Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Linnhoff-Popien, Dr. Feld

mobile and distributed systems group

Online-Hausarbeit 3
Intelligente Systeme im Sommersemester 2020

Abgabetermin: Geben Sie Ihre Lösung im Uni2Work bis zur Abgabefrist am 03.07.2020, 18:59 Uhr, ab! Sollten Sie nachweislich Internetprobleme haben, die eine Abgabe bis 18:59 Uhr nicht ermöglichen, so geben Sie bitte bis 23:59 Uhr ab und schreiben uns parallel dazu eine E-Mail, in der Sie um eine verlängerte Abgabefrist bitten und Ihre Umstände erklären.

Aufgabe 1: Klassische Suche vs. Lokale Suche

(2+2 Pkt.)

In der Vorlesung haben Sie Algorithmen der klassischen Suche (uninformierte Suchstrategien wie etwa Breitensuche oder informierte Suchalgorithmen wie etwa A*-Algorithmus) kennengelernt, aber auch Algorithmen der lokalen Suche (beispielsweise Tabu-Suche). Beide Ansätze können zur Lösung von Problemen verwendet werden.

Das **n-Damen-Problem** kann zum Beispiel mittels der **klassischen Suche** wie folgt gelöst werden: Gestartet wird mit einem leeren Schachbrett. Die Damen werden nacheinander auf ein freies Feld gesetzt und zwar so, dass kein Konflikt entsteht. Auf dem Weg zur Lösung ist also in jedem Schritt eine zwar unvollständige Konfiguration vorhanden, diese ist jedoch stets konfliktfrei. Wenn in einem Zustand keine Möglichkeit mehr für einen konfliktfreien Zug vorhanden ist, dann wird die zuletzt gesetzte Dame wieder entfernt (sogenanntes **Backtracking**) und mit der nächsten möglichen konfliktfreien Konfiguration fortgefahren. Somit wird der Suchraum strukturiert durchsucht, bis alle n Damen konfliktfrei gesetzt sind.

Bei der **lokalen Suche** kann es sich wie folgt verhalten: Die Suche wird mit einem Zustand begonnen, bei der sich bereits n Damen auf dem Brett befinden. Nun wird iterativ versucht, den Zustand zu verbessern. Dazu wird in jedem Schritt genau eine Dame versetzt. Ist der neue Zustand mindestens so gut wie der Vorherige (d.h. es existieren nicht mehr Konflikte als vorher), so wird mit diesem Zustand fortgefahren. Wie Sie wissen, werden je nach Algorithmus schlechtere Zustände ebenso akzeptiert oder abgelehnt. Die Ausführung dauert an, bis entweder ein Zielzustand erreicht ist (alle Konflikte beseitigt) oder anderweitige Abbruchkriterien gelten.

Die Problemformulierung des **n-Damen-Problems** sowohl für die klassische als auch für die lokale Suche kann entsprechend folgendermaßen aussehen:

Klassische Suche	
Anfangszustand	Ein leeres Brett.
Nachfolgerfunktion	Setze eine Dame auf ein freies Feld, sodass kein Konflikt entsteht. Ist dies nicht möglich, führe Backtracking durch.
Zieltest	Alle n Damen sind auf dem Brett platziert und die Anzahl der Konflikte ist gleich 0.
Pfadkosten	0

Lokale Suche	
Anfangszustand	Eine beliebige Anordnung von n Damen auf dem Brett.
Nachfolgerfunktion	Eine beliebige Dame wird auf ein anderes freies Feld versetzt.
Zieltest	Die Anzahl der Konflikte ist gleich 0.
Pfadkosten	0

Im Folgenden sollen Sie das **Knotenfärbungsproblem** und das **Erfüllbarkeitsproblem der Aussagenlogik** formalisieren. Dabei müssen Sie die Nachfolgerfunktion nicht bis ins letzte Detail ausformulieren, sondern lediglich das generelle Vorgehen beschreiben. Wichtig ist, dass der **Unterschied** zwischen der Vorgehensweise der klassischen Suche und der lokalen Suche deutlich wird.

a. Geben Sie für das **Knotenfärbungsproblem** sowohl für die klassische als auch für die lokale Suche eine Problemformalisierung an.

Klassische Suche	
Anfangszustand	
Nachfolgerfunktion	
Zieltest	Jeder Knoten besitzt genau eine Farbe und keine Kante verbindet Knoten gleicher Farbe.
Pfadkosten	0

Lokale Suche	
Anfangszustand	
Nachfolgerfunktion	
Zieltest	Jeder Knoten besitzt genau eine Farbe und keine Kante verbindet Knoten gleicher Farbe.
Pfadkosten	0

b. Geben Sie nun für das **Erfüllbarkeitsproblem der Aussagenlogik** sowohl für die klassische als auch für die lokale Suche eine Problemformalisierung an.

