AC3 Algorithmus für Dummies

Manuel Leppert
Insitute of Computer Science
Freie Universität Berlin
manuel.leppert@fu-berlin.de

Lisa Dimitrov
Insitute of Computer Science
Freie Universität Berlin
lisa.dimitrov@fu-berlin.de

Marvin Hagemeister
Insitute of Computer Science
Freie Universität Berlin
marvin.hagemeister@fu-berlin.de

Course on Data Visualization, Winter Term 2017/18.

Introduction

Der AC3 Algorithmus ist ein Algorithmus der zur Lösung von Constraint-Erfüllungsproblemen(auch Constraint Satisfaction Problem oder kurz CSP) eingesetzt werden kann um die Eigenschaft der Kantenkonsistenz herzustellen. Er wurde 1977 von Alan Mackworth [6] entwickelt und wird im Bereich der symbolischen KI eingesetzt. In [7] wird argumentiert, dass die deklarative Art wie mit CSPs programmiert werden kann dem "heiligen Gral" der Informatik am nächsten kommt. Der Kern der Idee ist dabei, dass der Nutzer ein Problem formuliert und der Computer dieses selbstständig löst, ohne dass der Nutzer spefizieren muss auf welche Art und Weise dies geschehen muss. Diese Art der Programmierung findet Anwendung in verschiedensten Gebieten wie Beispielsweise der künstlichen Intelligenz, in Datenbanken, Programmiersprachen, Computer Vision,...

Unsere Zielgruppe

Mit diesem Paper wollen wir eine Einführung in unser Visualisierungsprojekt für angehende Informatikstudenten, sowie alle an dem Thema interessierten Personen bieten. Wichtig bei der Umsetzung ist, dass wir idealerweise neben angehenden Informatiker*Innen auch anderweitig interessierte von der Thematik begeistern können.

Motivation

Obwohl der AC3 Algorithmus eine wichtige Grundlage zur Lösung von Erfüllungsproblemen in der Informatik darstellt, ist es schwierig geeignete Visualisierungen zu finden. Gerade für Einsteiger in der Thematik würde sich eine interaktive Visualisierung anbieten. Durch die Alltäglichkeit dieses Problems und die vergleichsweise simple Umsetzung, ist es auch möglich, einen tieferen Einblick in Graphen und Contraint Probleme -selbst für in der Informatik komplett unerfahrene Personen-, zu bieten. Widererwartend zu den von uns aufgezählten Eigenschaften des AC3-Algorithmus, existieren keinerlei aufwendigere oder interaktive Visualisierungen, die sich auf konkrete Probleme der Informatik beziehen oder die Funktionsweise im informatischem Kontext verdeutlichen. Da es sich um einen iterativen Algorithmus handelt, bietet sich eine schrittweise (animierte) Visualisierung an.

Constraint-Netze und AC3

```
1  // Reduziert Wertebereiche
2  function AC3
3      queue = Alle gerichteten Kanten des CSP
4      while (!empty(queue))
5          Entferne eine Kante (x, y) aus queue;
6          if(EntferneInkonsistenteWerte(x, y))
7              foreach (Nachbar z von x)
8                  queue.ADD(Kante(z, x))
9  function AC3 end
10
11 // Liefert true, wenn Wertebereich D(x) reduziert
       wird
12  function EntferneInkonsistenteWerte(x, y)
13     removed = false
14     foreach (Value v1 in D(x))
15         if(Kein v2 in D(Y), so dass (x=v1, y=v2)
               alle Constraints zwischen (x,y) erfllt)
16             D(x).LSCHE(v1)
17             removed = true
18     return removed
19 function EntferneInkonsistenteWerte end
```

