Seminar

Software Engineering für verteilte Systeme

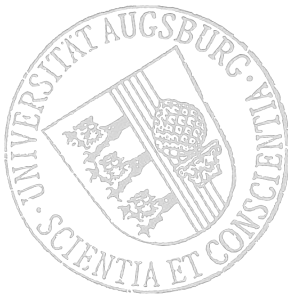
Sommersemester 2022

Challenges and Approaches of Solving Fuzzy Constraint Satisfaction Problems

[redacted]
Matrikelnummer: [redacted]
[redacted]

Betreuer: Tobias Foth
Softwaremethodik für verteilte Systeme (Prof. Bauer)
Universität Augsburg

Zusammenfassung In der folgenden Arbeit werden verschiedene Herangehensweisen und Herausforderungen zum Lösen von Fuzzy Constraint-Satisfaction-Problems behandelt. Da dieses Problem in vielen Bereichen der realen Welt Anwendung findet, ist die Forschung in diesem Feld sehr relevant. Es werden vier Verfahren vorgestellt, um das Finden von solchen Lösungen zu erleichtern: Branch-and-Bound, Forward-Checking, GENET und der Spread-Repair-Shrink-Algorithmus.



Inhaltsverzeichnis

1 Motivation

Aufgrund der Menge an Daten, die immer einfacher heutzutage gewonnen werden kann, steigt der Bedarf nach Techniken, die diese auswerten und benutzen, immer rasanter an. Die Forschung im Bereich der künstlichen Intelligenz ist von groSSer Bedeutung und wird in den verschiedensten Bereichen relevanter. In der realen Welt hat man allerdings nicht immer nur harte ganzzahlige Variablen und einfache Bedingungen, die erfüllt werden sollen.

Diese *einfacheren* Probleme wurden schon erfolgreich in der IT durch die verschiedensten Methoden gelöst, z.B. durch den SAT-Solver oder den Methoden der klassischen Constraint-Satisfaction-Problems (CSP). Der SAT-Solver (SAT Abk. für engl. satisfiability) behandelt die Erfüllbarkeit aussagenlogischer Formeln, in dem das Problem anhand von Klauseln und Literalen dargestellt werden kann. Als kurzes Beispiel dient $(\neg a \vee c) \wedge (b \vee c)$. Ein jedes CSP kann in ein SAT-Problem umgewandelt werden. [1] Ein Beispiel zu CSP erfolgt zu Beginn dieser Arbeit im Grundlagen-Abschnitt.

Wie wichtig diese Algorithmen in der realen Welt sind, erkennt man darin, wo sie Anwendung finden. Vor allem in der Planung und Suche sind sie von enormer Wichtigkeit. Zum Beispiel in der Planung von Transportaufgaben (eine Ware soll von Stadt A nach Stadt B mit einer bestimmten Ladung transportiert weden, hierbei muss entschieden werden welcher LKW (Gewicht der Ladung) von welchem Fahrer auf welcher Strecke bedient wird), ähnliches gilt in der Planung von Produktionsabläufen.

Doch nicht immer gibt es nur eindeutige boolsche Nebenbedingungen, diese können durch Präferenzen oder Prioritäten mit unterschiedlicher Gewichtung in unscharfe Nebenbedingungen abgewandelt werden. Hierbei handelt es sich um Fuzzy CSPs, welche im Folgenden genauer betrachtet werden sollen. Wenn ein System eine Entscheidung treffen muss und viele Bedingungen beachtet, ist auch die Laufzeit der Algorithmen entscheidend. Um dies möglichst effizient zu gestalten, werden vier verschieden Methoden vorgestellt, welche mögliche Herangehensweisen für das Lösen von FCSPs darstellen. Zum einen wird die systematische Suche, wie der Branch-and-Bound-Algorithmus [2] und auch das Forward-Checking [3], in Verbindung mit dem, im Grundlagen-Abschnitt erklärten, Backtracking [4] gebracht. Zum anderen wird die lokale Suche anhand des GENET-Algorithmus, ein neuronales Netzmodell [5], vorgestellt. Auch die Kombination aus beiden Suchen ist möglich und zeigt sich im Spread-Repair-Shrink-Algorithmus [6].

In dieser Arbeit liegt der Fokus auf den Herausforderungen und Herangehensweisen, wie Fuzzy CSPs gelöst werden können. Zuerst werden die Grundlagen zu CSP und dessen Erweiterung (FCSP) erläutert, ohne dass dies noch extra recherchiert werden muss. AnschlieSSend folgen die vier genannten Methoden zum Lösen von FCSPs. Im Anschluss wird die Arbeit kurz zusammengefasst und ein Fazit gezogen.