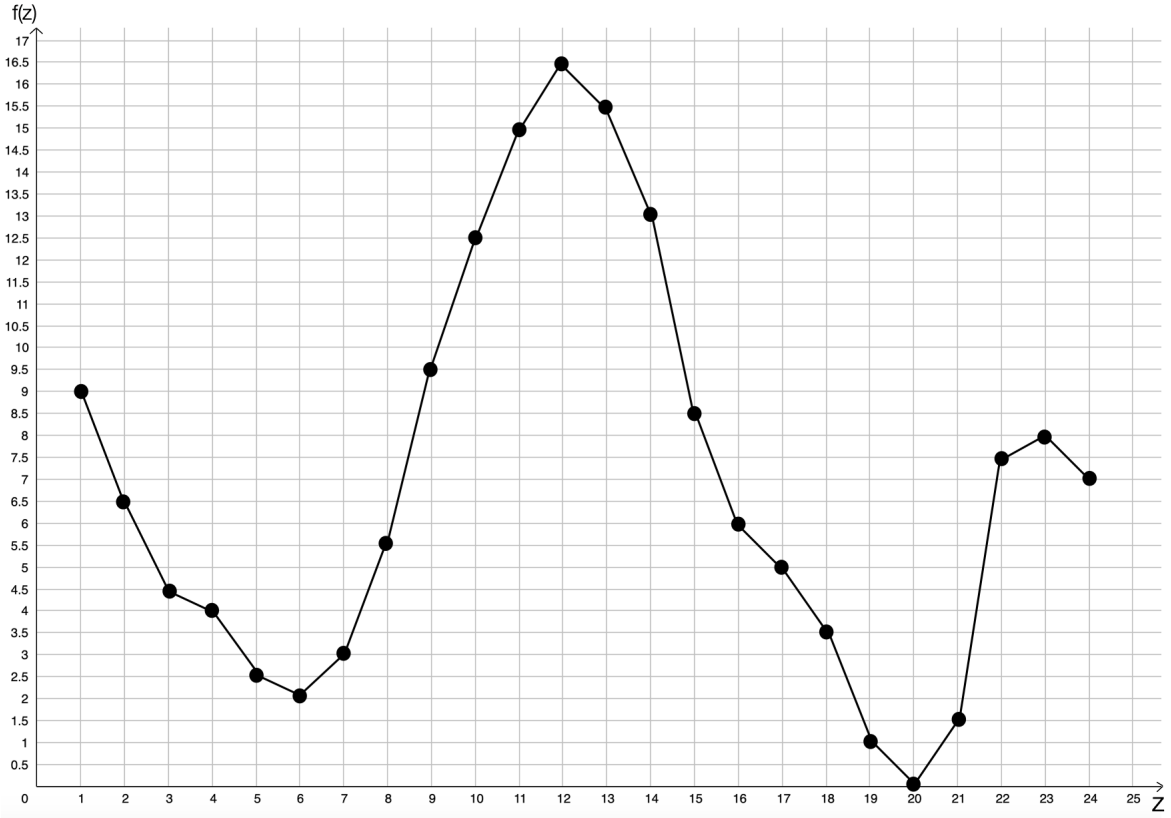
Klassische Suche	
Anfangszustand	
Nachfolgerfunktion	
Zieltest	Jeder Variable ist ein Wahrheitswert zugewiesen und die Formel ist erfüllbar.
Pfadkosten	0

Lokale Suche	
Anfangszustand	
Nachfolgerfunktion	
Zieltest	Jeder Variable ist ein Wahrheitswert zugewiesen und die Formel ist erfüllbar.
Pfadkosten	0

Aufgabe 2: Local Beam Search

(3+2 Pkt.)

In der folgenden Abbildung ist die Lösungslandschaft eines Optimierungsproblems gegeben, welches wir maximieren möchten. Die Fitness eines Zustands $z \in \{1, \ldots, 24\}$ ergibt sich aus dem Funktionswert $f(z)$. Der linke Nachbar eines Zustands z ist $z_l = z - 1$, der rechte Nachbar ist $z_r = z + 1$. Entsprechend besitzt der Zustand $z = 1$ nur einen rechten Nachbarn ($z_r = 2$) und $z = 24$ nur einen linken Nachbarn ($z_l = 23$). Alle übrigen Zustände haben jeweils einen linken und einen rechten Nachbarn. Zur besseren Übersicht sind die Funktionswerte aller möglichen Zustände durch Punkte in der Lösungslandschaft markiert.