Constraint Programming ist ein Programmier-Paradigma, das auf der Idee beruht Problemstellungen als ein Netz aus Variablen V und Constraints C zu modellieren. Jeder Variablen x_i ist dabei ein Wertebereich D_i zugeordnet. Constraints sind Teilmengen des Kreuzprodukts $C \subseteq D_1 \times D_2 \times D_3 ...$ der Wertebereiche, sodass diese Teilmengen das durch den Constraint definierte Prädikat erfüllt. Der Wertebereich der Variablen kann endlich, aber auch unendlich sein. Falls sich ein Constraint nur auf eine Variable bezieht, wird dieser Constraint unär genannt, ist er zweistellig wird er als ein binärer Constraint bezeichnet. Constraints dürfen sich auf beliebig viele Variablen beziehen, wobei die meisten Algorithmen binärisierte Constraint-Netze erwarten. Jeder n-stellige Constraint lässt sich jedoch algorithmisch in äquivalente binäre Constraints überführen, sodass nach einer Vorverarbeitung eines Constraint-Netzes garantiert werden kann, dass dieses ausschließlich aus binären Constraints besteht. Existiert für einen Constraint ein Tupel mit Werten x, sodass $(x_1, x_2, x_3, ...) \in D_1 \times D_2 \times D_3 \times ....$ gilt dieser Constraint als erfüllt. Sind alle Constraints $c_1, c_2, c_3, ... \in C$ erfüllbar existiert eine Lösung für das

Constraint-Netz.
Häufig kommen Beispiele zur Veranschaulichung von Constraint-Netz-Modellierungen aus dem Bereich der Logik-Rätsel, wie dem folgenden crypto-arithmetischen Rätsel. Gegeben ist folgender Text:

SEND
+MORE
=MONEY

Die Aufgabe ist es den numerischen Wert für **MONEY** zu ermitteln. Jedem Buchstaben des Rätsels muss dafür eine Zahl zwischen 0 und 9 zugeordnet werden mit der Einschränkung, dass die Addition mit den zugeordneten Werten korrekt ist. Zusätzlich darf jede Zahl nur für einen Buchstaben gleichen Typs verwendet werden .
Beispielsweis darf die 9 nicht gleichzeitig E und N zugeordnet werden und falls $E = 9$ gesetzt wird gilt dies für alle Vorkommen von E .
Eine mögliche Modelierung hierfür könnte so gestaltet werden, dass für jeden Buchstaben des Rätsels eine Variable mit dem Wertebereich $0, 1, 2, ..., 9$ angelegt wird. Desweiteren sind Übertragsvariablen mit einem Wertebereich $0, 1$ notwendig um die spaltenweise Addition korrekt abzubilden. Um das Rätsel zu lösen sind zwei Typen von Constraints notwendig, nämlich einem Constraint, der die Wertebereiche der Buchstaben-Variablen so einschränkt, dass $a \neq b$ falls a und b verschieden sind, dh. ein Constraint der sicherstellt, dass die Bedingung erfüllt ist nach der zwei verschiedene Buchstaben unterschiedliche numerische Werte erhalten müssen. Zusätzlich ist eine Modellierung der arithmetischen Additions-Operation notwendig, die den Übertrag zwischen zwei Spalten berücksichtigt, sodass die Zuordnung auch arithmetisch sinnvoll ist.
Die Modellierung der ersten Spalte $D + E = Y$ (SEN**D**+MOR**E**=MONE**Y**) könnte bspw. aus einem Constraint bestehen, der die Verschiedenheit der Werte ausdrückt, zb. $D \neq E \neq Y$ und einem Constraint der den arithmetischen Aspekt fasst
$D + E = 10 * \ddot{U}BERTRAG + Y$.

