Aufgabenblatt 2: Constraints und Spiele

Abgabetermin: 16.11.2016

Aufgabe 1: Layoutproblem (20%)

a) Repräsentieren Sie dieses Layoutproblem als Constraint Satisfaction Problem (CSP).
Variablen: $D_A=D_L=D_O=D_G=\{1,2,3,4\}$. Die Zahlen stehen für die Positionen.
Constraints:
(1) G<L,
(2) A+1=L
(3) L+1>O v L-1<O (darstellbar als |L-O| > 1) (|...| sind Betragsstriche, der Betrag einer Subtraktion ist immer positiv).
(4) G≠2
(5) O≠4
(6) A≠O
(7) O≠G
(8) A≠G

Statt (6) bis (8) könnte man auch mittels all-different ({A,G,L,O}) die paarweise Disjunktheit ausdrücken. Weil zwischen 2 Variablen (unten ausgedrückt durch eine Kante) aber nur 1 Constraint existieren kann, müsste man z.B. (1) dann als G<L ∧ G≠L ausdrücken. Das ist aber redundant, denn G<L impliziert bereits G≠L.

b) Zeichnen Sie den Constraintgraph.

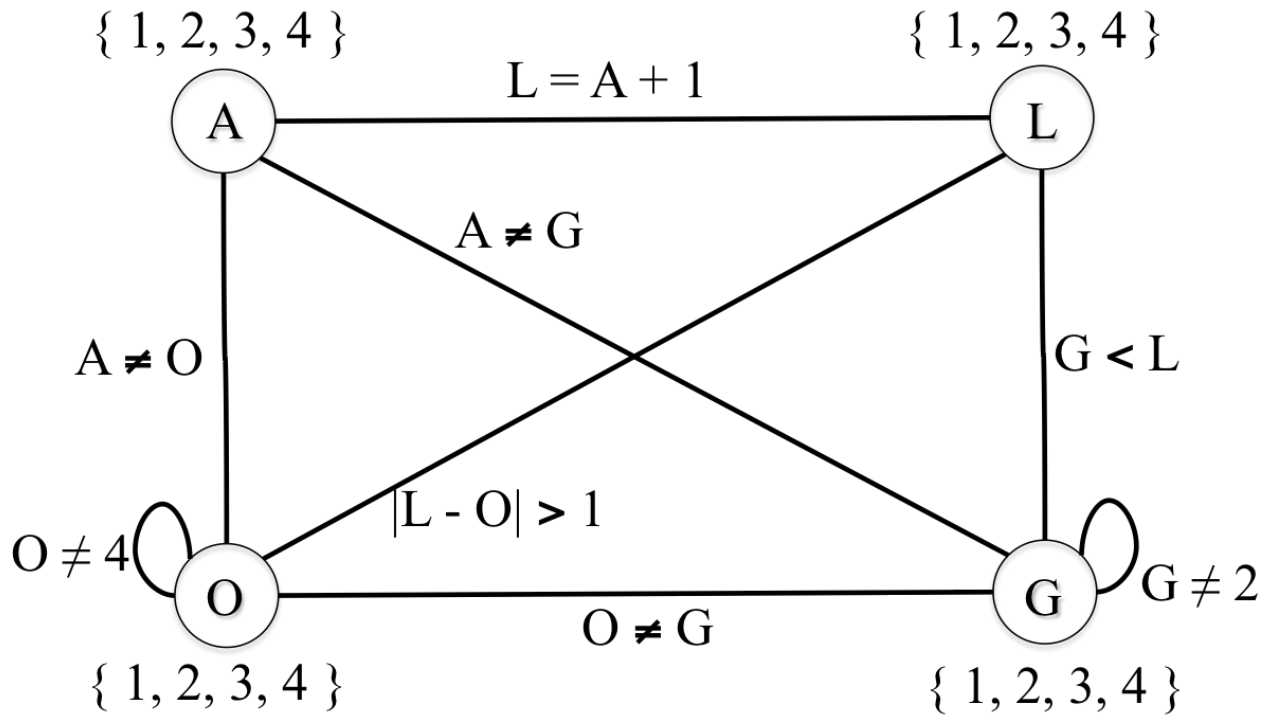


Abb. 1: Layout-CSP

c) Stellen Sie nacheinander 1- und 2-Konsistenz her.

