HAW HAMBURG

HAUSARBEIT
Kjell May

Minesweeper als Constraint Satisfaction Problem

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Inhaltsverzeichnis

In dieser Arbeit wird ein Algorithmus vorgestellt, der versucht, auf Grundlage von Constraint Satisfaction Algorithmen, ein beliebiges Minesweeper Spiel zu lösen. Die dabei eingesetzten Algorithmen schließen AC-3, Revise und Backtracking ein. Es werden außerdem die Herausforderungen, die das Lösen von Minesweeper bietet, diskutiert und angegangen. Der daraus enstandene *Solver* wird getestet und zeigt eine abnehmende Effizienz bei steigender Komplexität der Spielfeldkonfiguration (bestehend aus Spalten, Zeilen und Anzahl Minen). Im Vergleich zu anderen Implementierungen und Ansätzen für Minesweeper schneidet er schlechter ab, ist jedoch auch recht grundlegend aufgebaut und hat die Möglichkeit weiter ausgebaut zu werden.

Keywords: Constraint Satisfaction, CSP, AC-3, Revise, Backtracking, Minesweeper

1 Einleitung

Minesweeper ist ein klassisches Rätsel-/ Puzzlespiel von Microsoft Windows, welches auf früheren Windows-Versionen vorinstalliert gewesen ist. In dem Spiel geht es darum, alle Felder auf einem Spielfeld aufzudecken, unter denen sich keine Mine befindet. Schafft man das, hat man gewonnen, ansonsten verliert man das Spiel. Um dieses Ziel zu erreichen und die Minen zu lokalisieren, gibt das Spiel einem Hinweise in Form von Zahlen auf aufgedeckten Feldern, welche die Anzahl der umgebenden Minen beschreiben. Ein Beispiel eines gelösten Spiels könnte so aussehen (entnommen von [1]):

Abbildung 1: Ein beendetes Spiel auf Windows XP

2 Algorithmus

Der gesamte Quellcode zu dem hier beschriebenen Algorithmus befindet sich unter https://github.com/KjellooMelloo/Minesweeper_CSP

2.1 Definition

Um ein Constraint Satisfaction Problem formal beschreiben zu können, müssen eine Menge von Variablen, die Menge ihrer Wertebereiche und die Definition der Constraints festgelegt werden. Für Minesweeper habe ich dies wie folgt definiert:
Für ein Spielfeld mit $n \in \mathbb{N}$ Spalten und $m \in \mathbb{N}$ Zeilen sind die Variablen
$X = \{(x, y) | 2 \leq x \leq n \ und \ 2 \leq y \leq m\}$ und ihre Wertebereiche $D = \{0, 1\}$, wobei 1 für eine Mine steht und 0 für ein sicheres Feld.

Die Constraints müssen umfangreicher definiert werden. Für jede Variable (x, y) gibt es den Constraint auf ihm und alle seiner Nachbarn, dass der Wert auf dem Feld der Variablen, desweiteren als Konstante $k(x, y)$ mit $k \in \mathbb{N}$ benannt, gleich der Summe aller Werte der Nachbarsvariablen ist. Formal definiert wäre dies
$C = \{C(x, y) | k(x, y) = \sum_{\substack{0 \leq a \leq n-1 \\ 0 < b < m-1}} (a, b) \text{ mit } (a, b) \in N_G((x, y))\}$.
Binäre Constraints zwischen zwei benachbarten Variablen ergeben sich dann dadurch, dass eine Variable keinen Wert annehmen kann, welcher den Constraint des Nachbars verletzen würde. Diese sind Teilschritte dahin, den eigentlichen Constraint erfüllen zu können. Der Constraint-Graph ergibt sich für dieses Spiel trivial als Graph des Spielfelds.

2.2 Besonderheiten Minesweeper

Eine der Besonderheiten, um das Spiel Minesweeper zu lösen zu können, ist das Aufdecken der Felder, womit die Konstante auf diesem Feld bekannt wird und damit dynamisch Constraints definiert werden können. Daraus ergibt sich, dass der Algorithmus immer wiederholt wird, wenn neue Informationen durch Aufdecken erlangt werden. Mit dieser Art der unvollständigen Information über das Spielfeld unterscheidet es sich deutlich von anderen CSPs wie Sudoku, das Vier-Farben-Problem oder das Damenproblem. Eine weitere Herausforderung ist, dass nicht jedes Spiel mit Sicherheit lösbar ist.

