# Heuristische Optimierung durch menschliche Intuition – Das Beste aus zwei Welten

Iona Kuhn[1]

**Abstract:** Algorithmen können exakte Lösungen finden, indem sie den Suchraum durchlaufen. Ist das Problem aber zu groß, haben Algorithmen oft Schwierigkeiten eine gute Lösung in akzeptabler Zeit zu finden. Menschen hingegen scheinen komplizierte Probleme oft schnell „intuitiv“ zu lösen. Ziel dieser Arbeit ist es daher, menschliche Intuition zur Verbesserung von Heuristiken am Beispiel von Job-Shop-Problemen zu nutzen. Bei Job-Shop-Problemen müssen mehrere Aufträge auf unterschiedlichen Maschinen möglichst schnell erledigt werden, also die richtige Bearbeitungsreihenfolge gefunden werden. Um die Intuition von Menschen verwenden zu können, wurde ein Gamification-Ansatz eingesetzt, also ein Spiel implementiert in dem Menschen das Problem in einer übertragenen, vereinfachten Form lösen. Um das gewonnene menschliche Wissen in Heuristiken einzuarbeiten wurden zwei verschiedene Ansätze entwickelt und mit diesen Vergleiche zur ursprünglichen Heuristik durchgeführt. Dabei konnte gezeigt werden, dass in der Tat die Effizienz von Heuristiken mit menschlicher Intuition gesteigert werden kann.

**Keywords:** Heuristik, menschliche Intuition, Gamification, genetischer Algorithmus, Job-Shop-Problem, case-based reasoning

## 1 Einleitung

Für viele Herausforderungen in der heutigen Welt verwenden wir Algorithmen. Ist ein Problem zu komplex (wie z.B. das Traveling Salesman Problem), kann es aber zu lange dauern, es mit einem exakten Verfahren zu berechnen. Dann setzt man eine Heuristik ein, die sich z.B. durch zufälliges Ausprobieren im Suchraum der Lösung annähert, statt den Suchraum vollständig zu durchsuchen. Dabei ist aber nicht gesichert, dass immer die beste Lösung gefunden wird. Außerdem können auch diese Verfahren sehr lange dauern.

Wenn Menschen solche komplexen Herausforderungen angehen, hilft ihnen oft ihre Intuition, eine möglichst gute Lösung zu finden. Allerdings sind viele dieser Probleme zu komplex, um alleine von einem Menschen gelöst zu werden. Darüber hinaus können Computer wesentlich schneller rechnen.

Daher ist das Ziel dieser Arbeit, Heuristiken zu entwickeln und mit Hilfe von Strategien, die man aus dem Vorgehen von Menschen gewinnt, zu verbessern. Hierzu wurde die Problemklasse der Job-Shop-Probleme als Anwendungsdomäne gewählt. Ein Job-Shop-Problem ist eine Unterklasse von Scheduling-Problemen. Dabei geht es darum eine Reihe von Aufträgen, die auf unterschiedlichen Maschinen bearbeitet werden müssen,

[1] Otto-Schott-Gymnasium, An Schneiders Mühle 1, 55124 Mainz, iona.kuhn@web.de

möglichst effizient abzuarbeiten, d.h. in der Reihenfolge mit der insgesamt kürzesten Laufzeit und mit den geringsten Leerzeiten für die Maschinen.

Als Heuristik wurde eine vereinfachte Form eines genetischen Algorithmus gewählt, da solche Algorithmen oft zur Lösung von dieser und vergleichbaren Problemklassen verwendet werden [MHZ10]. Zudem lässt sich bei der vereinfachten Form eines genetischen Algorithmus die heuristische Suchkomponente (bezeichnet als Mutationsoperator) leicht austauschen.

Um den genetischen Algorithmus für die gewählte Problemklasse nun zu verbessern, muss man zunächst an die Strategien herankommen, mit denen Menschen solche Herausforderungen intuitiv angehen. Dafür wurde ein Gamification-Ansatz gewählt. Das bedeutet, dass man ein Computerspiel entwickelt, das einem Job-Shop-Problem entspricht. Mit dem Wissen aus den Daten, welche die Testspieler erzeugen, kann dann der genetische Algorithmus angepasst und mit der ursprünglichen Version verglichen werden.

