Aufgabenzuteilungen der Art (Alfons, Aufgabe 1, 2 Stunden). Diese Menge ist unveränderlich (atemporal) und kann im Zustand, aber ebenso gut außerhalb als globale Konstante repräsentiert werden.

A = [1,2,3,4,5] sei die Liste der noch offenen Aufgaben;

Z = { (A, {}, 0), (B, {}, 0), (C, {}, 0) } enthält die aktuellen Zuteilungen für A(lfons), B(ernd) und Christiane, zu Beginn jeweils eine leere Menge mit einer persönlichen Bearbeitungszeit von 0.

B = 0 ist die aktuelle Bearbeitungszeit (das Maximum der 3 Einzelbearbeitungszeiten.

S0 = (A, Z, B, R)

SZ = ([], Z, B*, R)*

Wenn alle Aufgaben zugewiesen wurden ([] bezeichnet eine leere Liste) ist ein Zielzustand erreicht. Z enthält die Aufgabenzuordnungen an die 3 Studierenden und B* die Gesamtbearbeitungszeit.*

Aktion: assign(([a|T], Z, B, R)) = (T, Zneu, Bneu, R) und

[a|T] bezeichnet eine Liste, deren erstes Element a und der Rest T ist. (T kann auch die leere Liste sein). a repräsentiert also die erste der noch offenen Aufgaben. Und:

(p,a,t) ∈ R (eine Person p bearbeitet Aufgabe a in t Stunden gemäß Aufgabenbeschreibung). Und:

(p,ap,tp)) ∈ Z (Person p hat aktuell die Aufgaben ap zugewiesen bekommen mit einer persönlichen Gesamtbearbeitungszeit tp) und

Zneu = Z \ {(p,ap,tp)} ∪ (p, ap∪a, tp+t) (die Aufgabe wird der Person zugeordnet und die persönliche Bearbeitungszeit um die Dauer der Aufgabe inkrementiert. D.h., das alte Element wird aus der Menge entfernt (mittels „\“) und das (um Aufgabe a mit Zeit t) aktualisierte der Menge hinzugefügt. Und:

Bneu = max(B, tp) (die neue Gesamtbearbeitungszeit erhöht sich genau dann, wenn die bisherige Gesamtbearbeitungszeit durch die aktuelle Aufgabenzuweisung größer wird.

Aktionskosten: Bneu – B (nur wenn die Gesamtbearbeitungszeit um ein Delta zunimmt, dann fallen Aktionskosten in Höhe dieses Deltas an. Ansonsten kostet eine Aktion 0

b)

Der Zieltest wird immer nur auf dem aktuellen Zustand ausgeführt. Die Teilbedingung „Kosten minimal“ kann nicht geprüft werden. Denn um Minimalität zu testen, müssen alle Ergebnisse vorliegen.

Man könnte die Suchproblembeschreibung dahingehend verändern, indem man in den Zieltest noch aufnimmt: „alle Zustände besucht“ – aber auch dieser Test ist nicht überprüfbar!

Die einzige „Lösung“ wäre, alle Lösungen zu generieren und die jeweils beste zu speichern. Es müsste also der gesamte Problemraum durchsucht werden. Dies widerspricht aber der Suchbaumalgorithmus-Idee, bei der ersten gefundenen Lösung zu terminieren.

c) Suchbaumcharakteristik:

Der Verzweigungsgrad ist 3, weil maximal 3 Studierende zur Bearbeitung der aktuellen Aufgabe infrage kommen. Die Tiefe des Suchbaums ist 5, weil in jeder Aktion eine Aufgabe zugewiesen wird.

d) Eigenschaften uninformierter Suchstrategien hinsichtlich der Problemformulierung.

Tiefensuche: Findet nach 5 Aktionen eine Lösung. Diese ist vermutlich nicht optimal.

*Breitensuche: Braucht sehr lange, da der gesamte Suchbaum vollständig generiert wird, bevor die erste Lösung vorne in der Queue liegt. Dieser erste Lösungspfad ist vermutlich ebenfalls nicht optimal. Konkret werden 1 + 2 + 2*3 + 2*3*1 + 2*3*1*2 Knoten bis zu Baumebene 4 generiert. Und auf der 5. (Blatt-) Ebene noch einmal 2*3*1*2*3 Knoten.*

Branch&Bound: Untersucht immer den kostengünstigsten Pfad. Ist relativ langsam (für dieses Problem aber in den meisten Fällen echt schneller als Breitensuche), und garantiert die optimale Lösung.