Es gibt Fälle, in denen nur mit 50% Wahrscheinlichkeit gesagt werden kann, wo sich eine Mine befindet. Beispiel (entnommen aus [2]):

Abbildung 2: Beide verdeckten Felder könnten hier eine Mine sein. Es gibt keine weitere Nachbarsfelder mit Hinweisen, also muss hier geraten werden

2.3 Ablauf

2.3.1 Startpunkt und Aufdecken

Die erste Herausforderung zum Lösen dieses Spiels ist die Wahl, welches Feld zuerst aufgedeckt wird, da alle Felder zu Beginn verdeckt sind. In der Windows-Version von Minesweeper ist sichergestellt, dass der erste Klick keine Mine aufdecken kann [1]. Dies habe ich für meine Implementierung des Spiels auch übernommen, aus einfachen Komfortgründen. Ist diese Voraussetzung gegeben, kann zu Beginn also einfach ein zufälliges Feld gewählt werden.

Jedes Mal, wenn ein Feld aufgedeckt wird und es keine Mine ist, kann der Wertebereich der Variable für das Feld auf $D = \{0\}$ reduziert, damit auch der Wert auf 0 gesetzt und die binären Constraints dieser Variable zu allen Nachbarn in beide Richtungen definiert werden. Als Heuristik habe ich hier zusätzlich eingebaut, dass falls die aufgedeckte Konstante eine 0 ist, direkt alle Nachbarn aufgedeckt werden können (rekursiv für weitere Nullen). Damit werden die trivialen Fälle der 0 direkt abgehandelt, wodurch die zugehörigen Variablen bereits konsistent sind.

Im Laufe des Algorithmus ergeben sich weitere sichere Felder, welche, wie oben beschrieben, wieder aufgedeckt werden können.

2.3.2 AC-3 und Revise

Der Kern der Algorithmen für CSPs stellt der mittlerweile weit verbreitete Algorithmus AC-3 mit Revise dar, wie er von Alan Mackworth in [3] beschrieben wird. Als Pseudocode ausgedrückt sind sie wie in den folgenden Abbildungen 3 und 4:

```
begin
  for i ← 1 until n do NC(i);
  Q ← {(i, j) | (i, j) ∈ arcs(G), i ≠ j}
  while Q not empty do
    begin
      select and delete any arc (k, m) from Q;
      if REVISE ((k, m)) then Q ← Q ∪ {(i, k) | (i, k) ∈ arcs(G), i ≠ k, i ≠ m}
    end
end
```

Abbildung 3: Pseudocode AC-3 Algorithmus

```
procedure REVISE((i, j)):
begin
  DELETE ← false
  for each x ∈ D_i do
    if there is no y ∈ D_j such that P_ij(x, y) then
      begin
        delete x from D_i;
        DELETE ← true
      end;
  return DELETE
end
```

Abbildung 4: Pseudocode Revise Algorithmus

Der AC-3 Algorithmus beginnt mit Node Consistency ($NC(i)$). Diese ist bereits durch das Aufdecken gegeben und es kann direkt mit den binären Constraints fortgeführt werden. In meiner Implementierung beinhaltet die Q die erzeugten binären Constraints. Für jede Kante wird dann *Revise* ausgeführt. Gibt es in *Revise* eine Änderung, kann der übrig gebliebene Wert direkt gesetzt werden, da der Wertebereich nur aus 0 und 1 besteht. Haben wir hier eine Mine gefunden, kann diese auch für das Spiel direkt (mit einer Flagge) markiert werden. Alle Nachbarn von k werden dann in Q hinzugefügt.
Desweiteren findet hier eine Überprüfung statt, ob die Variable (xk, yk) bereits konsistent ist (Referenz Konsistenz). Ist dies der Fall, sind alle benachbarten Felder mit dem Wert 0 sicher und können aufgedeckt werden. Ist Q leer, wird erst geschaut, ob das Spiel vorbei ist, nämlich genau dann wenn eine Mine oder alle Felder, die keine Mine sind, aufgedeckt wurden. In dem Fall sind wir bereits fertig und können beenden.

Ansonsten wird geschaut, ob sichere Felder zum Aufdecken hinzugefügt wurden und von vorne begonnen. Ist die Q leer, das Spiel aber noch nicht vorbei oder gelöst, geht es mit Backtracking weiter.

