Bachelor Projekt — Grundlagen der Künstlichen Intelligenz

B. Nebel, T. Engesser, R. Mattmüller, D. Speck
Wintersemester 2019/20

Universität Freiburg
Institut für Informatik

Übungsblatt 2

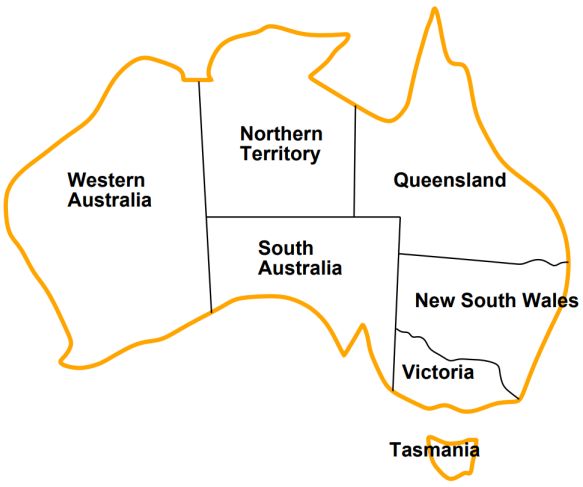
Aufgabe 2.1 (Constraint-Satisfaction-Problem (CSP))

(a) Mit PYTHON-CONSTRAINT vertraut machen:

> PYTHON-CONSTRAINT ist ein Modul für PYTHON, welches einen CSP-Solver beinhaltet. Ein CSP-Problem besteht aus einer Menge von Variablen mit dazugehörigen (endlichem) Wertebereichen und Bedingungen zwischen Variablen, welche wiederum festlegen, welche Kombinationen von Werten der Variablen zulässig sind.

Lies die Modulbeschreibung von PYTHON-CONSTRAINT[1], installiere PYTHON-CONSTRAINT, wie dort beschrieben, und löse das folgende Problem (Einfärben einer Landkarte).

- Variablen: $X = \{WA, NT, Q, NSW, V, SA, T\}$
- Wertebereiche: $D_x = \{0, 1, 2\}$ für alle Variablen $x \in X$
- Bedingungen: Benachbarte Regionen müssen unterschiedliche Farben zugewiesen bekommen. Zum Beispiel $NSW \neq V$.



(b) Mit MINESWEEPER vertraut machen:

> MINESWEEPERist ein Ein-Spieler-Spiel, bei dem ein Spieler durch logisches Denken herausfinden soll, unter welchem Feld sich eine Mine befindet (`https://de.wikipedia.org/wiki/Minesweeper`). Mache dich mit Minesweeper vertraut, verstehe die Regeln. Hier kannst du online Minesweeper spielen (http://minesweeperonline.com/) und überprüfen, ob du die Regeln verstanden hast.

(c) MINESWEEPER mit Hilfe von CSP lösen[2]:

[1] `https://pypi.org/project/python-constraint/`
[2] `https://www.clear.rice.edu/comp440/old/fall2007/handouts/pa3.pdf`

Klone das GIT-Repository [3]. Die Aufgabe ist es ein CSP zu konstruieren, welches ermöglicht Minesweeper zu spielen/lösen. Hierfür werden relevante Feld des Spielbretts als eine Variable mit Wertebereich $\{0, 1\}$ repräsentiert, wobei 0 für ein Feld ohne Mine und 1 für ein Feld mit einer Mine steht. Betrachten wir zum Beispiel das folgende 7x7 Spielbrett. Wir bezeichnen Felder, welche noch verdeckt sind und an ein aufgedecktes Feld grenzen, als das Randgebiet. In diesem Bespiel besteht das Randgebiet aus den Feldern (0,5), (1,5), (2,5), (3,0), (3,1), (3,5), (4,1), (4,5), (5,1), (5,2), (5,3), (5,4) und (5,5).

	0	**1**	**2**	**3**	**4**	**5**	**6**
0	0	0	0	0	1		
1	0	0	0	0	2		
2	1	1	1	0	2		
3			1	0	1		
4			3	2	2		
5							
6							

Fülle in der Datei *minesweeper_csp.py* die folgen Methoden.

