Animation Aussagenlogischer Beweise

markus.iser@kit.edu tobias.heuer@kit.edu

I. Einführung

Visuelle Darstellungen von Algorithmen können unser Verständnis von algorithmischen Konzepten und der Natur der zugrunde liegenden Probleme verbessern. In diesem Projekt geht es um Visualisierungen von Läufen des Algorithmus Conflict-Driven Clause Learning (CDCL), welcher aktuell der erfolgreichste Algorithmus zur Lösung des aussagenlogischen Erfüllbarkeitsproblems (SAT Problem) ist. Trotz der Komplexität des SAT Problems werden Implementierungen von CDCL in so genannten SAT Solvern erfolgreich in der industriellen Praxis eingesetzt, z.B. bei der Softwareverifikation, der Produktkonfiguration oder der automatisierten Planung.

II. Theoretischer Hintergrund

CDCL ist ein hybrider Algorithmus, der *Suche* (vgl. DPLL Algorithmus) und *Resolution* (vgl. Sättigungsalgorithmus) miteinander vereint. Für eine erfüllbare Instanz gibt ein SAT Solver eine Belegung (=Zertifikat für die Erfüllbarkeit) aus, welche in polynomieller Zeit überprüft werden kann (daher SAT $\in$ NP). Moderne SAT Solver geben aber auch Zertifikate aus, wenn eine gegebene Instanz nicht erfüllbar ist. Ein solches Zertifikat ist eine Folge voneinander abgeleiteter Klauseln, die mit der leeren Klausel endet. Das sind im schlimmsten Fall exponentiell viele, andernfalls gälte NP=co-NP. Gelernte Klauseln sind aber nicht nur für einen Beweis der Unerfüllbarkeit einer Instanz von Bedeutung, sondern sind auch ein wichtiges Mittel zur Begrenzung des Suchraums.

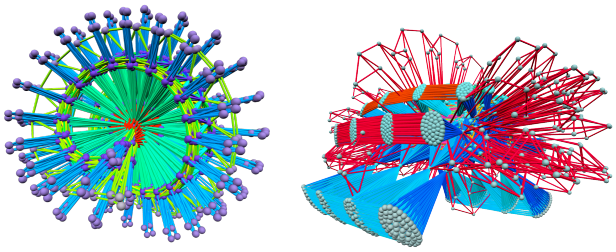

Fig. 1: Variable Interaction Graph zweier SAT Instanzen [3]

Welche gelernte Klauseln wichtig sind, kann man leider erst hinterher wissen. Daher wird das Klausellernen und -vergessen durch Heuristiken gesteuert. Um ein besseres Verständnis von solchen Heuristiken zu gewinnen, wollen wir in einer Visualisierung beobachten können, welche Klauseln gelernt werden und wann dies geschieht. In der Graphdarstellung eines SAT Problems, welches in konjunktiver Normalform (CNF) vorliegt, sind die Variablen die Knoten und die Klauseln die Hyperkanten. Zur Visualisierung von SAT Instanzen wird aber meistens die Cliqueneinbettung dieses Hypergraphen gewählt, welche im SAT Bereich als Variable Interaction Graph (VIG) bekannt ist (vgl. Abbildung 1).

III. Animation von CDCL

Wir wollen CDCL Läufe anhand gelernter Klauseln visualisieren, die in Form eines Klauselbeweises gegeben sind. Dieser Beweis kann als zweiter Parameter in Form eines Pfades zu einer Beweis Datei (DRAT Format [2]) angegeben werden. Es ist aber auch möglich gelernte Klauseln während der Algorithmus läuft mitzulesen (über die IPASIR Schnittstelle eines CDCL Solvers [1]).

Es gibt verschiedene Möglichkeiten, die Animation zu gestalten, daher soll Ihr Programm eine Menge an Konfigurationsmöglichkeiten bieten. Das geht vom einfachen farblichen Hervorheben der beteiligten Variablen, über eine konfigurierbare Fensterbreite für eine Art Hitzekarte, bis hin zum dynamischen Anpassen der Kantengewichte.

Die Softwarearchitektur nimmt hier zwei Gestalten an, wovon bei der ersten die Klauselproduzenten und -konsumenten in Beziehung gesetzt sind, um entsprechende Schnittstellen zu definieren. Bei der zweiten Gestalt, sind das Datenmodell (die Klauselmenge und die Konfiguration), die Graphvisualsierung und die fortlaufende Aktualisierung des Modells sowie Benutzerinteraktionen zur Änderung der Konfiguration in Beziehung gesetzt.

Da die Graphen recht groß werden können, wird es nicht möglich sein, einen gegebenes Layout nach jedem Update des Datenmodells vollständig neu zu berechnen (d.h., bis zum Equilibriumszustand laufen lassen). Daher sollten Sie eine Bibliothek auswählen, in der sie die Anzahl der Berechnungsschritte kontrollieren können. Sonst ist die Wahl der Technologie offen.

References

[1] Balyo, T., Biere, A., Iser, M., Sinz, C.: SAT Race 2015. Artif. Intell. **241**, 45–65 (2016)
[2] Heule, M.J.H.: The DRAT format and drat-trim checker. CoRR **abs/1610.06229** (2016), http://arxiv.org/abs/1610.06229
[3] Sinz, C.: Visualizing SAT instances and runs of the DPLL algorithm. J. Autom. Reason. **39**(2), 219–243 (2007)