Implementing a Strategy for Clause Selection with Alternating Path Theory for Automatic Theorem Proving In Python

Coursework

Bachelor of Science

at the course of studies Computer Science
at the Cooperative State University (DHBW) Stuttgart

von
Jannis Gehring

02/19/2025

Time of project:	09/30/2024 - 06/12/2025
Student number, course	6732014, TINF22B
Supervisor	Prof. Dr. Stephan Schulz

Selbstständigkeitserklärung des Studenten

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: *Implementing a Strategy for Clause Selection with Alternating Path Theory for Automatic Theorem Proving In Python* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt, falls beide Fassungen gefordert sind.

Ort, Datum

Abteilung, Unterschrift

.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

Inhaltsverzeichnis

Aus Gründen der besseren Lesbarkeit wird auf die gleichzeitige Verwendung der Sprachformen männlich, weiblich und divers (m/w/d) verzichtet. Sämtliche Personenbezeichnungen gelten gleichermaßen für alle Geschlechter.

Abkürzungsverzeichnis

DHBW	Duale Hochschule Baden-Württemberg
DEAP	Distributed Evolutionary Algorithms in Python
TPTP	Thousands of Problems for Theorem Provers

Quellcodeverzeichnis

Hinweis: Sofern bei einzelnen Listings keine Quellen angeführt werden, wurden sie vom Verfasser erstellt.

1 Einleitung

Dieses Kapitel leitet in die theoretischen Grundlagen und Hintergründe der Arbeit ein. Dabei wird sowohl auf das Themenfeld von automatischen Theorembeweisern als auch auf genetische Algorithmen eingegangen.

1.1 Automatische Theorembeweiser

Ein automatischer Theorembeweiser ist eine Software, welche anhand eines gegebenen Satzes von Axiomen nachweist, dass eine bestimmte Aussage zutrifft.

Speziell für die Prädikatenlogik erster Stufe gibt es eine Vielzahl an Beweisern [1], [2], [3]. Diese versuchen meist, die Aussage, dass aus den Axiomen die zu beweisende Aussage folgt, zu einem Widerspruch zu führen. Dazu nutzen sie Verfahren wie Resolution oder Superposition Calculus [4].

Diese Systeme unterstützen auch Prädikatenlogiken höherer Stufe oder spezielle Einschränkungen wie Prädikatenlogik mit Gleichheit.

1.1.1 Der Theorembeweiser E

Der Theorembeweiser E [5] ist ein von *Stephan Schulz* entwickelter Theorembeweiser für die vollständige Prädikatenlogik erster Stufe, unterstützt aber auch monomorphe höherstufige Prädikatenlogik mit Gleichheit. Er akzeptiert eine Problemspezifikation, die typischerweise aus einer Anzahl von Klauseln oder Formeln sowie einer Vermutung besteht.

Der Theorembeweiser liest das Problem ein und übersetzt es in konjunktive Normalform. In einer optionalen Vorverarbeitung wird das Problem nochmals vereinfacht. Danach wird ein Superposition-Calculus-Algorithmus [4] angewendet, der versucht, aus der Klauselmenge die leere Klausel abzuleiten. Gelingt dies, ist das übersetzte Problem unerfüllbar.

Wichtig ist hier vor allem, dass alle Beweisschritte in der Praxis recht komplex sind und durch eine sehr große Zahl an Konfigurationsparametern verschiedener Datentypen konfiguriert werden können. Eine passende Wahl dieser Konfigurationsparameter ist dabei entscheidend für den Erfolg des Beweisers. Aufgrund der Komplexität des Gesamtsystems ist es jedoch nicht mehr möglich, mit theoretischen Mitteln zu erschließen, was der optimale Satz an Konfigurationsparametern ist. Dies hängt möglicherweise auch von der Art des Problems ab.

1.1.2 Evaluation von Theorembeweisern: TPTP

Zum Vergleich von Theorembeweisern, aber auch zur Evaluation während der Entwicklung, ist ein Satz von Problemen hilfreich. Thousands of Problems for Theorem Provers (TPTP) [6] ist eine Sammlung solcher Probleme, welche häufig für das Benchmarking von Prover-Systemen herangezogen wird.

Das Projekt definiert dabei auch einen einheitlichen Standard [7] zur strukturierten Beschreibung solcher Probleme. Dieses Format wird von E als direktes Eingabeformat für Probleme akzeptiert [5].

1.2 Evolutionäre Algorithmen

Dieses Kapitel beschreibt die Grundlagen evolutionärer Algorithmen. Dieser Abschnitt geht insbesondere auf die Begrifflichkeiten und Methoden ein. Darüber hinaus wird auf die Implementierung und insbesondere die Programmbibliothek Distributed Evolutionary Algorithms in Python (DEAP) eingegangen.