Die konkreten Funktionswerte aller Zustände sind außerdem mit folgender Tabelle gegeben:

z	1	2	3	4	5	6	7	8	9	10	11	12	13
f(z)	9.0	6.5	4.5	4.0	2.5	2.0	3.0	5.5	9.5	12.5	15.0	16.5	15.5

z	14	15	16	17	18	19	20	21	22	23	24
f(z)	13.0	8.5	6.0	5.0	3.5	1.0	0.0	1.5	7.5	8.0	7.0

a. Führen Sie den Algorithmus **Local Beam Search** entsprechend der Vorlesung (Seiten 201-203) für $k = 3$ mit der initialen Zustandsmenge $\{4, 15, 22\}$ aus. Geben Sie in folgender Tabelle für jeden Iterationsschritt die aktuell verwalteten Zustände (*k-current-states*), den höchsten Fitnesswert der aktuell verwalteten Zustände (*max_state_value*), die Nachbarn der aktuell verwalteten Zustände (*neighbors*) sowie den Fitnesswert des aktuellen besten Nachbarn (*max_neighbor_value*) an. Beachten Sie, dass auch die aktuell verwalteten Zustände in derselben Iteration wieder zu Nachbarn werden können, da zwei **benachbarte** aktuell verwaltete Zustände ebenso Nachbarn voneinander sind. Beispiel: Wenn die Menge der aktuell verwalteten Zustände $\{1, 2, 3\}$ ist, dann ist die Menge der Nachbarn entsprechend $\{1, 2, 3, 4\}$ (da 1 Nachbar von 2, 2 Nachbar von 1 und 3, 3 Nachbar von 2, und 4 Nachbar von 3 ist). Diese Annahme gilt ebenso für Teilaufgabe b).

Führen Sie nun die folgende Tabelle so lange fort, bis der Algorithmus terminiert. Welcher Zustand wird als endgültige Lösung zurückgegeben?

iteration	k-current-states	max_state_value	neighbors	max_neighbor_value
1	$\{4, 15, 22\}$	8.5	$\{3, 5, 14, 16, 21, 23\}$	13.0
2	...	...	...	
...	...	...	...	

b. Nun sollen Sie zeigen, dass die Diversität der Startzustände einen starken Einfluss auf die Güte des Ergebnisses hat. Wählen Sie dazu $k = 3$ „schlechte“ **verschiedene** initiale Zustände derart, dass der Algorithmus **nicht** zum globalen Maximum $z = 12$ findet und stattdessen in einem lokalen Maximum terminiert. Zeigen Sie dies, indem Sie auch hier die folgende Tabelle ausfüllen und für jeden Iterationsschritt die aktuell verwalteten Zustände und Nachbarn mitsamt deren jeweils bester Güte angeben. Beachten Sie, dass Sie die lokalen Maxima, d.h. die Zustände $z = 1$ und $z = 23$ **nicht** in Ihre initiale Zustandsmenge wählen dürfen.

iteration	k-current-states	max_state_value	neighbors	max_neighbor_value
1	...	...	...	...
2	...	...	...	...
...	...	...	...	...

Aufgabe 3: Simulated Annealing

(6 Pkt.)

Gegeben ein **Optimierungsproblem** $f(x) = x$ mit $x \in \{1, \ldots, 9\}$, welches wir maximieren möchten. Der Algorithmus startet mit der **Initiallösung** $x = 5$. Die **Zeit** ist definiert als $t = \{1, \ldots, 4\}$. Ferner ist ein **Abkühlprozess** durch $\alpha(t) = 4 - t$ definiert.

Wie Sie wissen, spielt Wahrscheinlichkeit eine wichtige Rolle bei der Ausführung von Simulated Annealing. Das heißt, wir müssen nun einen Mechanismus definieren, wie Sie in Ihrer Abgabe **Zufall** modellieren. Dazu nehmen Sie die **letzte** Ziffer Ihrer Matrikelnummer und teilen diese durch 10. Sie erhalten also eine Zahl $z \in \{0,0;\ 0,1;\ \ldots;\ 0,8;\ 0,9\}$.
Wenn der Algorithmus nun einen Lösungskandidaten mit einer Wahrscheinlichkeit $p \in [0,1]$ akzeptieren soll, dann prüfen Sie, ob $z \leq p$ zutrifft. Wenn beispielsweise die letzte Ziffer Ihrer Matrikelnummer 3 lautet, dann gilt für Sie $z = 0,3$. Wenn der Lösungskandidat nun mit einer Wahrscheinlichkeit von $p = 0,9$ akzeptiert werden soll, dann prüfen Sie, ob $z \leq p$ gilt. Im gegebenen Beispiel ist $0,3 \leq 0,9$ wahr, demnach wird der Kandidat akzeptiert. Wenn beispielsweise ein Kandidat mit einer Wahrscheinlichkeit von $0,2$ akzeptiert werden soll, dann ist $z \leq p$ falsch (da $0,3 \not\leq 0,2$) und der Kandidat wird abgelehnt.