2 Grundlagen

Befasst man sich in der Informatik mit Algorithmen, die für die Suche und Planung nützlich sind, ist es wichtig, vorab einige Grundlagen zu erläutern, die für das Verständnis dieser Arbeit relevant sind. In diesem Abschnitt folgen Definitionen und Erklärungen, um mögliche Unklarheiten zu vermeiden.

2.1 Constraint-Satisfaction-Problem

Bei einem CSP (Constraint-Satisfaction-Problem) handelt es sich um ein Problem aus der künstlichen Inteligenz und aus dem Operations-Research-Bereich, bei dem bestimmte Bedingungen erfüllt werden sollen. Mathematisch gesehen ist ein CSP ein Triple $P =< X, D, C >$. Dieses Triple besteht aus einer Menge n von Variablen $X = \{x_1, \ldots, x_n\}$. Jede dieser Variablen x_i hat jeweils eine endliche Domain $D = \{D_1, \ldots, D_n\}$, der ein Wertebereich zugeteilt wird. Zudem sind t Constraints $C = \{C_1, \ldots, C_t\}$ gegeben, welche bestimmen, welche Kombinationen von Belegungen aus den Variablen erlaubt bzw. nicht erlaubt sind. Diese Bedingungen müssen alle simultan erfüllbar sein. Die Lösung dieses Problems ist ein n-Tuple, das Assignment $A = \{a_1, \ldots, a_n\}$, wobei $a_i \in D_i$ ist und jede Bedingung aus C erfüllt ist. Liegt keine vollständige Erfüllung der Bedingungen vor oder keine Domäne wurde zugewiesen, also $A = \emptyset$, so gilt das Problem als nicht gelöst. [7]

Ein bekanntes Beispiel hierfür ist das n-Damen-Problem. Dieses CSP wirft die Fragestellung auf, wie viele Möglichkeiten es gibt, n Damen auf einem $n \times n$-Schachfeld zu platzieren, ohne dass sich zwei Damen gegenseitig bedrohen. Zwei Damen dürfen sich somit nicht auf der selben Reihe, Linie oder Diagonalen befinden. [8]

Ein weiteres nennenswertes Beispiel, ist das Zeitplanungsproblem. Dieses tritt z.B. bei Universitäten auf, wenn Prüfungen von Studenten organisiert werden. Für die konkrete Planung werden die Anzahl an Studenten, Anzahl von Prüfungen, eine Menge von Räumen und die verfügbaren Zeiträume berücksichtigt. Hierbei ist allerdings zu beachten, dass jede Prüfung nur in einem Raum und zu einem bestimmten Zeitpunkt absolviert werden kann. AuSSerdem muss darauf geachtet werden, dass mehrere Prüfungen, die von einem Studenten angetreten werden, zeitlich voneinander abgegrenzt werden. Des Weiteren gibt es Restriktionen wie viele Studenten sich in einem Raum befinden dürfen. Ziel der Zeitplanung ist es, eine Lösung ohne mögliche Kollisionen in einem geringen zeitlichen Abstand zu finden.[9]

Um solche Probleme zu lösen, gibt es verschiedene Ansätze und Algorithmen. Die bekannteste und einfachste Herangehensweise ist das Backtracking. Im Backtracking-Algorithmus werden zuerst gültige Werte den Variablen zugewiesen. Dies wird solange wiederholt bis alle Variablen einen Wert haben. Sobald noch nicht jede Variable einen Wert hat oder ihr kein Wert mehr zugewiesen werden kann, ist die Lösung nicht vollständig. Deshalb geht man der Reihe nach rekursiv immer den letzten Schritt zurück, um aus der Sackgasse raus zukommen und versucht jedes Mal eine andere Option zu wählen. Dies geschieht solange, bis die erste potentielle Lösung gefunden wurde oder bis alle alternativen Wege durchlaufen wurden und es somit zu keinem erfüllenden Assignment kommt.[4]

Zusammenfassend lässt sich also sagen, wenn es eine Lösung für das Problem gibt, kann es auf jeden Fall durch Backtracking gefunden werden. In Abbildung 1 sieht man, dass der Algorithmus immer von oben startet und als Erstes den linken Pfad entlang geht bis er scheitert, diesen zurückgeht und dann zum nächsten Pfad weitergeht, bis er (hier in grün) die Lösung findet. Natürlich hat hier die Wahl der Reihenfolge der Instanziierung erheblichen Einfluss auf die Laufzeit. Im Worst-Case ist diese exponentiell.