AC3 - Motivation und Funktionsweise
AC3 ist ein Algorithmus der als Eingabe ein binäres Constraint-Netz erhält und in diesem Kantenkonsistenz herbeiführt. Ein binärer Constraint c_i der die Variablen X und Y verbindet gilt dann als konsistent, wenn sich für für jedes $x \in X$ ein $y \in Y$ findet lässt, sodass (x, y) c_1 wahr werden lässt. Anders ausgedrückt, jeder Wert x im Wertebereich von X muss in Y einen Wert haben, sodass der dazwischen liegende Constraint wahr wird.
Ein kantenkonsistentes Constraint-Netz ermögtlicht eine effizientere Lösungssuche als ein Durchsuchen des gesamten Problemraums, da verhindert werden kann, dass Pfade durchsucht werden, die inkonsistente Wertebereich enthalten, womit auch Backtracking vermieden werden kann.
AC3 ist ein Algorithmus der systematisch über alle Constraints iteriert und die Wertebereiche so anpasst, dass kantenkonsistenz hergestellt wird. AC3 ist in dieser Hinsicht ein fortgeschrittener Algorithmus, da durch ein intelligentes Hinzufügen von zu überprüfenden Kanten die Verarbeitungszeit reduziert werden kann. Wie in Abbildung gesehen werden kann, werden zuerst alle Kanten des Netzes in die Queue geschoben. Falls ein Wert aus einem Wertebereich von X verschwindet, werden -wie sich in den Zeilen 6 - 8 erkennen lässt- im Folgenden nur noch die Wertebereiche überprüft, die drauf beruhen, dass ein Wert in X diese stützt.

Ansätze zur Visualisierung

Whitebox vs Blackbox
Um AC3 zu erklären haben wir uns für eine Whitebox-Darstellung entschieden. In einer Whitebox-Darstellung geht es darum die Arbeitsweise eines Konzeptes im Detail darzustellen indem anhand von Pseudocode oder Implementierungen die verschiedenen Schritte betrachtet und explizit dargestellt werden.

Bei AC3 handelt es sich um einen Algorithmus, der sich grundsätzlich auch für eine Blackbox-Darstellung eignet, da sich durch diese erkennen lässt, worin der intendierte Nutzen steckt. Die konzeptionelle Mehrleistung die AC3 im Vergleich zu einer naiven Implementierung erbringt, steckt jedoch im Detail, nämlich dem Aufrufverhalten. Eine naive Implementierung zur Kantenkonsistenz würde in einer Black-Box-Visualisierung von der Black-Box-Visualisierung von AC3 nicht unterscheidbar sein. Um diese Abgrenzung zu schaffen muss die Arbeitsweise als White-Box verstanden werden.

Morphologische Analyse / Design Space
Da AC3 aus einem eher theorielastigen Bereich der Informatik kommt, der primär über Lehrveranstaltungen an Hochschulen verbreitet wird, fehlt es bisherigen Visualisierungen aufgrund des Mediums (Folien) an direkter Interaktivität. In unserer Visualisierung soll eine Interaktivitäts-Ebene integriert werden, die es erlaubt, ähnlich wie in einem Debugger Schritte vor und zurück zu tun. Ideal wäre außerdem wenn es zusätzlich zu "Schritt vor"/"Schritt zurück"-Aktionen möglich wäre Schleifenausführungen zu überspringen, da der Algorithmus aus einer Vielzahl von paarweisen Vergleichen besteht, die besonders bei größeren Wertebereichen langatmig und repetetive werden können. Möglicherweise könnte es sinnvoll auch ein Zurückspringen zu den letzten n durchgeführten Aktionen zu erlauben um analog des "Eyes beat Memory"-Prinzips den Nutzer dabei zu unterstützen Veränderung visuell zu erkennen um daraus den zugrundeliegenden Mechanismus abzuleiten. Klickbare Elemente im Sinne eines Objekt-Inspektors der das "Hineinsehen" in Objektzustände kann Nutzer dabei unterstützen das Verhalten des Algorithmus auf höheren Ebenen zu verstehen. Alternativ ist es möglich den Wertebereich einer Variable (bzw. die Mächtigkeit des Wertebereichs) kompakt in einem Density-Layout, Balken oder einer Farbkodierung darzustellen. Weiter ist vorstellbar Wertebereiche anpassbar zu gestalten, wobei die Einschränkung gelten muss, dass dies nur initial der Fall sein darf, da der Algorithmus sonst mglw. nicht korrekt arbeiten kann. Neben Interaktivitäts-Aspekten kann über ein Visualisierungkonzept auch in Kategorien von Enthüllungsgrad (Disclosure) und Erklärweise nachgedacht werden. Aspekte des Enthüllungsgrades wurden indirekt angesprochen mit der Möglichkeit zum Überspringen von Schleifen-Ausführungen, da diese verschiedene Detail-Ebenen des Algorithmuses repräsentieren. Um AC3 zu verstehen ist eine Transparenz auf allen Ebenen ideal, da der Algorithmus vergleichsweise geradlinig funktioniert um irritierende Komplexität nicht verstecken zu müssen. Da selbst atomare Operationen gut nachvollziehbar sind, sollte ein 100%-White-Box-Ansatz daher zielführend sein.