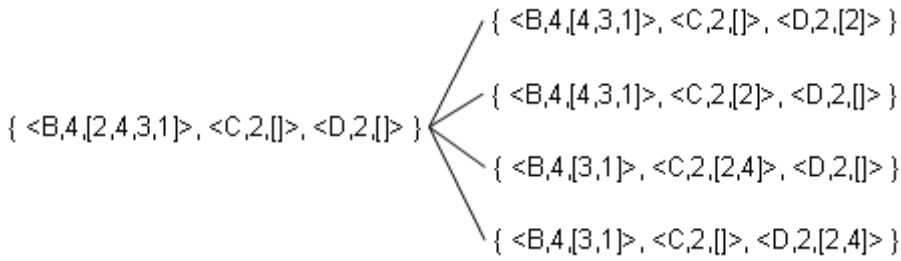
Aufgabe 2 – Rangierbahnhof

Hier kann man die Lok sowie den Gleisabschnitt A „wegabstrahieren“. Denn die Lok kann jeweils nur 1 – 2 Waggons von einem Gleisabschnitt über A zu einem anderen Gleisabschnitt transportieren, vorausgesetzt es sind genügend Waggons auf dem Quellgleis und genügend Platz auf dem Zielgleis. Auch kann man das Problem dahingehend vereinfachen, indem man beim Verschieben einer Menge von Waggons diese immer so weit wie möglich zum Ende des Zielgleises hin schiebt – es ergibt keinen Sinn, z.B, den Waggon 2 auf das erste freie Feld in C zu schieben. Gleise sind Stacks mit maximaler Kapazität. Der Stack enthält 0, …, Kapazität Waggons. Die Stack-Variante ist deshalb sinnvoll, weil man ja an die Waggons immer nur von der einen Seite herankommt. Zustand: 3-elementige Menge von 3-Tupeln, wobei jedes Tupel aus Gleisname, Kapazität und Stack besteht. Es gilt weiterhin die Zulässigkeitsbedingung dass alle Waggons in der Menge (jeweils genau einmal) enthalten sind, unter Beachtung der jeweiligen Kapazitäten. Damit ist sichergestellt, dass jeder Zustand zulässig ist. Der Zustandsraum definiert sich dann als Menge aller möglichen Zustände.

Anfangszustand: { <B,4,[2,4,3,1]>, <C,2,[]>, <D,2,[]> }
Zielzustand: { <B,4,[1,2,3,4]>, <D,2,[]>, <C,2,[]> } // Reihenfolge der Elemente egal
Aktion: rangiere(*N*, *Q*, *Z*) (rangieren von 1 oder 2 Waggons von einem Gleis *Q* zu einem Gleis *Z* unter Beachtung der Kapazitäten). Sei Z´ der Nachfolgezustand, NoChange das nicht von der Aktion betroffene Tupel, k (X) die Kapazität des Gleisstücks X (wie sie im Tupel an 2. Stelle steht), stack(X) der Stack (in Listenform) des Gleisstücks X (aktueller Füllstand des Gleisstücks). top(N,S) die N obersten Elemente des Stacks S, pop(N,S) ergibt den Stack, nachdem N pop-Operationen auf ihm ausgeführt wurden. Dieses gelingt natürlich nur dann, wenn auch mindestens N Elemente auf dem Stack liegen. Analog push(S,S1) legt die Elemente des Stacks S auf den Stack S1 (in der korrekten Reihenfolge) – push gelingt nur dann, wenn die Kapazität des Stacks nicht überschritten wird.

- $N \in \{ 1, 2 \}$ (die Zahl der zu bewegenden Waggons)
- $Q, Z \in \{ B, C, D \}$ und zusätzlich $Q \neq Z$
- $Z' = \{<Z, k(Z), push(top(N,stack(Q))>, <Q, k(Q), pop(N,stack(Q)>, NoChange\}$

4 Folgezustände existieren für den Startzustand:



Der Verzweigungsgrad ist theoretisch maximal N * |Q| * |Z| = 2*3*2 = 12. Jedoch ist er tatsächlich kleiner aufgrund der geringen Gleiskapazitäten. So können im Zustand {<B,4,[1,2]>,<C,2,[3,4]>,<D,2,[]>} maximal 6 Aktionen ausgeführt werden.
Dasselbe gilt, falls je ein Waggon auf C und auf D steht. Also ist der tatsächliche Verzweigungsgrad für dieses Problem 6. Die maximale Tiefe des Baums kann mit der Zahl der theoretisch möglichen Zustände sehr konservativ abgeschätzt werden: Es existieren 4 Wagons und 3 Gleise mit insgesamt 8 Feldern. Der zyklenfreie Suchbaum hätte dann 8 über 4 = 8!/4! = 1680 Zustände. Eine sehr viel gröbere und somit schlechtere Schätzung lautet 4^8 = 65536 Zustände.