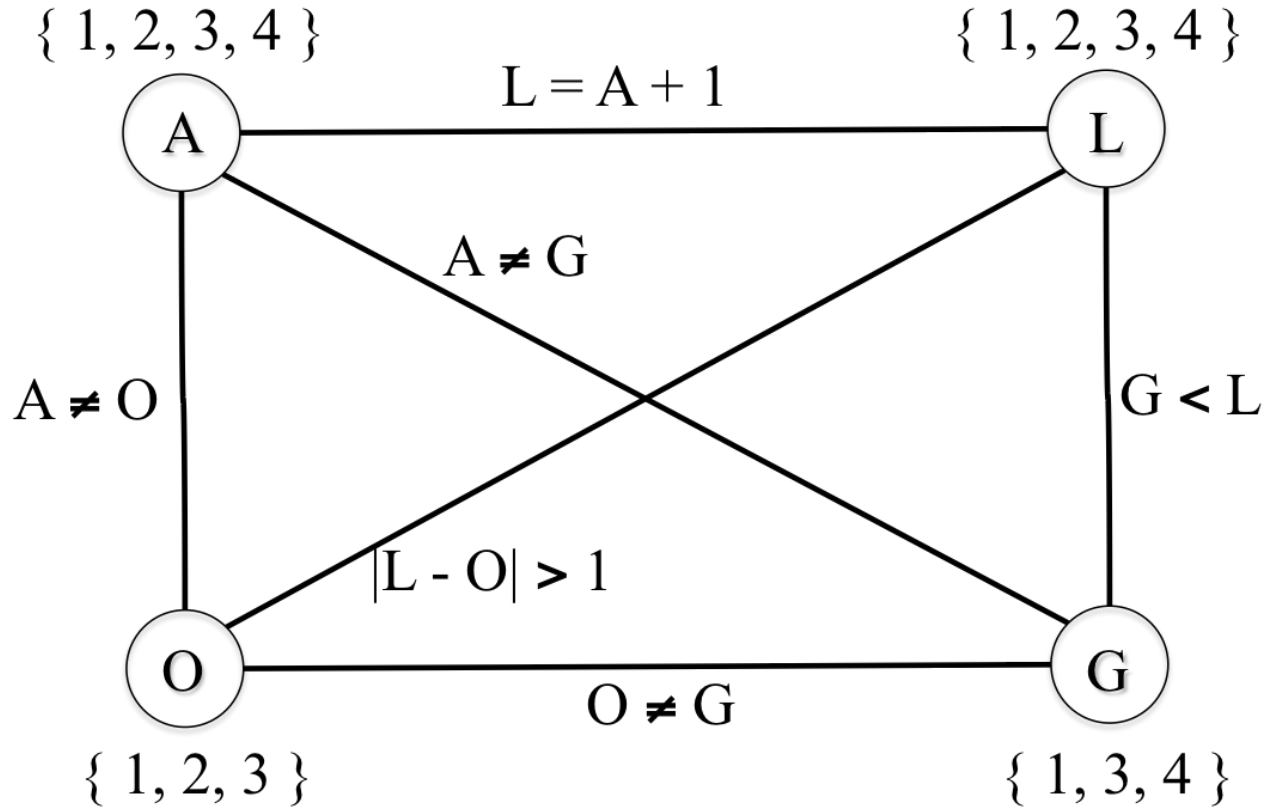


Abb. 2: 1-konsistentes CSP

Die einstelligen Constraints (4) und (5) sind ausgewertet und können aus dem Constraintgraphen entfernt werden. Die Wertebereiche der Variablen O und G wurden durch die Auswertung der 1-stelligen Constraints auf jeweils 3 Werte eingeschränkt:

D_A={1,2,3,4}
D_L={1,2,3,4}
D_O={1,2,3}
D_G={1, 3,4}

Berechnung der 2-Konsistenz:
Es folgt eine kurze Simulation des AC3-Algorithmus, um 2-Konsistenz zu erreichen, Ausgangspunkt ist das 1-konsistente CSP:

Die Ungleichheitsconstraints (6), (7) und (8) schränken wenig ein, sie führen erst dann zu einer Veränderung, wenn eine Variable nur noch einen einstelligen Wertebereich hat. Dieser Wert wird dann aus der Domain der anderen Variable entfernt. Die ≠-Constraints werden daher in der Queue ganz hinten einsortiert.