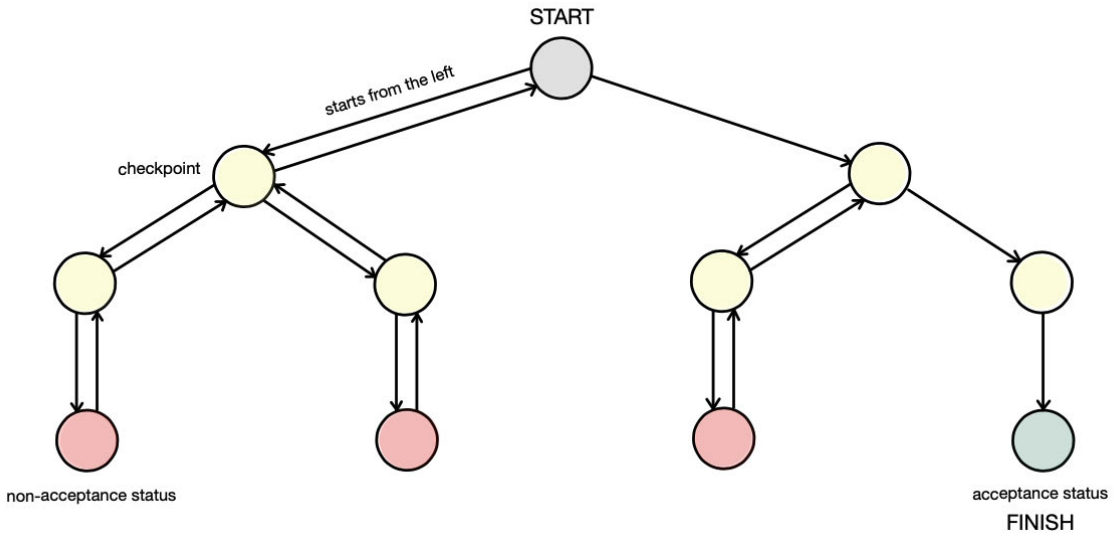


Abbildung 1: Backtracking anhand eines einfachen Suchbaums nach eigener Darstellung in Anlehnung an [10]

Bei klassischen Constraint-Satisfaction-Problemen gibt es also eine Menge von harten Randbedingungen, die von einer Reihe von möglichen Werten eingeschränkt werden. In der realen Welt sind diese Einschränkungen allerdings nur selten hart und bestimmte Präferenzen sind erwünscht. Auch sollen manche Bedingungen vor anderen Vorrang haben. Um dies realisieren zu können werden Nebenbedingungen durch unsichere Parameter erweitert, sogenannte Fuzzy-Constraint-Satisfaction-Probleme. [11]

2.2 Fuzzy Constraint-Satisfaction-Problem

Ein FCSP (Fuzzy-Constraint-Satisfaction-Problem) berücksichtigt im Gegensatz zu CSPs Präferenz- und Prioritätsbedingungen. Die Modellierung beider erfolgt durch eine Fuzzy-Relation R zwischen den Variablen der Teilmenge $X' = \{x_1, x_2, \ldots, x_k\}$ von X. Diese Fuzzy-Relation gibt einen Zufriedenheitsgrad an und ist ein Indikator für das AusmaSS, in dem diese weicheren Bedingungen erfüllt sind. Ist dieser gleich 0, erfüllt er die Bedingungen überhaupt nicht. Ist er gleich 1, so erfüllt er die Bedingungen vollständig. Im Gegensatz zu CSP sind allerdings auch Zwischenwerte zwischen 0 und 1 möglich, welches eine Teilerfüllung bedeutet. Der globale Zufriedenheitsgrad einer Fuzzy-Bedingung $\alpha_P \in [0, 1]$ gibt die Gesamtzufriedenheit an. Des Weiteren gibt es einen Schwellenwert

$\alpha_0 \in [0, 1]$, welcher eine festgelegte Untergrenze des akzeptablen globalen Zufriedenheitsgrades vom Benutzer ist. Das Ziel eines FCSPs ist die Kombination aus beiden Graden, sodass $\alpha_P \geq \alpha_0$ ist. [12]

Um zu veranschaulichen, wie Prioritäten in der realen Welt Einfluss nehmen, zeigt folgendes Beispiel. Angenommen ein Austauschstudent möchte sich eine Wohnung mieten und stellt folgende Bedingungen an die vermittelnde Agentur [13]:

$R1$: Die Unterkunft sollte eine bestimmte Qualität haben, aber nicht zu teuer sein.

$R2$: Da er nur für 1 Jahr Austauschstudent ist und gerne - aber nicht zu oft - umzieht, soll das Apartment unter einem Jahr gemietet werden.

$R3$: Die Wohnung sollte zu FuSS von der Universität aus erreichbar sein, da er sich dadurch teure Fahrten sparen kann.