Am besten Breitensuche verwenden, denn eine optimale Lösung ist gesucht. Der Aufwand für die Suche wird sicherlich im Minutenbereich liegen, während die „Kosten" des Rangierens erheblich höher liegen. Somit kostet eine mit Tiefensuche schnell berechnete, aber suboptimale Lösung in der Ausführung deutlich mehr Zeit.
Bemerkung:

- Wenn die Lokomotive mit modelliert wird, dann müssen auch Leerfahrten der Lok gültige Aktionen sein.

Aufgabe 3: A*-Simulation

							8
11							7
9	7	5	5				6
9		3	3		Z		5
11			5		9	11	4
	11	9	7		9	11	3
		11	9	9	9	11	2
			11	11	11		1
a	b	c	d	e	f	g	

							8
							7
9	7	5	5				6
		3	3		Z		5
			5		9	11	4
	11	9	7		9	11	3
		11	9	9	9	11	2
			11	11	11		1
a	b	c	d	e	f	g	

a) In den Feldern stehen die f-Werte, grau hinterlegt sind die untersuchten Zustände, weiß die

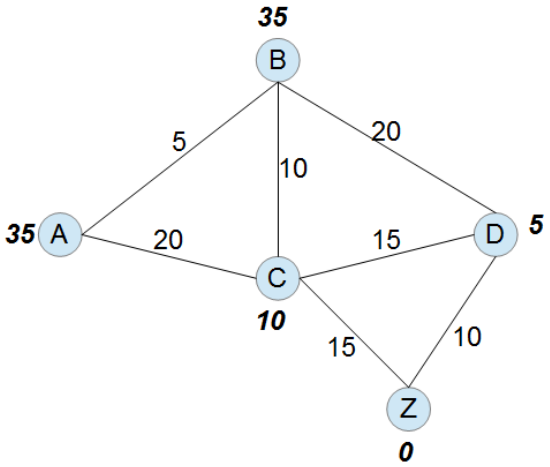
noch in der sortierten Liste befindlichen Pfade. Je nach Tiebreakauflösung ist die linke oder die rechte Abbildung korrekt. Der Lösungspfad lautet (c5,d5,d4,d3,d2,e2,f2,f3,f4,f5).

b) Der erste Pfad der durch dynamische Programmierung entfernt wird (und kein Zyklus ist) lautet (c5,d5,d6) oder alternativ (c5,c6,d6), wenn das Feld d6 zum zweiten Mal erreicht wird (mit gleichen Kosten wie beim ersten Mal). Der jeweils andere Pfad bleibt in der sortierten Listen enthalten.

c) 10 Pfade bzw. 9 Pfade

Aufgabe 4: Auswirkungen von Heuristiken auf A* (20%)

Betrachten Sie folgenden ungerichteten Graphen.



a) Führen Sie mit der so gegebenen Heuristik den A*-Algorithmus aus.

Schritt	*Expandierter Knoten*	*Queue*	*Anmerkungen*
1	*A(35)*	*AC(30), AB(40)*	
2	*AC(30)*	*ACZ(35), AB(40), ACD(40), ACA(75), ACB(65)*	*ACA entfällt wegen Zyklus, ACB wegen dynamischer Progr.*
3	*ACZ(35)*	-	*Z ist Zielknoten*

Der A-Algorithmus liefert den Pfad A-C-Z.*

b) Das Ergebnis ist nicht optimal. Was ist die Ursache?

Die Pfadkosten der gefundenen Lösung betragen 35, während die Kosten der optimalen Lösung A-B-C-Z nur 30 beträgt. Der Grund dafür ist, dass die Heuristik nicht zulässig ist: Sie überschätzt die Kosten für Knoten B (35), dessen tatsächliche Kosten nur 25 beträgt. Deshalb expandiert A den Knoten B zu spät und findet bereits vorher eine Lösung.*

c) Verändern Sie die Heuristik so, dass der A*-Algorithmus eine optimale Lösung liefert, und führen Sie mit dieser Heuristik A* erneut aus.

Wir verändern die Heuristik für B z.B. so, dass sie jetzt 25 beträgt.