Der Revise-Algorithmus schaut sich die Kanten/ binären Constraints an und prüft auf mögliche Verletzung der Constraints. Es wird jeder Wert von i der Kante (i, j) bzw. $((xi, yi), (xj, yj))$ probeweise gesetzt und geschaut, ob dies dazu führt, dass j keinen Wert annehmen kann, der diesen Constraint erfüllen könnte. In dem Fall wird der getestete Wert für i gelöscht und $DELETE$ auf *false* gesetzt. Der Test auf Verletzung dieses Constraints wird wie folgt durchgeführt:

Für jeden Wert im Wertebereich von j wird geschaut, ob dies seinen Constraint zu allen Nachbarn verletzt. Dies kann der Fall sein, wenn für einen Nachbarn zu viele oder zu wenig Minen existieren würden, also die Summe der Werte aller Nachbarn größer als die Konstante sind oder die Anzahl der unbestimmten Nachbarn nicht mehr die erforderliche Anzahl an Minen erreichen könnten.
Für *Revise* muss beachtet werden, dass wir die Konstante von j bei (i, j) wissen müssen, das Feld also aufgedeckt sein muss. Ist das nicht der Fall, wird diese Kante einfach übersprungen.

Nach AC-3 mit Revise und vor Backtracking kommt im Algorithmus noch ein Zwischenschritt. Da immer wieder zu AC-3 zurückgekehrt wird, kann es sein, dass bereits alle Minen gefunden wurden. Tritt der Fall ein, können die restlichen Felder einfach aufgedeckt und der *Solver* vollständig konsistent gemacht werden. Dann ist der Algorithmus ebenfalls abgeschlossen. Ansonsten geht es mit Backtracking weiter.

2.3.3 Backtracking

Ist keine Lösung durch AC-3 gefunden worden, kommt Backtracking zum Einsatz, wie er auch in [3] beschrieben wird. Hierbei geht es darum, für alle nicht belegten Variablen alle möglichen Zuweisungen zu generieren, testen und gültige zu sammeln.
Schon bei kleinen Wertebereichen führt dies zu einer hohen Komplexität. Im unserem Fall mit einem Wertebereich der Größe 2, liegt die Komplexität bei $\mathcal{O}(2^n)$. Bei einem Minesweeper-Spielfeld für die Schwierigkeit *Beginner* (8x8 mit 10 Minen) nimmt die Komplexität häufig ungewollte Ausmaße an. Um dem entgegenzuwirken, wurden folgende Maßnahmen getroffen:

1. Um nicht alle Möglichkeiten generieren zu müssen und dann zu testen, wird vorzeitig abgebrochen, falls eine Zuweisung nicht gültig sein kann. Dies tritt beispielsweise ein, wenn die Summe der Werte in der Zuweisung größer als die Anzahl der übrigen Minen ist. Umgesetzt wird dies rekursiv aus Effizienz- und Lesbarkeitsgründen.

2. Da viele Berechnungen durchgeführt werden, bei Tests auf Gültigkeit, wird *Memoization* eingesetzt, also eine Art Cache eingeführt, in dem bereits berechnete Lösungen gespeichert werden.

3. Um die Komplexität weiter zu reduzieren, werden außerdem Heuristiken festgelegt und genutzt. Da es regelmäßig vorkommt, dass ein Großteil der unbelegten Variablen keine aufgedeckten Nachbarn haben und damit wenig über die Gültigkeit einer Zuweisung dieser Variablen gesagt werden kann, werden diese exkludiert. In die Menge der Variablen, die für Backtracking angeschaut werden, kommen also nur solche, welche mind. einen Nachbar besitzen, der aufgedeckt ist. Desweiteren werden davon nur maximal 10 Variablen genommen, um die Komplexität auf $\mathcal{O}(2^{10})$ zu begrenzen.

Wie werden Zuweisungen nun auf Gültigkeit geprüft?