Der Austauschstudent priorisiert die verschiedenen Bedingungen nach seiner Wahl und die Agentur bietet ihm Möglichkeiten an, welche am besten passen. Um eine Entscheidung zu treffen mit der die Bedingungen bestmöglich erreicht werden, können verschiedene Ansätze angewendet werden, die in dem folgenden Kapitel genauer betrachtet werden. Dieses Optimierungsproblem kann die unterschiedlichsten Parameter annehmen und ist somit flexibel auf mehrere Probleme anwendbar.

3 Challenges and Approaches of Solving Fuzzy Constraint-Satisfaction-Problems

Ziel bei der Herangehensweise eines FCSP ist es, die beste Lösung zu finden. Aber auch eine lokal ausreichende Lösung ist wünschenswert. Hierbei erfüllt eine bestimmte individuelle Beschränkung ein gegebenes Mimimum besser. Ebenso kann eine global ausreichende Lösung von Vorteil sein, da die gemeinsame Zufriedenheit ein gegebenes Minimum übersteigt. Der Schwellenwert kann als Prozentsatz des Zufriedenheitsgrades der Lösung angegeben werden oder als absolute Zahl. [14]

Das folgende Kapitel konzentriert sich auf die verschiedenen Herangehensweisen und deren Herausforderungen, wie Fuzzy Constraint Satisfaction Problems gelöst werden können.

3.1 Branch-and-Bound

Das Backtracking-Verfahren kann auch auf die FCSPs angewandt werden. Jedoch besteht hier das Problem, dass es eben nur stattfindet, wenn die Bedingungen verletzt sind. Für FCSPs kann man dies erweitern, indem Eigenschaften, wie der Grad der gemeinsamen Zufriedenheit, ausgenutzt und somit Schranken eingeführt werden. Dieser Grad kann unabhängig von der Zufriedenheit der einzelnen Nebenbedingungen oder monoton zu ihnen berechnet werden. Diese Erweiterung kann mithilfe des Branch-and-Bound-Algorithmus (BB) realisiert werden.[14]

Im BB-Algorithmus werden zuerst die freien Variablen im Suchbaum von oben beginnend ausgewählt, mögliche Belegungen werden durch eine foreach-Schleife getestet und anschlieSSend wird die Suche unterteilt. Dies geschieht, wenn die aktuelle Belegung

harte oder unwichtige Constraints verletzt, die im *bound* vorgegeben wurden. Die verletzten Constraints werden in der Variable *distance* verwaltet und misst die Anzahl an Beschränkungen, die durch die gewählten Werte verletzt werden. Alle Variablen haben eine Belegung, sobald ein Blattknoten erreicht wird. Die aktuelle Belegung wird zur neuen Lösung, wenn diese besser als der *bound* ist, andernfalls ändert sich nichts. Im Endeffekt werden also - durch Vergleichen - uninteressante Teilbäume verworfen, wenn diese zu einem schlechteren Ergebnis führen würden oder Widersprüche zu den Bedingungen erkannt werden. BB ist somit eine Erweiterung zu Backtracking, nur das hierbei nicht die erstbeste Lösung, die alle Bedingungen erfüllt, gewählt wird, sondern nach allen möglichen Lösungen gesucht wird. Beim BB wird eine Bewertungsfunktion verwendet, die die Anzahl der verletzten Beschränkungen beinhaltet und somit kann frühzeitig erkannt werden, wann ein Pfad abgeschnitten werden sollte. Abbruchbedingungen, wie z.B. Schranken, die besagen, dass wir mit einer gewissen Anzahl an verletzten Bedingungen zufrieden sind oder ein Timeout bei der Verarbeitung in Echtzeit, der die aktuell bisher beste Antwort liefert, können eingeführt werden. [2]

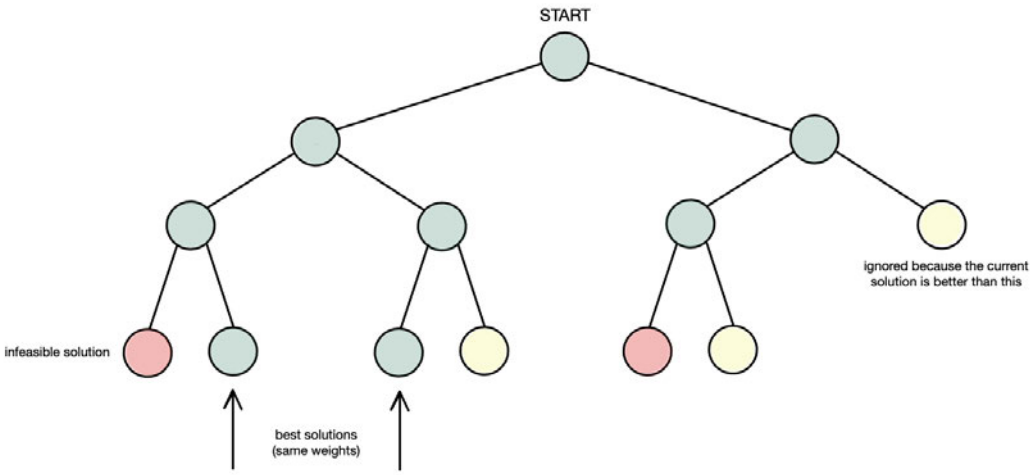


Abbildung 2: Branch-and-Bound-Suchbaum nach eigener Darstellung in Anlehnung an [15]