Hinsichtlich der Erklärweise setzen andere Visualisierungen auf unterschiedliche Grade an Formalisierung und Umfang an ergänzendem Text, auch die Intensität der Fachsprache im Vokabular unterscheidet sich. Aufgrund des gegebenen Kontextes der Visualisierung bietet es sich an die drei genannten Aspekte möglichst zu minimieren und den Inhalt möglichst direkt in die Visualisierung zu integrieren.

- AC3 arbeitet mit speziellen Graphen (Constraint-Netzen)
- Nicht zu komplex und relativ einfach zu verstehen
- Die Modellierung durch einen Graphen wird zusätzlich visualisiert
- Sämtliche Aspekte des Algorithmus sollen dargestellt und verständlich gemacht werden

Live-Beispiel und Pseudocode

Figure 1: Live-Beispiel mit vorgehobenem Pseudocode und Graphen

Ein Ansatz für die Visualisierung des Whitebox-Treatments ist eine Gegenübertstellung von Pseudocode und eines Beispielgraphen.

Dabei befindet sich auf der linken Seite der Pseudocode, welcher schrittweise farbig hervorgehoben wird und die korrespondierenden Kanten, bzw. Knoten auf der rechten Seite aktualisiert werden.

Vergleich AC1 und AC3

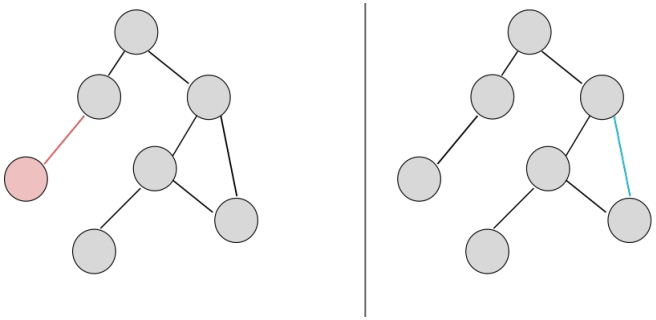

Figure 2: Graphischer Performancevergleich zweier Algorithmen

Um die Effizienz des AC3 Algorithmus in Relation zu setzen, kann er mit seinem Vorgänger, dem AC1 Algorithmus, verglichen werden.
Da es sich bei dem AC1 Algorithmus um einen naiven Ansatz handelt, stellt man bei der Ausführung eine klare Überlegenheit des AC3 Algorithmus fest.

Um dies zu visualisieren, können wir zwei identische Graphen gegenüber stellen, auf denen jeweils der AC1, oder der AC3 Algorithmus operiert. Beide Algorithmen werden zeitgleich, mit der gleichen Geschwindigkeit pro Schritt ausgeführt.

SMART-Zieldefinition

Gesamtziele:

1. Constraint-Netze so erklären, dass ein 1. Semester Informatikstudierender die wesentlichen Komponenten (bestehnt aus Variablen, zugeordneten Wertebereichen und einschränkenden Constraints) nennen und die Art und Weise des Zusammenhängens zwischen ihnen erleutern kann.

Die Zielerreichung kann in einem Dialog überprüft werden, sobald die Kerninhalte der Webseite implementiert sind.