Zuerst werden nur die Zuweisungen genommen, wo die Summe der Werte zwischen 1 und Anzahl Minen liegt bzw. gleich Anz. Minen ist, wenn es sich um die letzen Felder/ Variablen handelt. Diese Unterscheidung beschreibt, dass die übrigen Minen nicht alle auf oder neben den gerade betrachteten Feldern liegen müssen, sondern noch weiter außerhalb sein können. Wenn es sich um die letzten Felder handelt, müssen sich die Minen offensichtlich unter denen befinden. Beispielsweise für den Fall, dass in der oberen linken Ecke eine 1 aufgedeckt wurde und Lösungen generiert werden sollen, sollten 3 mögliche Zuweisungen enstehen, nämlich dass genau ein Nachbar die Mine haben könnte.
Alle damit gültigen Zuweisungen werden dann weiter überprüft, indem alle Werte probeweise gesetzt werden und auf Verletzung der Constraints ihrer Variablen geprüft wird. Wird nur der Constraint *einer* Variablen verletzt, ist die gesamte Zuweisung ungültig und wird verworfen. Andernfalls ist sie gültig und kann zu der Menge der gültigen Lösungen hinzugefügt werden. An dieser Stelle werden dann auch Berechnungen in den Cache getan, zu Beginn der Lösungsüberprüfung abgefragt und regelmäßig aktualisiert (da sich das Spielfeld und damit die Umstände dynamisch ändern).

Wie wird mit den gültigen Zuweisungen umgegangen?

Je nachdem, wie viele Lösungen generiert worden sind, wird unterschiedlich weiter verfahren.

1. **Es wurde keine gültige Lösung gefunden.** Konnte keine gültige Lösung gefunden werden, können wir nur damit weitermachen, ein zufälliges Feld auszuwählen und aufzudecken und anschließend wieder zu AC-3 zu gehen.

2. **Es wurde genau eine gültige Lösung gefunden.** Konnte nur eine Lösung gefunden werden, dann kann direkt jedem der zugehörigen Variablen sein Wert zugewiesen werden. Hier zugewiesene Minen können markiert und sichere Felder aufgedeckt werden. Danach wird wieder AC-3 ausgeführt.

3. **Es wurden mehrere gültige Lösungen gefunden.** Konnten mehrere Lösungen gefunden werden, werden diese zuerst gemeinsam betrachtet. Sollte es sichere Felder in jeder Lösung geben, also solche, für welche die Variable immer 0 ist, können diese aufgedeckt und zu AC-3 zurückgekehrt werden. Gibt es jedoch kein sicheres Feld, wird die erste Lösung ausgewählt und alle dort als sicher markierten Felder aufgedeckt und auch zu AC-3 zurückgekehrt

2.4 Determinierung

Der Algorithmus kann auf mehrere Arten determinieren. Folgend die Möglichkeiten:

1. Der Algorithmus determiniert, weil das Spiel nach Minesweeper Regeln beendet ist. Dies ist entweder der Fall, wenn alle Felder aufgedeckt wurden, die keine Minen sind oder eine Mine aufgedeckt wurde. Der Solver könnte eine Mine aufdecken, wenn bei der Lösungsfindung in Backtracking keine Lösung gefunden und folgend ein zufälliges Feld aufgedeckt wurde.

2. Der Algorithmus determiniert, weil der Solver konsistent ist. In dem Fall müssen vorher nicht alle Felder aufgedeckt sein, es wird aber der Vollständigkeits halber gemacht. Der Solver ist konsistent genau dann, wenn alle Variablen konsistent sind, jede Variable einen Wert hat bzw. Wertebereich der Größe 1 und wenn die Summe der Variablen mit dem Wert 1 gleich der Anzahl der Minen ist. Variablen sind an sich konsistent, wenn sie als Mine markiert sind oder wenn die Summe der Werte der Variablen der Nachbarn gleich der Konstante k dieser Variablen sind.

3. Das Spiel ist nicht vorbei und der Solver ist nicht konsistent, aber alle Minen sind markiert und es existieren noch verdeckte Felder. Dann werden diese einfach aufgedeckt, bekommen ihre Werte zugewiesen und der Solver ist anschließend konsistent. Dies spart eine womöglich aufwändige Lösungsgenerierung durch Backtracking, weil es ein trivialer Fall ist.