In Abbildung 2 wird anhand eines Suchbaumes dargestellt, wie der Algorithmus funktioniert. Die grünen Nodes signalisieren die erfolgreiche Suche, die roten Knoten werden aufgrund nicht erfüllter Bedingung verworfen und die gelben Nodes werden wegen einer besseren gefundenen Lösung ignoriert.

Je mehr Constraints erfüllt sind, umso besser ist die Lösung. Wie im Backtracking-Algorithmus ist auch hier die Worst-Case-Laufzeit exponentiell, wenn alle möglichen Kombinationen von Werten getestet und alle Bedingungen überprüft werden. BB ist im Vergleich zu Backtracking nicht schlechter, da hier nicht nur die perfekte Lösung gefunden wird.[2]

3.2 Forward-Checking

Freuder als auch Ruttkay verwenden als übliche Optimierungstechnik die BB-Methode, die auf das Backtracking aufbauen, da empirische Tests gezeigt haben, dass das reine Backtracking kein praktikabler Ansatz zur Lösung CSPs in der realen Welt sind. Eine andere Methode, die bisher die gröSSte Effektivität laut den meisten empirischen Studien zur konstruktiven heuristischen Suche bei CSPs bescheinigt, ist es, das Backtracking mit Forward-Checking (FC) zu kombinieren. FC funktioniert bei normalen CSPs folgendermaSSen: Sobald eine neue Variable instanziiert wird, erfolgt jedes Mal eine Überprüfung. Bei der Überprüfung wird darauf geachtet, ob die aktuell belegte Variable eine Bedingung mit einer noch nicht belegten Variable teilt. Aus der Domain der noch nicht belegten Variablen werden alle Werte eliminiert, die mit der Bedingung inkonsistent sind. Wird der Wertebereich der Domain einer solchen Variable leer, so ist es sinnlos diese weiter zu betrachten und sie wird verworfen. In Verbindung mit FCSP sind die Constraints nicht hart, sondern weich und der Zugehörigkeitsgrad wird mit einer bestimmten Untergrenze verglichen. Sobald der Grad unter diesen Schwellenwert gerät, kann die Variable gelöscht werden, da sie nicht zu einer besseren Lösung beiträgt. Damit das Backtracking in dieser Kombination auch funktioniert, muss ein weiterer Datensatz hinzugefügt werden, in dem die veränderten Domänen gespeichert werden. [3]

3.3 GENET

Ein anderer Lösungsansatz von FCSPs ist der sogenannte GENET-Algorithmus (GA). Hierbei handelt es sich um ein neuronales Netzmodell, in dem es Label-Knoten gibt, deren Zustand entweder *an* oder *aus* ist. Sobald sich ein Knoten im Zustand *an* befindet, wird die entsprechende Zuordnung gewählt. Um gleiche Variablen zu repräsentieren gibt es sogenannte Cluster. Eine Verbindung zwischen Knoten unterschiedlicher Cluster (ein Tupel) wird hergestellt, dessen erste Komponente der Zufriedenheitsgrad des kombinierten Labels (i, j) ist. Die zweite Komponente stellt das Gewicht der Connection dar. In jedem Cluster darf nur ein Label-Knoten angeschaltet sein, weshalb jeder Zustand eine Zuweisung eines Wertes zu jeder Variablen im gesamten Netz darstellt. Die Bestimmung des Anfangszustandes erfolgt zufällig, somit wird in jedem Cluster ein beliebiger Label-Knoten auf *an* gesetzt. AnschlieSSend berechnet jeder Knoten seine Eingaben parallel und asynchron in jedem Kovergenzzyklus. Der Knoten mit den meisten Eingaben in jedem Cluster wird eingeschaltet, alle anderen sind im Zustand *aus*. Gewinner eines jeden Clusters ist der Knoten mit der Variable mit den geringsten Einschränkungen. Ein stabiler Zustand wird nach einigen Zyklen erreicht und eine akzeptable Lösung wurde gefunden. Wird keine Lösung gefunden, befindet sich das Modell in einem lokalen Minimum. Wenn es allerdings mehrere Gewinnerknoten gibt, wird der zuletzt Eingeschaltete zum endgültigen Gewinner. Ist keiner eingeschalten, so wird nach dem Zufallsprinzip entschieden, um so einen chaotischen Zustandswechsel zu vermeiden. Um einen Weg aus einem lokalen Minimum zu finden, muss es mindestens zwei Knoten geben, die im Zustand *an* sind. Ihre Eingaben werden nach jedem Lernzyklus reduziert, solange sich der Netzzustand nicht ändert. Irgendwann wird i oder j nicht mehr in deren eigenen Cluster