Ausgangssituation ist der 1-konsistente Zustand (siehe Abb. 2):

D_A={1,2,3,4}
D_L={1,2,3,4}
D_O={1,2,3}
D_G={1,3,4}

Und nun die Auswertung der Constraints. Dargestellt sind nachfolgend nur Constraints, die auch Wertebereichseinschränkungen zur Folge haben:

(1) → D_L={2,3,4}, D_G={1,3}

(2) → D_A={1,2,3} (D_G={1,3} unverändert)
(5) → D_L={3,4}, D_O={1,2}
(1) → D_A={2,3} (D_L={3,4} unverändert)
alle weiteren Auswertungen führen zu keinen Veränderungen, der Algorithmus terminiert.
Abbildung 3 zeigt das 2-konsistente CSP: Alle Variablen haben 2-stellige Domains.

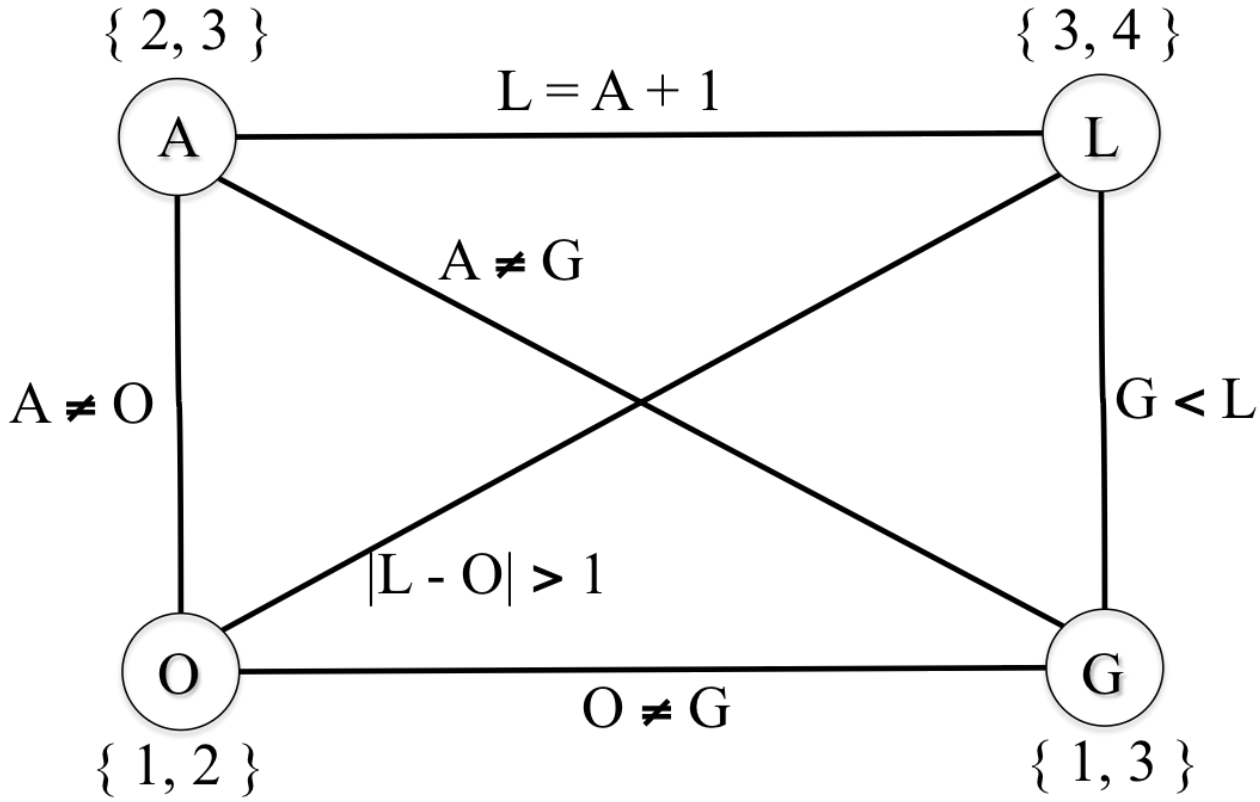


Abb. 3: 2-konsistentes CSP

d) Lösen Sie das 2-konsistente CSP mittels Backtracking und Forward Checking unter Einsatz geeigneter Heuristiken.