3 Experimente und Ergebnisse

Um die Qualität des Solvers bestimmen zu können, wurde er mit unterschiedlichen Spielfeldgrößen und Minenanzahl getestet. Eine Beispielausgabe der Konsole für ein Spiel der Schwierigkeit "Beginner"(8x8 mit 10 Minen):

```
Mines left: 10, Cells left: 64
Starting solve with AC-3 and revise
Mines left: 10, Cells left: 64
Generating solutions
Solutions found: 129
Looking for a safe cell in any solution
Found safe cell: 5 5
Mines left: 3, Cells left: 31
Starting solve with AC-3 and revise
Mines left: 3, Cells left: 30
Generating solutions
Solutions found: 129
Looking for a safe cell in any solution
Found safe cell: 6 0
Mines left: 3, Cells left: 30
Starting solve with AC-3 and revise
Mines left: 3, Cells left: 25
Generating solutions
Solutions found: 175
Looking for a safe cell in any solution
No safe cell found.
Taking corner
Mines left: 3, Cells left: 25
Starting solve with AC-3 and revise
Mines left: 0, Cells left: 8
No more mines left, can uncover rest of cells
Won
```

Zur Übersicht sind hier Ergebnisse verschiedenster Feldkonfigurationen und ihre Ergebnisse bei 50 Durchläufen mit durchschnittlicher Laufzeit und wie weit sie in gescheiterten Versuchen gekommen sind:

Höhe x Breite	nMinen	Erfolgsrate	ØZeit	ØFortschritt
5x5	3	80%	0.0132s	91.12%
8x8	10	12%	0.1019s	31.04%
12x12	25	2%	0.2381s	18.21%
16x16	40	0%	0.2495s	13.1%

Wie sich erkennen lässt, ist der Solver bereits ab der Spielfeldkonfiguration für Beginner nicht sehr erfolgreich. Das liegt daran, dass recht häufig ein Punkt erreicht wird, an dem viele Lösungen möglich sind, kein sicheres Feld sich daraus ergibt und dann ein zufälliges Feld genommen werden muss. Für einfachere Spiele, kann er jedoch das Spiel recht verlässlich lösen.

4 Fazit und Aussicht

In dieser Hausarbeit wurde ein Algorithmus entwickelt und vorgestellt, der versucht Minesweeper als Constraint Satisfaction Problem lösen zu können. Es hat sich gezeigt, dass diese Version schnell an Grenzen stößt und nur sehr einfache Konfigurationen lösen kann. Der Solver ließe sich an einigen Stellen sicherlich noch erweitern und verbessern. Man könnte beispielsweise bei Auswahl des nächsten Feldes als Heuristik Wahrscheinlichkeiten, dass ein Feld eine Mine ist, nehmen, um möglicherweise mehr Fortschritt erlangen und ein vorzeitiges Ende verzögern zu können. Ein paar Beispiel für andere Ansätze und Untersuchungen:
David Becerra hat in seiner Bachelorthesis [2] intensiver verschiedene Ansätze ausprobiert und ist zu deutlich besseren Ergebnissen gekommen. Der Benutzer *DavidNHill* hat einen Solver geschrieben (siehe [4]), der bei jedem Klick die Wahrscheinlichkeit für ein sicheres Feld berechnet. Robert Massaioli hat einen Algorithmus geschrieben, der Minesweeper mit einem Matrizen-Ansatz löst (siehe [5]). Aus einer Kombination der verschiedenen Ansätze könnte ein womöglich effizienterer Algorithmus entwickelt werden. Wie jedoch in [6] und [7] von Richard Kaye gezeigt wurde, ist Minesweeper NP-complete und damit ein effizienter Algorithmus zum Lösen dieses Spiel zu finden, sehr herausfordernd.

Literatur

[1] Minesweeper website. https://minesweepergame.com/. Abgerufen: 2023-02-23.

[2] David J. Becerra. Algorithmic approaches to playing minesweeper. Bachelor's thesis, Harvard College, 2015. URL http://nrs.harvard.edu/urn-3:HUL.InstRepos:14398552.

[3] A. Mackworth. Consistency in network relations. *Artificial Intelligence*, 8:99–118, 1977. URL https://doi.org/10.1016/0004-3702(77)90007-8.

[4] Jsminesweeper. https://github.com/DavidNHill/JSMinesweeper. Abgerufen: 2023-02-23.

[5] Solving minesweeper with matrices. https://massaioli.wordpress.com/2013/01/12/solving-minesweeper-with-matricies/. Abgerufen: 2023-02-23.

[6] R. Kaye. Minesweeper is np-complete. *The Mathematical Intelligencer*, 22:9–15, 2000. URL https://link.springer.com/article/10.1007/BF03025367.

[7] R. Kaye. Some minesweeper configurations. Technical report, School of Mathematics The University of Birmingham, 2007. URL https://web.mat.bham.ac.uk/R.W.Kaye/minesw/minesw.pdf.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original