1.2.1 Allgemeines

Evolutionäre Algorithmen (EA) sind eine Klasse von Optimierungsverfahren, die sich an den Prinzipien der natürlichen Evolution orientieren, wie sie in der Biologie vorkommen. Sie basieren auf Konzepten wie Selektion, Mutation, Rekombination (Crossover) und Überleben des Stärkeren. Diese Algorithmen werden häufig für komplexe Optimierungsprobleme eingesetzt, bei denen herkömmliche mathematische Ansätze schwer anzuwenden sind. [8]

1.2.2 Grundlegender Ablauf

Evolutionäre Algorithmen beginnen mit der zufälligen Initialisierung einer Population von Lösungen. In jeder Iteration werden die Individuen anhand einer Fitnessfunktion bewertet. Die besten Lösungen werden ausgewählt, kombiniert und mutiert, um neue Individuen zu erzeugen. Anschließend wird die Population aktualisiert, und der Prozess wiederholt sich, bis eine Abbruchbedingung erfüllt ist.

```python
population = initialize_population(size=100)
while not termination_condition():
    fitness = [evaluate(ind) for ind in population]
    # selection, mutation und crossover
    selected = select(population, fitness)
    offspring = crossover(selected)
    offspring = mutate(offspring)
    population = replace(population, offspring)
```

Listing 1: Pseudocode evolutionäre Algorithmen

1.3 Begriffe

Das Nachfolgende Dokument verwendet dabei insbesondere die folgenden Begriffe verwendet:

- **Initialisierung**: Eine Population von Individuen (Lösungen) wird zufällig erzeugt.
- **Fitnessbewertung**: Jedes Individuum wird anhand einer Fitnessfunktion bewertet, die angibt, wie gut es das Problem löst.
- **Selektion**: Individuen mit hoher Fitness werden bevorzugt ausgewählt, um Nachkommen zu erzeugen.
- **Rekombination** (Crossover): Zwei oder mehr Individuen kombinieren ihre Merkmale, um neue Individuen zu erzeugen.
- **Mutation**: Kleine zufällige Änderungen werden an den Nachkommen vorgenommen, um die genetische Vielfalt zu erhöhen.
- **Ersatz**: Die neue Population ersetzt die alte, wobei oft die besten Individuen erhalten bleiben.
- **Abbruchbedingung**: Der Algorithmus endet, wenn eine zufriedenstellende Lösung gefunden wurde oder eine vorgegebene Anzahl von Iterationen erreicht ist.

1.3.1 Arten von Evolutionäre Algorithmen

1.4 Ähnliche Arbeiten

2 Architektur

2.1 Fitnessfunktion

2.1.1 Auswahl der Evaluationsprobleme

2.1.2 Auswechseln der Evaluationsprobleme

2.2 Crossover- und Mutationsoperatoren

2.3 Der Optimierungsalgorithmus

3 Implementierung

3.1 Grundlegende Infrastruktur und Ausführungsumgebung

3.2 Zeitmessung

3.3 Umgang mit verschiedenen Datentypen der Suchparameter

3.4 Parameterbereiche für Suchparameter

3.5 Unittests und Validirung von Suchparametern

3.6 Herausforderungen bei der Implementierung

4 Auswertung

5 Fazit

5.1 Zusammenfassung der Ergebnisse

5.2 Ausblick auf zukünftige Arbeiten

5.3 Reflexion

Bibliographie

[1] A. Riazanov und A. Voronkov, „The design and implementation of VAMPIRE“, *AI Commun.*, Bd. 15, Nr. 2, 3, S. 91–110, Aug. 2002.

[2] S. Schäfer und S. Schulz, „Breeding Theorem Proving Heuristics with Genetic Algorithms“, in *GCAI 2015. Global Conference on Artificial Intelligence*, G. Gottlob, G. Sutcliffe, und A. Voronkov, Hrsg., in EPiC Series in Computing, vol. 36. EasyChair, 2015, S. 263–274. doi: 10.29007/gms9.

[3] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, und P. Wischnewski, „SPASS Version 3.5“, in *Automated Deduction – CADE-22*, R. A. Schmidt, Hrsg., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, S. 140–145.

[4] L. Bachmair und H. Ganzinger, „Rewrite-based equational theorem proving with selection and simplification“, *Journal of Logic and Computation*, Bd. 4, Nr. 3, S. 217–247, 1994.

[5] S. Schulz, S. Cruanes, und P. Vukmirović, „Faster, Higher, Stronger: E 2.3“, in *Automated Deduction – CADE 27*, P. Fontaine, Hrsg., Cham: Springer International Publishing, 2019, S. 495–507.

[6] G. Sutcliffe, „The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0“, *Journal of Automated Reasoning*, Bd. 59, Nr. 4, S. 483–502, 2017.

[7] G. Sutcliffe, „The Logic Languages of the TPTP World“, *Logic Journal of the IGPL*, 2022, doi: 10.1093/jigpal/jzac068.

[8] K. Weicker, „Natürliche Evolution“, in *Evolutionäre Algorithmen*, Wiesbaden: Springer Fachmedien Wiesbaden, 2015, S. 1–18. doi: 10.1007/978-3-658-09958-9_1.