gewinnen können und das lokale Minimum wird somit überwunden. In Abbildung 3 wird der Fuzzy (GA) dargestellt. [5]

randomly turn on a label node per cluster
for each $W_{i_y j_z}$ **do**
  set $W_{i_y j_z} = \alpha_{i_y j_z} - 1$
end for
while $\alpha_P < \alpha_0$ **do**
  for each cluster **do**
    (*asynchronously in parallel*)
    turn on only node with maximum input
    and keep original state if needed
  end for
  if local minimum reached **then**
    for each $W_{i_y j_z}$ **do**
      update $W_{i_y j_z}$:
      $W^{new}_{i_y j_z} \leftarrow W^{old}_{i_y j_z} + O^{old}_{i_y} O^{old}_{j_z} (\alpha_{i_y j_z} - 1)$
    end for
  end if
end while

Abbildung 3: Pseudocode des Fuzzy GENET-Algorithmus [5]

Zum besseren Verständnis dient Abbildung 4, in dem das bekannte Roboter-Kleidungsproblem Anwendung findet. Zur Erklärung: Ein Roboter hat aufgrund seiner kleinen Garderobe die Qual der Wahl. Ihm stehen Cordovans oder Turnschuhe, ein weiSSes oder grünes Hemd und 3 Hosenvarianten - Jeans, blaue Hose und graue Hose - zur Verfügung. Zu Beginn wurden ihm folgende Bedingungen gestellt: die Turnschuhe passen nur zu einer Jeans, die Cordovans harmonieren nur zu der grauen Hose und dem weiSSen Hemd, das weiSSe Hemd funktioniert entweder nur mit einer Jeans oder zur blauen Hose, und das grüne Hemd passt nur zu der grauen Hose. Offensichtlich gibt es hierfür keine Lösung, da es zu viele Einschränkungen gibt. Um eine bestmögliche Lösung zu generieren, werden verschiedene Zufriedenheitsgrade (siehe Abbildung 4) eingesetzt.

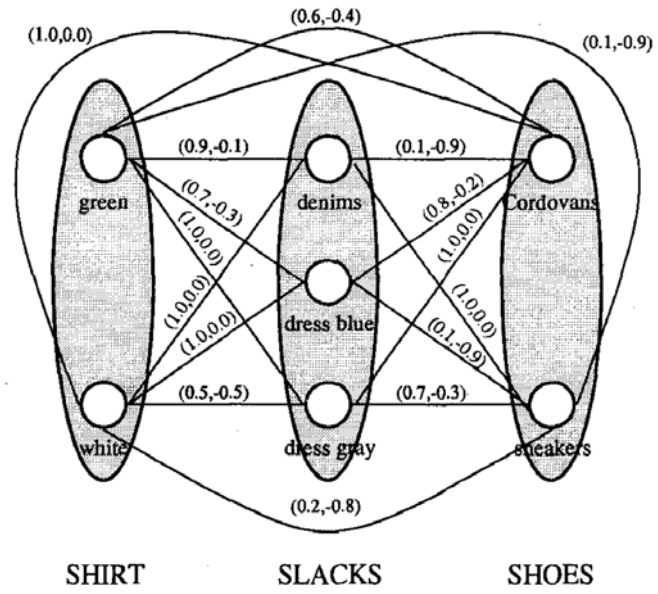


Abbildung 4: Fuzzy GENET-Algorithmus am Beispiel des Roboter-Kleidungsproblem [5]