2-konsistentes CSP: D_A={2,3}, D_L={3,4}, D_O={1,2}, D_G={1,3}

Lösen mit BT+FC+MRV+MCV:
MRV und MCV bevorzugen keine Variable. Wir beginnen willkürlich mit A

1. A=2.
   FC wertet (2), (6), (8) aus: D_L:{3}; D_O:{1}; D_G:{1,3}
   Nun liefern MCV {O,G,L} und MRV {O, L}. Wir wählen O.
2. O=1.
   FC wertet (3) und (7) aus ((6) wurde bereits ausgewertet und muss nicht erneut geprüft werden): D_L:{3} (unverändert), D_G:{3}.
   Erneut bringt der Einsatz von Heuristiken keine Entscheidung
3. L=3
   FC erkennt verletztes Constraint (1)
   Backtracking auslösen, der einzige Backtrackpunkt liegt bei der 1. Belegung

1. A=3.
   FC wie oben ergibt D_L:{4}; D_O:{1,2}; D_G:{1}
   L und G sind MRV und MCV. Wir wählen L
2. L=4
   FC wertet (1) und (3) aus: D_G:{1} (unverändert); D_O:{1,2} (unverändert)
   Nun ist G MRV
3. G=1
   FC wertet (7) aus: D_O:{2}
   Nun sind alle Constraints getestet, O kann ohne weiterer Prüfung mit 2 belegt werden:
4. O=2
   Das CSP ist gelöst: A=3, L=4, G=1, O=2.
   Die Buchstabenfolge lautet: GOAL

e) Lösen Sie das 1-konsistente CSP mittels Backtracking und Forward Checking unter Einsatz geeigneter Heuristiken.

1-konsistentes CSP:

D_A={1,2,3,4}
D_L={1,2,3,4}
D_O={1,2,3}
D_G={1,3,4}

Dieses Mal BT+FC mit der Gradheuristik zur Variablenauswahl und Verwendung von MRV im Falle einer uneindeutigen Auswahl:

1. O=1

   D_A={2,3,4}; D_L={3,4}; D_G={3,4}
2. G=3 (zur Wahl stände auch L mit demselben Grad 2 und derselben Domaingröße 2)

   D_A={2,4}; D_L={4}

3. L=4 (Gradheuristik erneut uneindeutig, aber MRV zugunsten von L)

   D_A={} Backtracking auslösen

2. G=4

   D_A={2,3}; D_L={} Backtracking auslösen

1. O=2

   D_A={3,4}; D_L={4}; D_G={1,3,4} (unverändert)

2. L=4

   D_A={3}; D_G={1,3}

3. A=3

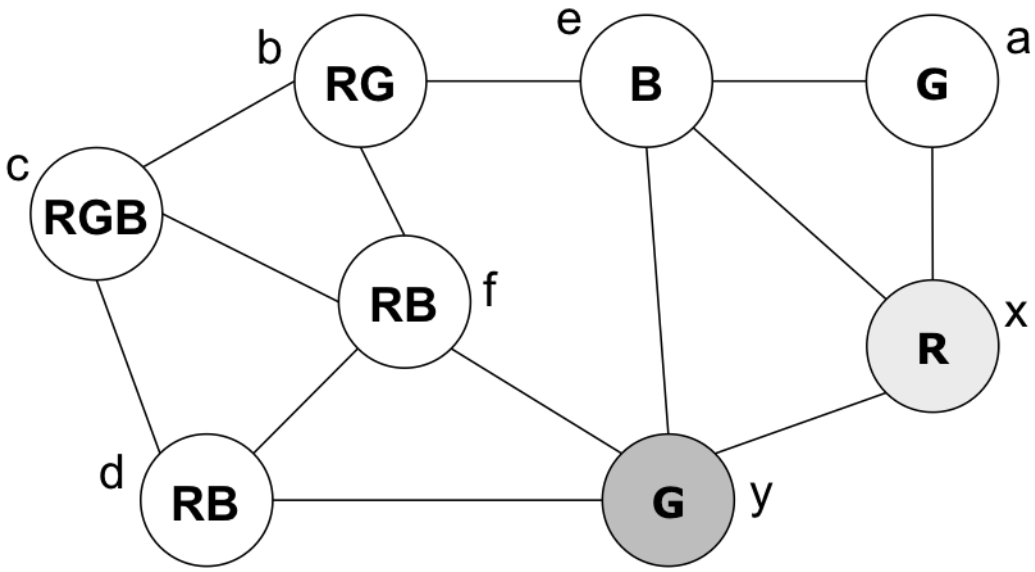
   D_G={1}

4. G=1

   CSP gelöst mit O=2, L=4, A=3, G=1

Aufgabe 2 – Graphenfärbeproblem

a) Der 2-konsistente Zustand ist in der nachfolgenden Grafik abgebildet.