## 2 Verwandte Arbeiten

Bereits früher wurden Versuche unternommen, menschliche Erfahrungen algorithmisch zu nutzen. In [Wu02] wurde zum Beispiel ein Algorithmus auf alten Weisheiten aufgebaut um eine Verbesserung zu erzielen. Das algorithmische Nutzen von menschlichen Vorgehensweisen ist aber immer noch Bestandteil aktueller Forschungen. So wurden beispielsweise in [KUG18] bekannte (hypothetische) menschliche Verhaltensweisen beim Lösen des Traveling Salesmen Problems eingesetzt. Bei diesen beiden Arbeiten wurden jeweils vorhandene bzw. angenommene Strategien von Menschen verwendet, der Mensch aber nicht selbst eingesetzt, um Strategien zu erzeugen.

In [JBS17] wird gezeigt, dass man Heuristiken mit menschlichen Strategien verbessern kann. Die Autoren haben aber den Menschen immer konkret nach ihrer Strategie befragt und nicht zunächst unabhängig menschliche Strategien gesammelt.

Der Ansatz der Gamification wurde schon häufiger benutzt (siehe etwa [HKS14] für einen Vergleich verschiedener Ansätze), beispielweise um die Motivation und das Engagement von Mitarbeiter eines Unternehmens zu erhöhen. In der hier beschriebenen Arbeit geht es jedoch darum Gamification zu nutzen um ein Verfahren zur Verbesserung von algorithmischen Heuristiken durch menschliche Intuition zu entwickeln.

## 3 Job-Shop-Problem

Bei dem hier eingesetzten Job-Shop-Problem sollen $n$ Jobs auf $m$ Maschinen bearbeitet werden. Jeder Job $j$ besteht dabei aus $t_j$ Teilarbeiten. Die Reihenfolge der Teilarbeiten eines jeden Jobs ist klar definiert. Innerhalb eines Jobs muss jede Teilarbeit auf einer

anderen Maschine ausgeführt werden. Welche Maschine das ist, steht von Anfang an fest. Die Bedingung dafür, wann mit der Arbeit an einer Teilarbeit begonnen werden kann, ist, dass die vorhergehende Teilarbeit fertig und dass die benötigte Maschine frei ist. Es gibt keinen Wartebereich, d.h. wenn eine Teilarbeit auf einer Maschine fertig ist, kann diese erst wieder gestartet werden, wenn das Produkt zur nächsten Maschine gebracht worden ist. Es werden keine Transportzeiten zwischen den Maschinen und keine Rüstzeiten betrachtet. Alle Maschinen sind vom Zeitpunkt 0 an immer verfügbar. Die Jobs können zum Zeitpunkt 0 beginnen. Es existieren keine parallelen Maschinen oder Maschinengruppen. Keine Maschine kann mehrere Teilarbeiten gleichzeitig bearbeiten. Alle Jobs haben die gleiche Priorität. Es gibt keine Freigabe- oder Fälligkeitstermine für die Jobs. Leerzeiten auf den Maschinen und Wartezeiten in den Jobs sind zulässig. Eine angefangene Teilarbeit kann nicht unterbrochen werden. Eine Maschine hat eine feste Bearbeitungsdauer. Jede Teilarbeit auf einer Maschine dauert also gleichlang (vgl. auch [He03, Re17]).

Die Zielsetzung ist, die gesamte Bearbeitungsdauer aller Jobs zu minimieren. Außerdem muss nun festlegt werden, mit wie vielen Maschinen gearbeitet werden soll. Hier wurde $m = 6$ gewählt, damit es im Rahmen der Gamification für einen menschlichen Spieler nicht zu umfangreich wird.