3.4 Spread-Repair-Shrink

Ein e andere Herangehensweise FCSPs zu lösen, ist die Kombination aus der systematischen und der lokalen Suche. Hier gilt es die Vorteile beider auszunutzen, um so ein effizientes Lösen zu ermöglichen. Wie bei der systematischen Suche vorgegangen wird, haben wir bereits anhand der Beispiele BB und FC gesehen, die in Verbindung mit dem Backtracking stehen. In der lokalen Suche fand der Fuzzy GA Anwendung oder auch die Hill-Climbing-Methode, welche in diesem Paper nicht erläutert wurde. Der Spread-Repair-Shrink-Algorithmus (SRS) ist eine hybride Suche, der sich als effizientes Verfahren erweist, um für FCSPs eine qualitativ gute Näherunglösung zu generieren. Das grundlegende Vorgehen ist die lokale Suche mithilfe der systematischen Suche durchzuführen, um aus einem lokalen Optima zu entkommen und dabei die Ausbreitung von Beschränkungen zu kontrollieren. Der bereits entwickelte Spread-Repair-Algorithmus (SR) versucht wiederholt eine Zielbeschränkung zu reparieren, indem der Wert einer Variable in dessen Bereich geändert wird. Sobald sich der SR in ein lokales Maximum verirrt, versucht er, durch Änderung des Reperaturziels auf die in der Umgebung liegenden Nachbarbeschränkungen, heraus zu finden - das sogenannte *Spreading*. Der SRS erweitert den SR insoweit, dass der lokalen Suche eine neue systematische Funktion *Shrinking*, also Schrumpfen, hinzugefügt wird, die das Ziel zurück zum vorherigen Ziel ändert. Sie ist somit eine Umkehrung der Spreading-Funktion.Durch diese Kombination wird die in SR beobachtete Redundanz der Berechnung unterdrückt. Die Blattknoten entsprechen den potenziellen Zielrandbedingungen, die für die *Repair*-Funktion relevant sind und jeder Wurzelknoten entspricht einer der am meisten verletzten Randbedingungen. Der Algorithmus versucht durch die drei Funktionen die Zielbeschränkung effektiv

zu reparieren. Experimente mit den unterschiedlichsten Problemen haben ergeben, dass die Konvergenz bei SRS am besten ist. Fuzzy GENET hingegen könnte durch Ausnutzen des max-min-Problems verbessert werden. Die CPU-Zeit beider Varianten ist in Hinblick auf den Zufriedenheitsgrad laut der Experimente gleich. Die CPU-Zeit zur Findung der optimalen Lösung konnte durch SRS im Vergleich zu FC erheblich reduziert werden. Somit kann in kurzer Zeit eine relativ gute Näherungslösung für groSSe Probleme dank SRS gefunden werden. Abbildung 5 zeigt den Pseudocode des SRS-Algorithmus.[6]

```
Function SRS(constraints C)
    {input C: a set of the worst constraints}
    closed ← φ.
    open ← C.
    While C ≠ φ and open ≠ φ do
        node ← the first element of open.
        open ← open \ node.
        If not Repair(node) then
            Spread()
        Else if node ∈ C then
            C ← C \ {node}.
        Else if Repair(the parent of node) then
            Shrink( )
        Else
            Spread( )
        End if
        Sort open by heuristic h( ).
    End while
    If C = φ then return TRUE
    Else return FALSE
End function

Procedure Spread( )
    closed ← closed ∪ node.
    open ← open ∪ neighbors(node) \ closed.
End procedure

Procedure Shrink( )
    node ← parent of node.
    open ← open \ the descendants of node.
    closed ← closed \ descendants of node.
    If node ∈ C then
        C ← C \ {node}.
    Else
        open ← open ∪ node.
        If Repair(the parent of node) then Shrink( )
    End if
End procedure

Function Repair(constraint c)
    Try to repair c.
    If repaired then
        return TRUE
    Else
        return FALSE
    End if
End function

Begin program
    Do while SRS(worst constraints).
End program
```

Abbildung 5: Pseudocode des Spread-Repair-Skrink-Algorithmus [6]

4 Fazit

In dieser Arbeit wurden die Herausforderungen und Herangehensweisen zum Thema Fuzzy Constraint-Satisfaction-Problemen vorgestellt. Im ersten Teil dieses Papers wurden die Grundlagen zu Constraint-Satisfaction-Problems und dessen Erweiterung Fuzzy Constraint-Satisfaction-Problem erläutert. Um CSPs zu lösen wurde die Standardmethode Backtracking vorgestellt. AnschlieSSend wurde auf vier Möglichkeiten ein FCSP zu lösen eingegangen. Als Erstes begonnen wurde mit dem sehr bekannten Branch-and-Bound-Algorithmus, welcher in Kombination mit Backtracking verwendet wurde. Danach wurde ebenfalls Forward-Checking in Verbindung mit Backtracking gebracht. Um einen Gegensatz zur systematischen Suche zu betrachten, wurde dann auf eine lokale Suche, dem GENET-Algorithmus, eingegangen. Die Kombination der beiden Suchen erfolgte im Spread-Repair-Shrink-Algorithmus, welcher den zuvor erforschten SR-Algorithmus erweiterte. Es gibt natürlich noch viele weitere Ansätze zu diesem Thema, aber aufgrund der begrenzten Seitenanzahl wurde nur auf die vielversprechendsten Heuristiken eingegangen.