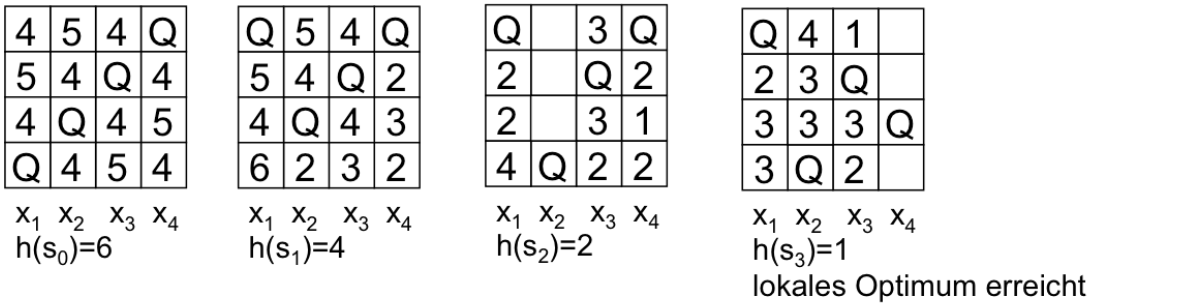
b) und c) Unsinnig wäre, die Variable a zu belegen, weil dies keinen Effekt hätte (die Constraints a≠e und a≠x sind ja bereits „vollständig" ausgewertet worden und ohne Änderung in mindestens einem Wertebereich würde die 2-Konsistenzprüfung nichts Neues ergeben (AC-3 würde alle Constraints einmal prüfen ohne dass irgend eine Variable eine Wertebereichseinschränkung erfahren würde)).
Sinnvoll ist die Belegung von f (z.B. f=R), weil dann der AC-3-Algorithmus D_b={G}, D_d={B}, D_c={} berechnet und die Inkonsistenz feststellen würde. Nach f=B würde AC-3 D_b={R}, D_d={R}, D_c={G} einen Zustand mit lauter 1-stelligen Wertebereichen errechnen.
Mit derselben Begründung sind auch die Variablen **{b, c, d}** geeignete Kandidaten für die erste Belegung. Der Effekt wäre derselbe, entweder würde die Inkonsistenz festgestellt werden (z.B. nach c=B oder nach c=R) oder alle Domainvariablen besitzen einstellige Wertebereiche nach c=G.

Aufgabe 3 – Lokale Suche

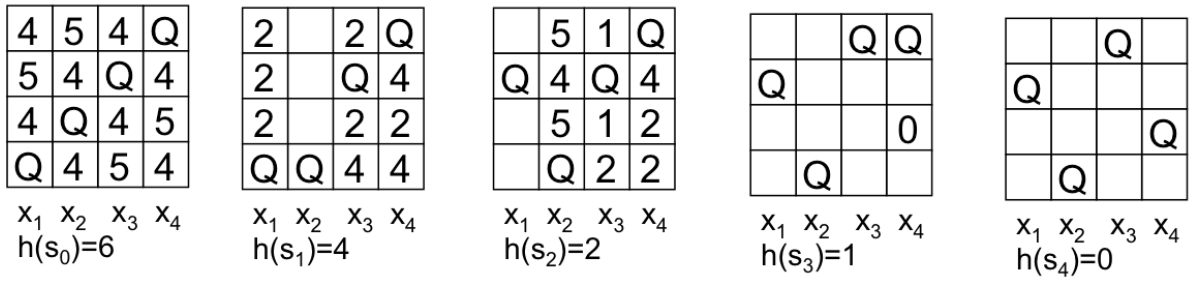
In die Felder sind die heuristischen Werte eingetragen. Die heuristischen Werte für die Spalte der gerade bewegten Dame wurden in der Musterlösung nicht notiert, weil sie ja zu einer Verschlechterung führen *müssen*. Die lokale Suche, die nur den aktuellen Zustand kennt, weiß dies natürlich nicht und würde auch diese Aktionen zur Nachbarschaft zugehörig identifizieren.
a)

I. Nach 4 Schritten endet die lokale Suche im lokalen Optimum.
II. Wir wählen „zufällig" immer die „richtige" Aktion, sodass die Suche nach 4 zustandsverbessernden Aktionen ins Ziel führt.

I.