Schritt	*Expandierter*	*Queue*	*Anmerkungen*

	Knoten		
1	*A(35)*	*AC(30), AB(30)*	
2	*AB(30)*	*ABC(25), ABD(30), ABA(45)*	*ABA entfällt wegen Zyklus, AC wegen dyn. Progr.*
3	*ABC(25)*	*ABCZ(25), ABD(30), ABCD(35)*	*ABCB und ABCA sind Zyklen*
4	*ABCZ(25)*	*ABD(30), ABCD(35)*	*Z ist Zielknoten*

Aufgabe 5: Iterative Tiefensuche

a) Algorithmusbeschreibung, Vollständigkeits- und Optimalitätseigenschaften:

Bei der iterativen Tiefensuche werden mehrere Tiefensuchen mit beschränkter Tiefe hintereinander durchgeführt, wobei die Suchtiefe der aufeinanderfolgenden Suchen immer weiter erhöht wird.

ITS ist vollständig, wenn der Verzweigungsfaktor endlich ist, und optimal, wenn die Pfadkosten monoton steigend von der Tiefe des Knotens abhängen (es also keine negativen Aktionskosten gibt).

Der Laufzeitaufwand entspricht $O(b^d)$, Speicheraufwand entspricht $O(bd)$.

Mit anderen Worten: iterative Tiefensuche vereinigt die besten Merkmale der Breiten- und der Tiefensuche.

b) Wie viele Knoten generiert die iterative Tiefensuche, wenn ein Baum mit Verzweigungsgrad b = 35 und einer Tiefe d = 5 vollständig durchsucht wird? Wie hoch ist der prozentuale Overhead der iterativen Tiefensuche gegenüber der normalen Tiefensuche?

Generierte Knoten der normalen Tiefensuche:

$1 + 35 + 35^2 + 35^3 + 35^4 + 35^5 = 54.066.636$

Tiefensuche:

$1 + (1+35) + (1+35+35^2) + \ldots + (1+35+\ldots+35^5) =$

$1 + 36 + 1.261 + 44.136 + 1.544.761 + 54.066.636 = 55.656.831$

55.656.831 / 54.066.636 = 1,029. *Das entspricht einem Overhead von 2,9%*

Die iterative Tiefensuche erzeugt bei diesem Suchbaum somit nur etwa 3% mehr Knoten als die normale Tiefensuche.

c) Beziehung zwischen prozentualem Overhead und Verzweigungsfaktor:

Der relative Overhead sinkt mit steigendem Verzweigungsfaktor. Die Terme der Knotenanzahlen für beide Arten der Suche werden dominiert von dem letzten Term, für die maximale Suchtiefe. Für normale Tiefensuche ist dieser b^d, für ITS ist er etwa $b^d + b^{d-1}$. Daraus ergibt sich ein Verhältnis von ca. $(b^d + b^{d-1}) / b^d = 1 + 1/b$, also ein Overhead von etwa 1/b: Je größer der Verzweigungsfaktor, desto mehr nähert sich der Overhead 0 an.

Aufgabe 6: Herausfordernde Variante des „Pfeile werfen"-Problems (10%)

a) Bei einem Suchproblem kann das Optimierungskriterium mit den bekannten Suchbaumalgorithmen nur über *minimale* Pfadkosten definiert werden. (Vergleiche das Originalproblem aus der Übung 02 Suche). In der hier dargestellten Variante sind aber maximale Pfadkosten gefragt, d.h. eine Lösung in maximaler Baumtiefe.

b) Verschiedene Herangehensweisen bieten sich an:

1. Zustand=aktuelle Punktsumme; Aktion=addiere Punkte gemäß möglichen Würfen (16, 18, ..., 40), sofern die Summe 100 nicht überschreitet; Zieltest: aktuelle Punktsumme=100. Tiefensuche einsetzen und dabei immer den kleinst möglichen Pfeil zuerst verwenden. Ein Zustand wird also expandiert, indem der Pfad mit dem kleinsten Nachfolger ganz oben auf den Stack gelegt wird, der Pfad mit dem zweitkleinsten Nachfolger direkt darunter usw. Diese Suchstrategie terminiert mit derjenigen Lösung, die im Baum am tiefsten liegt.
2. (sehr exotische Lösung, die gleich mehrere Prinzipen der Suchprobleme verletzt) A* in Verbindung mit negativen Aktionskosten und negativer Heuristik: Jede Aktion kostet uniform -1. h(x) = -(1) * (100-x). Dann wird auch der Spezialfall mit einer Dartscheibe mit 2 Einträgen (50, 100) richtig gelöst. Der Zielpfad (0,100) hat einen f-Wert von -1 während der Pfad, der zur optimalen Lösung führt (0,50) mit -60 „besser“ bewertet ist und somit expandiert wird, sodass die Lösung (0,50,100) mit Kosten -2 erreicht wird.