2. Ein illustrative Beispiele für Constraint-Netze geben, die einerseits die Komplexität des Problems zeigen (es sollte nicht trivial lösbar sein in dem Sinne, dass eine Lösung durch 5-10 sekündiges nachdenken offensichtlich wird), andererseits für eine intuitiv verständliche Modellierung zugänglich sind (was eine Variable ist, was der Wertebereich ist und wie die Constraints modelliert sein müssen, sollte intuitiv nachvollziebar sein). Das Zielerreichung ist schwierig zu überprüfen. Am ehesten ist wiederum eine Dialog nach der ersten Implementierung mit potentiellen Lesern geeignet um die Zielerfüllung zu testen.

3. AC3 in einer Pseudocode-Variante präsentieren die ohne größere Programmierkenntnisse und formale Vorbildung verständlich ist indem einerseits geeignete Formulierungen gewählt werden, andererseits die Kerngedanken abstrakt genug deutlich gemacht werden um diese am Pseudocode nachvollziehen zu können. Eine Überprüfung ist ebenfalls in einem Dialog mit direkten Fragen wie "gibt es Unklarheiten bei der Notation/Syntax?", "gibt es Unklarheiten bei dem Verhalten der Schleifen oder von Funktionsaufrufen?" möglich.

4. Die Visualisierung soll den Zusammenhang von Pseudocode und realen Veränderungen von Wertebereichen zeigen, dh. für jeden Schritt muss klar sein (insofern dies sinnvoll ist), wo sich der Algorithmus in der konkreten Ausführung befindet und was gerade betrachtet wird. Eine Zielüberprüfung ist möglich indem sobald die Implementierung lauffähig ist, diese schrittweise durchgeführt und die Einhaltung überprüft wird.

5. Die Visualisierung sollte in großen (pro Kante) und kleinen Schritten (pro Wert $\in$ Wertebereich) interaktiv durchführbar sein. Die Zielüberprüfung hierzu gilt analog zu vorherigem Punkt.

6. Es sollen mindestens zwei Visualisierungen genauer untersucht und sinnvolle Variationen überlegt werden. Die Berwertung erfolgt innerhalb der Gruppe indem Argumente gesammelt und abgewogen werden sollen. Falls sich zu einem späteren Zeitpunkt, speziell während der Implementierung sinnvolle Varianten ergeben, sollen diese ebenfalls im Sinne des Design-Spaces offen gehalten werden.

References

Visualisierungsziele:

[1] Universität Kassel, FG Wissensverarbeitung: Kapitel 3: Constraints (WS 2008/09)
https://www.kde.cs.uni-kassel.de/wp-content/uploads/lehre/ws2008-09/KI/folien/4_03_Constraints.pdf

[2] Donald Bren School of Information and Computer Sciences, Chapter 3: Consistency Algorithms (2003)
https://www.ics.uci.edu/~dechter/books/chapter03.pdf

[3] Guido Tack, Constraint Propagation - Models, Techniques, Implementation (2009)
http://www.ps.uni-sb.de/Papers/abstracts/tackDiss.pdf

[4] Yuanlin Zhang, Roland H.C. Yap, CoMaking AC-3 an Optimal Algorithm(2001)
https://pdfs.semanticscholar.org/ce50/c8888d1f1a03a8619a3c0c5d5cc655835ee5.pdf

[5] Bernhard Nebel,Julien Hue und Stefan Wölfl,Constraint Satisfaction Problems - Enforcing Consistency (2012)
http://gki.informatik.uni-freiburg.de/teaching/ss12/csp/csp04-handout.pdf

[6] Richard J. Wallace, Why AC-3 is almost always bettern then AC-4 for establishing Arc Consistency in CSPs (1993)
https://www.ijcai.org/Proceedings/93-1/Papers/034.pdf

[7] Eugene C. Freuder, In Pursuit of the Holy Grail (1997)
https://link.springer.com/content/pdf/10.1023%2FA%3A1009749006768.pdf