II.



b) Siehe a) I. In der Nachbarschaft existiert kein Zustand mit einer besseren heuristischen Bewertung als der aktuelle Zustand. Im Allgemeinen weiß man nicht, ob ein gegebenes Problem eine Lösung hat. Wollt man beispielsweise 5 Damen auf dem 4x4-Brett platzieren, kann der Zielzustand mit h=0 gar nicht erreicht werden. Man weiß im Allgemeinen also nicht, ob die gefundene Lösung tatsächlich die beste ist oder nicht. (Dies ist nur für h(x)=0 eindeutig der Fall.)

c)

I. Restart mit anderer Konfiguration. Die aktuell beste Lösung sollte zwischengespeichert werden.
II. Vergrößerung der Nachbarschaft im lokalen Optimum, indem man 2 (oder sogar noch mehr) Damen simultan versetzt.
III. Tabusuche mit möglichst viel Speicher, der verwendet wird, um Zustandswiederholungen zu vermeiden. Dann wird jeweils die beste Aktion gewählt, die nicht notwendigerweise zu einer Zustandsverbesserung führen muss.

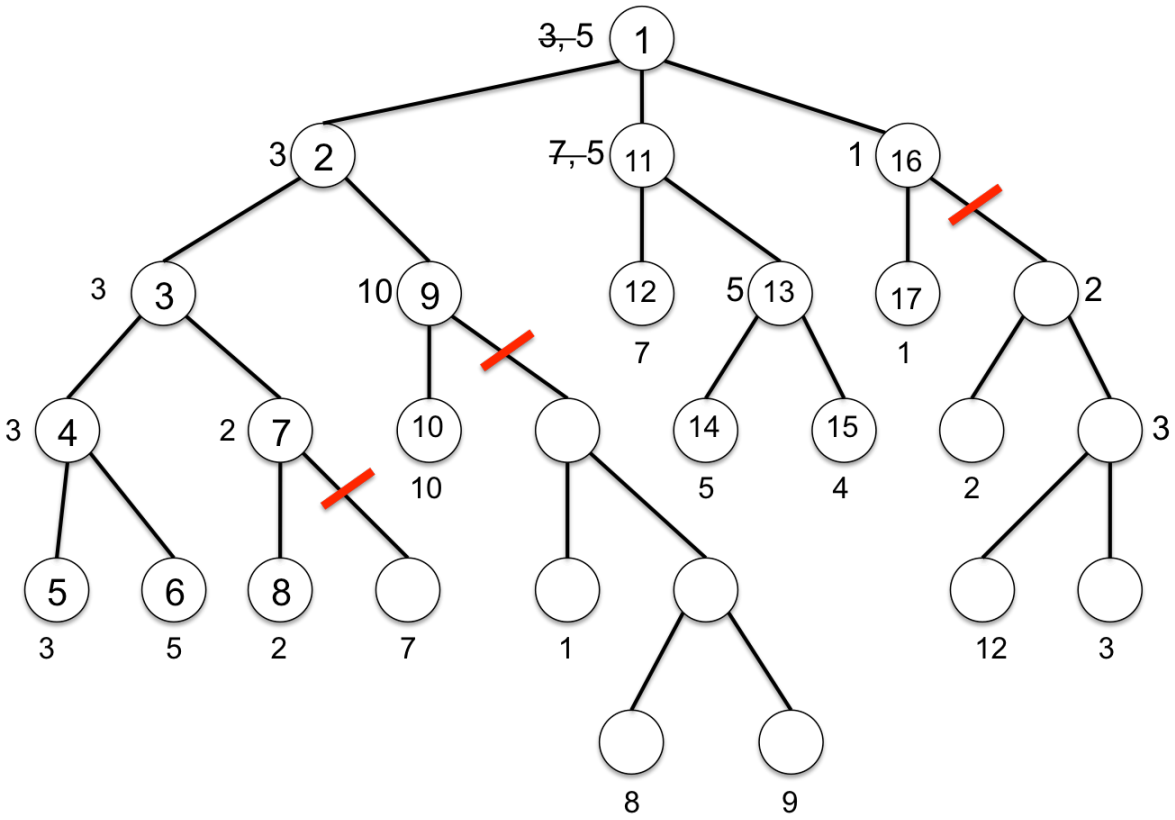
Die Verbesserungen I und II kosten Zeit aber keinen zusätzlichen Speicher. I kann beliebig lange dauern, während II irgendwann terminieren muss, weil in jedem Schritt eine Verbesserung stattfindet. III kostet zusätzlichen Speicher, dafür weniger Rechenzeit als II und I.