## 4 Das Problem als Spiel – Gamification

Um an die Strategien heranzukommen, mit denen der Mensch ein Job-Shop-Problem mittels Intuition löst, wurde wie oben erläutert ein Gamification-Ansatz gewählt. Dazu wurde ein Spiel entworfen, das sechs Maschinen umfasst, mit denen unterschiedliche Produkte hergestellt werden können. Hier wurde dabei ein Kochspiel gewählt (Abb. 1), welches ein Spiel aus dem Genre des Zeitmanagements ist.


Abb. 1: Bild des Startbildschirmes des Spiels
(beispielhaft ist die Anleitung für die Gnocchi angezeigt)

Die sechs Maschinen des Job-Shop-Problems entsprechen dann sechs Küchengeräten, mit denen zehn verschiedene Produkte aus den Zutaten hergestellt werden können. Die Zutaten können unmittelbar zur Verfügung stehen, müssen aber auch zum Teil zuerst mit anderen Maschinen (und Zutaten) hergestellt werden. Das Ziel des Spieles ist es, die vorgegebene Bestellung in Form von Aufträgen (= Herstellung eines Produktes, vorher Job genannt) möglichst schnell abzuarbeiten. Wichtig dabei ist die Reihenfolge, in der die Aufträge durchgeführt werden.

Bei dem Spiel gibt es vier Level mit 3, 5, 7 bzw. 9 herzustellenden Aufträgen. Bei jedem Level wird beim Starten zufällig eine Bestellung mit der entsprechenden Anzahl an Aufträgen generiert. Jedes Level muss später von einem Spieler fünfmal gespielt werden, wobei ein Spieler in einem Level immer die gleiche Bestellung erhält. Diese Entscheidung wurde getroffen, damit jeder Spieler sich verbessern kann und so beim letzten Durchlauf eines Levels möglichst die optimale Zeit erreicht.

Im Folgenden sind alle 10 Aufträge, welche im Spiel auftauchen können in abstrakter Form (als Liste von Listen) angegeben. Dabei ist in den Unterlisten immer die Maschine (mit m1 = „Schäler“, m2 = „Schneider“, m3 = „Gnocchimaschine“, m4 = „Ofen“, m5 = „Verzierer“, m6 = „Mixer“) und die entsprechend für den Arbeitsschritt benötigt Zeit dazu aufgelistet. In der Reihenfolge, wie sie in der übergeordneten Liste sind, müssen sie auch abgearbeitet werden.

| | |
|---|---|
| a1 („Pommes“): | `[[m1, 10], [m2, 10], [m4, 20]]` |
| a2 („Gnocchi“): | `[[m1, 10], [m3, 20]]` |
| a3 („Schokokuchen“): | `[[m4, 20], [m5, 5]]` |
| a4 („Milchshake“): | `[[m6, 15]]` |
| a5 („Obstsalat“): | `[[m1, 10], [m2, 10]]` |
| a6 („Schokoeis“): | `[[m5, 5]]` |
| a7 („Smoothie“): | `[[m1, 10], [m6, 15]]` |
| a8 („Schokomilchshake“): | `[[m5, 5], [m6, 15]]` |
| a9 („Pommes mit Ketchup“): | `[[m1, 10], [m2, 10], [m4, 20], [m5, 5]]` |
| a10 („Kakao“): | `[[m5, 5]]` |

Die Spieldaten wurden für die Auswertung in einzelnen Textdateien gespeichert.

## 5 Genetischer Algorithmus

Ein genetischer Algorithmus besteht in der Regel aus einer Liste mit Lösungsindividuen, aus denen durch Selektion, Rekombination und Mutation neue Individuen erzeugt und mit Hilfe einer Fitnessfunktion bewertet werden. Dieses wird so oft wiederholt, bis eine hinreichend gute Lösung gefunden wird bzw. für eine vorher festgelegte Anzahl an Durchläufen (im folgenden Iterationen genannt). Bei dem hier verwendeten Algorithmus wurde auf die Rekombination verzichtet, wodurch das Rahmenwerk vereinfacht wird und es dadurch nur eine austauschbare Komponente gibt. Somit entspricht der gewählte genetische Algorithmus einer Art Hill-Climbing-Verfahren (welches unter Umständen

sogar vorteilhaft seien kann, vgl. [FM93]). Des Weiteren wurde auch der Selektionsoperator verändert, in dem nicht immer aus den besten Individuen neue erzeugt werden, sondern aus zufälligen. Dieser Ansatz wurde gewählt, da man teilweise über eine zunächst schlechtere Lösung eine deutlich bessere Lösung finden kann.