Wie sie außerdem wissen, wird bei Simulated Annealing in jedem Iterationsschritt ein **zufälliger Nachbar** gewählt. Auch hier benötigen Sie eine Strategie. Wir definieren nun Folgendes: wenn die **vorletzte** Ziffer Ihrer Matrikelnummer aus $\{0,1,2,3\}$ stammt, dann gilt für Sie $n = [L, L, R]$. Wenn die vorletzte Ziffer Ihrer Matrikelnummer aus $\{4,5,6\}$ stammt, dann gilt $n = [L, R, L]$ und wenn sie aus $\{7,8,9\}$ stammt, entsprechend $n = [R, L, L]$. Beispiel: wenn die vorletzte Ziffer Ihrer Matrikelnummer die 8 ist, dann gilt bei Ihnen $n = [R, L, L]$.
Mittels n definieren wir nun die "zufällige" Wahl eines Nachbarn. Im Beispiel mit $n = [R, L, L]$ ist es also so, dass im ersten Iterationsschritt der **rechte** (R) Nachbar gewählt wird, in den zwei folgenden Iterationsschritten hingegen jeweils der **linke** (L) Nachbar. Die Begriffe linker Nachbar und rechter Nachbar haben folgende Bedeutung: Der linke Nachbar eines Zustands z ist $z_l = z-1$, der rechte Nachbar ist $z_r = z + 1$. Wenn Sie also beispielsweise den Zustand $z = 3$ betrachten, so ist dessen linker Nachbar $z_l = 2$ und dessen rechter Nachbar $z_r = 4$.

Führen Sie nun Simulated Annealing mit den oben beschriebenen Angaben und Konventionen aus. Verwenden Sie dazu den abgebildeten Pseudocode, wie Sie ihn aus der Vorlesung (Seite 208) kennen, sowie die folgende Tabelle.

```
1: function SIMULATED-ANNEALING( problem, schedule) returns a solution state
2:     inputs: problem, a problem
3:             schedule, a mapping from time to "temperature"
4:     local variables: current, a node
5:                      next, a node
6:                      T, a "temperature" controlling prob. of downward steps
7:     current ← MAKE-NODE(INITIAL-STATE[problem])
8:     for t ← 1 to ∞ do
9:         T ← schedule[t]
10:        if T = 0 then return current
11:        next ← a randomly selected successor of current
12:        ΔE ← VALUE[next] – VALUE[current]
13:        if ΔE > 0 then current ← next
14:        else current ← next only with probability e^(Δ E/T)
```

Füllen Sie nun die Tabelle aus und tragen Sie die entsprechenden aktuellen Werte in die Felder. Mit "(falls true/false)" ist gemeint, dass Sie diese Zeile nur ausfüllen, wenn der logische Ausdruck in gegebener Zeile entsprechend wahr/falsch ist.

Zeile	
%	z=
%	n=
1	problem =
2	schedule =
7	current =
8	t =
9	T =
10 (falls true)	current =
11	next =
12	value[next] =
12	value[current] =
12	delta_E =
13 (falls true)	current =
14 (falls false)	probability =
14 (falls false)	current =
8	t =
9	T =
10 (falls true)	current =
11	next =
12	value[next] =
12	value[current] =
12	delta_E =
13 (falls true)	current =
14 (falls false)	probability =
14 (falls false)	current =
8	t =
9	T =
10 (falls true)	current =
11	next =
12	value[next] =
12	value[current] =
12	delta_E =
13 (falls true)	current =
14 (falls false)	probability =
14 (falls false)	current =

Selbstständigkeitserklärung

Hiermit versichere ich, dass die abgegebene Lösung alleinig durch mich angefertigt wurde und ohne die Hilfe Dritter entstanden ist. Insbesondere habe ich keine Lösungen von Dritten teilweise oder gänzlich abgegeben.

Matrikelnummer, Name	Ort, Datum

Unterschrift