Aufgabe 4: Minimax

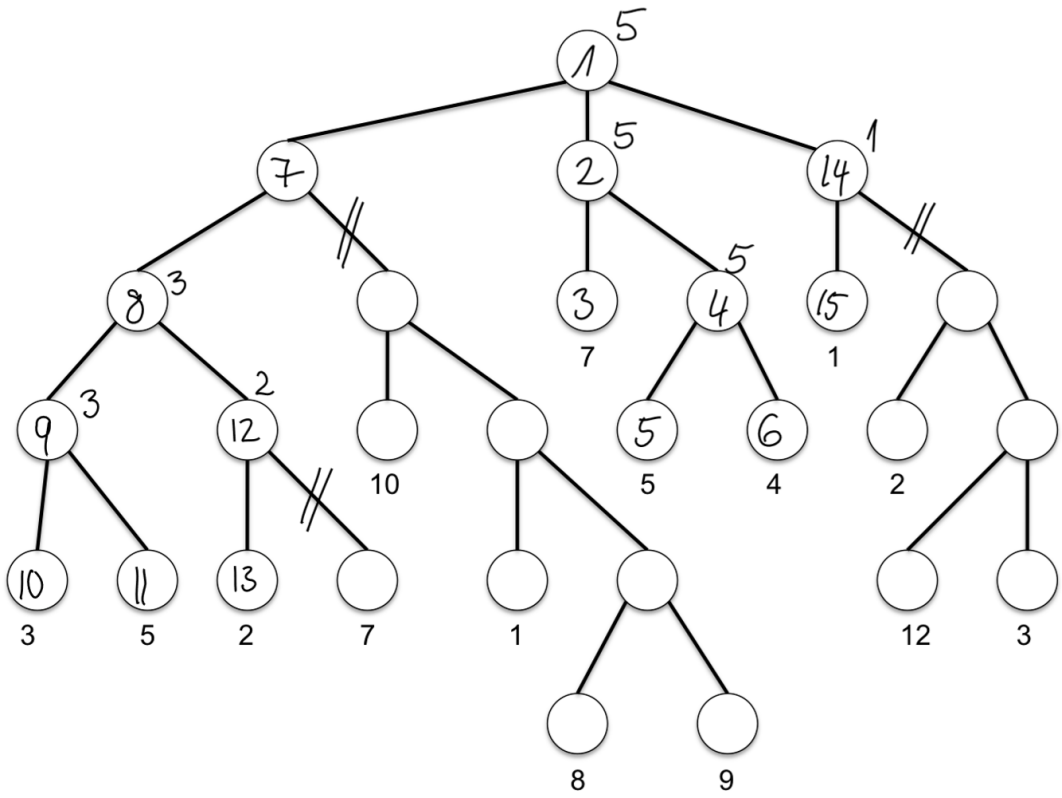
In den Knoten steht der Zeitpunkt der Generierung, neben den Knoten die (Entwicklung der) Minimax-Werte. Rot durchgestrichen sind durch Cutoffs nicht generierte Zweige.

Die rationale Zugfolge lautet (A, C, H, O). MAX kann einen Gewinn von 5 erwarten.

a) und b):



c) Die Reihenfolge der untersuchten Knoten steht als Zahl in den Knoten.



Aufgabe 5 – Harvester-Spiel

a) bis c)