Zusammenfassend lässt sich sagen, dass der erweiterte Bereich von CSP, die Fuzzy CSPs, schon seit Jahrzehnten erforscht werden. Sie spielen in der Forschung als auch in der Industrie eine groSSe Rolle und haben in den letzten Jahren auch schon beträchtliche Erfolge erzielt. Zukünftige Forschungsarbeiten könnten sich noch weiter auf die Qualität und die Laufzeit konzentrieren, um effizientere Lösungen zu erzielen.

Literatur

[1] Jun Gu u. a. *Algorithms for the Satisfiability (SAT) Problem: A Survey,* CINCINNATI UNIV OH DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 1996. URL: https://apps.dtic.mil/sti/citations/ADA326042.

[2] Eugene C. Freuder und Richard J. Wallace. „Partial constraint satisfaction“. In: *Artificial Intelligence* 58.1 (1992), S. 21 70. ISSN: 00043702. DOI: 10.1016/0004-3702(92)90004-H. URL: https://linkinghub.elsevier.com/retrieve/pii/000437029290004H.

[3] H.W. Guesgen und A. Philpott. „Heuristics for solving fuzzy constraint satisfaction problems“. In: *Proceedings 1995 Second New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems.* 1995, S. 132 135. DOI: 10.1109/ANNES.1995.499457.

[4] Daniel Hunter Frost. *Algorithms and Heuristics for Constraint Satisfaction Problems.* 1997. URL: https://www.ics.uci.edu/~dechter/publications/R69.pdf (besucht am 07. 06. 2022).

[5] J.H.Y. Wong und Ho-Fung Leung. „Solving fuzzy constraint satisfaction problems with fuzzy GENET“. In: *Proceedings Tenth IEEE International Conference on Tools with Artificial Intelligence (Cat. No.98CH36294).* 1998, S. 184 191. DOI: 10.1109/TAI.1998.744840.

[6] Yasuhiro Sudo und Masahito Kurihara. „A New Hybrid Algorithm for Solving Fuzzy Constraint Satisfaction Problems“. In: *SCIS and ISIS 2006* (2006), S. 7.

[7] Eugene C. Freuder und Alan K. Mackworth. „Constraint Satisfaction: An Emerging Paradigm“. In: *Foundations of Artificial Intelligence*. Bd. 2. Elsevier, 2006, S. 13 27. ISBN: 978-0-444-52726-4. DOI: 10.1016/S1574-6526(06)80006-4. URL: https://linkinghub.elsevier.com/retrieve/pii/S1574652606800064 (besucht am 14. 06. 2022).

[8] Igor Rivin, Ilan Vardi und Paul Zimmermann. „The n -Queens Problem“. In: *The American Mathematical Monthly* 101.7 (1994), S. 629 639. ISSN: 0002-9890, 1930-0972. DOI: 10.1080/00029890.1994.11997004. URL: https://www.tandfonline.com/doi/full/10.1080/00029890.1994.11997004.

[9] Sally C. Brailsford, Chris N. Potts und Barbara M. Smith. „Constraint satisfaction problems: Algorithms and applications“. In: *European Journal of Operational Research* 119.3 (1999), S. 557 581. ISSN: 03772217. DOI: 10.1016/S0377-2217(98)00364-6. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221798003646.

[10] *File:backtracking.jpg*. 2021. URL: https://dev.to/josethz00/what-is-backtracking-56cg (besucht am 09. 06. 2022).

[11] Didier Dubois, Helene Fargier und Henri Prade. „Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty“. In: *Applied Intelligence* 6.4 (1996), S. 287 309. ISSN: 0924-669X, 1573-7497. DOI: 10.1007/BF00132735. URL: http://link.springer.com/10.1007/BF00132735.

[12] Jason H Y Wong und Ho-fung Leung. „Extending GENET to Solve Fuzzy Constraint Satisfaction Problems“. In: *AAAI-98 Proceedings* (1998), S. 6.

[13] Xudong Luo u. a. „Prioritised fuzzy constraint satisfaction problems: axioms, instantiation and validation“. In: *Fuzzy Sets and Systems* 136.2 (2003), S. 151 188. ISSN: 01650114. DOI: 10.1016/S0165-0114(02)00385-8. URL: https://linkinghub.elsevier.com/retrieve/pii/S0165011402003858.

[14] Z. Ruttkay. „Fuzzy constraint satisfaction“. In: *Proceedings of 1994 IEEE 3rd International Fuzzy Systems Conference*. 1994, 1263 1268 vol.2. DOI: 10.1109/FUZZY.1994.343640.

[15] *File:branchandbound.jpg*. 8. Apr. 2022. URL: https://www.geeksforgeeks.org/0-1-knapsack-using-branch-and-bound (besucht am 09. 06. 2022).

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat.

Alle Ausführungen der Arbeit, die wörtlich oder sinngemäSS übernommen wurden, sind als solche gekennzeichnet.

Augsburg, 21. Juni 2022