- ADD_VARIABLES: Füge für jede Zelle des Randgebietes eine Variable zum CSP hinzu.
- ADD_CELL_CONSTRAINTS: Füge eine Bedingung hinzu, welche fordert, dass für jedes aufgedeckte Feld, das an ein Randfeld angrenzt, die Zahl, die in diesem Feld angezeigt wird, die Summe der Minen in den angrenzenden verdeckten Feldern entspricht.
- ADD_MINES_CONSTRAINT: Füge eine Bedingung hinzu, welche fordert, dass die Anzahl an Minen der Randgebietfelder nicht größer sein darf, als die Anzahl an Minen im Spiel.
- NEXT_CELL: Berechne das Feld, welches als nächstes aufgedeckt werden soll. Hierbei sollen alle möglichen Lösungen berücksichtig werden und das vielversprechendste Feld zurück gegeben werden. In anderen Worten, es soll ein Feld gewählt werden, das in jeder bzw. in den meisten Lösungen keine Mine enhält.

Das Spiel lässt sich mit dem Aufruf *python3 minesweep_main.py* starten. Per Default wird das folgende Feld gespielt:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	0	1	1	1	0	0	0
1	0	0	0	0	2	X	2	0	0	0
2	1	1	1	0	2	X	2	1	1	1
3	1	X	1	0	1	1	1	1	X	1
4	2	3	3	2	2	2	1	1	1	1
5	1	X	X	2	X	X	1	0	0	0
6	1	2	2	2	2	2	1	0	0	0
7	0	0	1	1	1	0	0	1	2	2
8	0	0	1	X	1	0	0	1	X	X
9	0	0	1	1	1	0	0	1	2	2

Die Datei *python3 minesweep_main.py* hat folgende Parameter.

[3] `https://gkigit.informatik.uni-freiburg.de/dspeck/minesweeper-csp`

- -board D M: ein zufälliges Brett mit größe D x D und M Minen wird erstellt
- –fast: "fast mode"

(d) Ansatz verbessern:

Identifiziere die Schwachstellen und Probleme des Ansatzes und erarbeite Lösungsvorschläge.

Aufgabe 2.2 (Linear Programming (LP))

(a) Mache dich mit LPs und SCIPY.OPTIMIZE vertraut[4].

SCIPY.OPTIMIZE ist ein Modul für PYTHON, welches einen LP-Solver beinhaltet. Ein LP-Problem beschäftigt sich mit der Optimierung einer linearen Zielfunktion. Diese Zielfunktion wird durch lineare Gleichungen und Ungleichungen eingeschränkt.

(b) Es gibt eine natürliche LP-Formulierung für das kürzeste Wegproblem[5].

Gegeben einen gerichteten Graphen (V, E) mit Startknoten s, Zielknoten t und Kosten w_{ij} für jede Kante (i, j) in E, definieren wir ein LP mit Variablen x_{ij}:

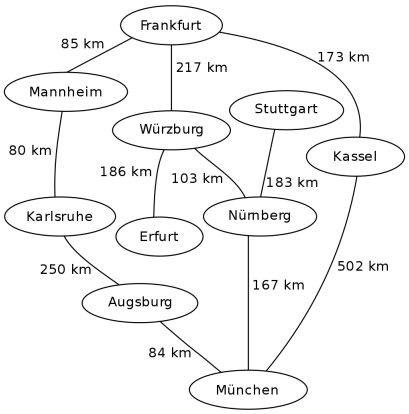
$$\min \sum_{(i,j)\in E} w_{ij}x_{ij}$$

$$x_{ij} \geq 0 \text{ for all } (i,j) \in E$$

$$\sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1 \text{ if } i = s \\ -1 \text{ if } i = t \\ 0 \text{ otherwise} \end{cases}$$

Die Idee dahinter ist, dass die Variable x_{ij} eine Indikatorvariable dafür ist, ob die Kante (i, j) Teil des kürzesten Weges ist. Mit anderen Worten, wenn die Variable x_{ij} den Wert 1(0) annimmt, ist die Kante (i, j) (nicht) Teil des kürzesten Pfades. Als Bedingung ist es erforderlich, dass genau eine ausgehende Kante vom Startknoten s und genau eine eingehende Kante vom Zielknoten t ausgewählt wird. Für alle anderen Knoten muss die Anzahl der ausgewählten ein- und ausgehenden Kanten gleich sein.

Die Aufgabe ist es den kürzesten Weg des folgenden Graphens[6] als LP zu formulieren und mit SCIPY.OPTIMIZE zu berechnen. Hierbei ist Frankfurt der Startknoten und München der Zielknoten. Beachte, dass Kanten in beide Richtungen gültig sind.



[4]E.g., http://www.vision.ime.usp.br/~igor/articles/optimization-linprog.html
[5]https://en.wikipedia.org/wiki/Shortest_path_problem#Linear_programming_formulation
[6]https://de.wikipedia.org/wiki/Dijkstra-Algorithmus#/media/File:MapGermanyGraph.svg