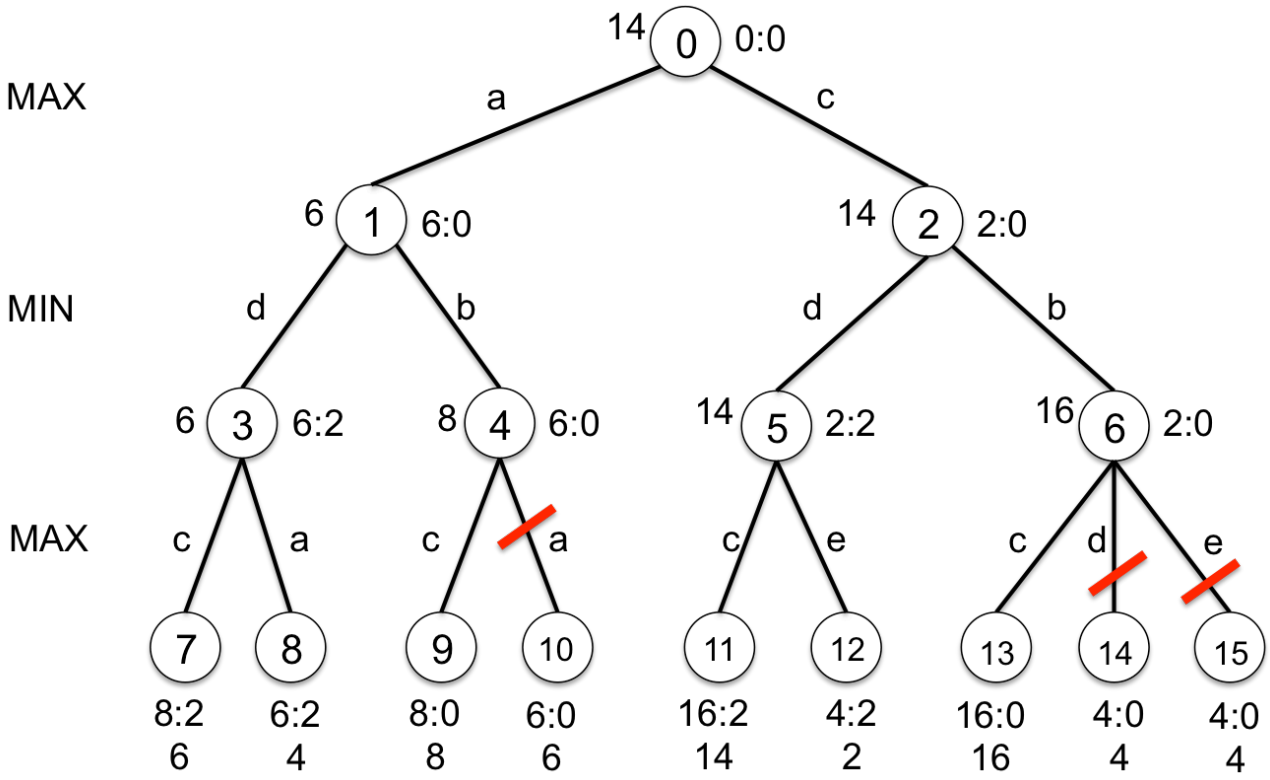
Rechs neben einem Knoten ist der aktuelle Zählstand notiert, in der Form a:b wobei a der Punktestand von Max und b der Punktestand von Min ist.. An den Blattknoten ergibt sich die Bewertung als Differenz beider Werte.

Nachdem h(7)=6 bis zum Knoten 6 hochgereicht wurde und danach beim Tiefenabstieg und Wiederaufstieg im Knoten 8 der Wert 8 eingetragen wird, ist zum ersten Mal die erste Cutoff-Bedingung erfüllt: Mit dem aktuellen Wissen (ohne Knoten 10 generiert zu haben) weiß MIN bereits, dass er im Zustand 6 die Aktion d bevorzugen würde.

Das nächste erreichte Blatt ist 11 mit einem Wert h(11)=14. Weil auch 12 keine Verbesserung bringt, wird dieser Wert an die Knoten 5 und 2 beim rekursiven Aufstieg eingetragen. Hier (im Knoten 2) findet kein Cutoff statt, weil die Wurzel aktuell den Spielwert des linken Teilbaums enthält (h(0=6)) und aus Sicht des Startspielers aktuell nicht die beste Wahl wäre. Jedoch kann MIN ja noch Verbesserungen (im Vergleich zum aktuellen Wert h(2)=14) erreichen, sodass der Baum weiter aufgespannt werden muss.

Das nächste Blatt 13 hat einen Wert 16, der beim Aufstieg im Knoten 6 vermerkt wird. Erneut ist die Cutoff-Bedingung erfüllt und die anderen Nachfolger von 6 werden nicht generiert.

Eine Ebene höher findet keine Änderung statt, denn MIN wählt das Minimum seiner bewerteten Nachfolgeknoten. Im letzten Schritt aktualisiert dieser Wert 14 die bisherige Bewertung der Wurzel.



d)

Das Spiel wird von demjenigen Spieler gewonnen, der das Feld e erreicht und dort

verweilt. Diese Aktionssequenz kann bei bestem Spiel beider Spieler nur von MAX erreicht werden, indem er zunächst nach c und danach nach e läuft. Diese Aktionssequenz c*e (wobei * ein beliebiger Zug von MIN ist) wird im aktuellen Spielbaum jedoch (in den Knoten 12 und 15) jedoch deutlich schlechter bewertet als – global gesehen – suboptimale Zugsequenzen.

MAX spielt zwar den richtigen Zug c (aber mit der unzureichenden / falschen „Begründung“).

Um die optimale Zugsequenz c*e*e (e im 5. Halbzug abernten) zu finden müsste MAX den Spielbaum schon bis zum 5. Halbzug aufbauen.