Auf das oben beschriebene Job-Shop-Problem übertragen ergibt sich entsprechend, dass ein Lösungsindividuum die Reihenfolge ist, in der bei einem Problem die einzelnen Aufträge durchgeführt werden. In der Liste der Individuen sind somit verschiedene Permutationen, von denen dann mittels der Fitnessfunktion die Zeit berechnet wird, die man bräuchte, wenn man die Aufträge in dieser Reihenfolge abarbeiten würde. Das Individuum mit der kürzesten Zeit ist dann dementsprechend auch das beste. Als Mutation werden zur Erzeugung neuer Individuen in zufällig ausgewählten Individuen zwei Aufträge vertauscht, sodass sich eine neue Reihenfolge der Abarbeitung ergibt. Nun werden die neuen Individuen der Liste hinzugefügt und man löscht die entsprechende Anzahl an Individuen mit der längsten Bearbeitungszeit für die Aufträge. Dieser Zyklus von Auswählen, Mutation, Auswertung und Löschen wurde dann für ein bestimmtes Problem im Regelfall 1000mal iteriert. Dies ergibt dann die beste vom genetischen Algorithmus gefundene Lösung mit Zeit und Reihenfolge der Aufträge.

## 6 Auswertung der Spieldaten

Das in Kapitel 4 beschriebene Spiel wurde von insgesamt 70 Spielern gespielt. Für das Ziel dieser Arbeit war es zuerst wichtig zu untersuchen, ob die Spieler im Laufe des Spiels etwas gelernt haben. Dieses ist in der Tat so, wie man in Abb. 2 beispielshaft für Level 3 und 4 erkennt (grüne Kurve). Im Durchschnitt wird die Zeit für ein Level mit jedem Durchlauf kürzer und somit beim letzten Durchlauf die schnellste Lösung der Spieler erreicht.

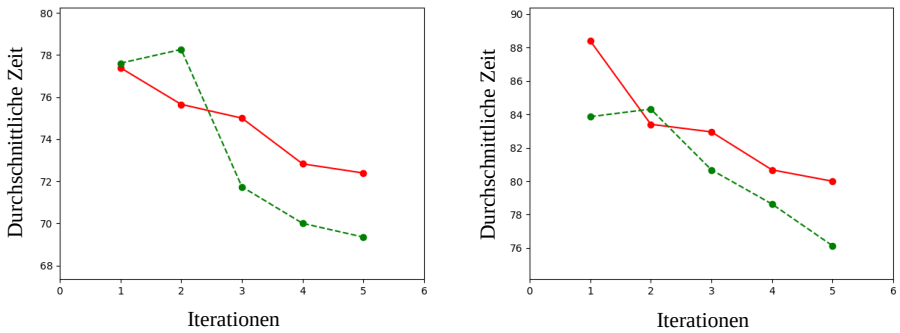


Abb. 2: Der Graph für das dritte Level ist links und der Graph für das vierte Level ist rechts. Dabei zeigt der grüne Graph die durchschnittliche Zeit (der erspielten Reihenfolge) des Menschen und der rote Graph die Zeit des genetischen Algorithmus.

Weiter stellt sich die Frage, ob der Mensch überhaupt besser ist als die Heuristik, also der genetische Algorithmus. Hierzu wurde das Spiel auch von einem genetischen

Algorithmus „gespielt“, dem die gleichen Bedingungen wie dem Menschen gegeben wurden (d.h. eine Individuenliste der Länge 1 und pro Iteration ein neues Individuum). Bei diesem Vergleich kann man deutlich erkennen, dass der genetische Algorithmus schlechter ist (rote Kurve in Abb. 2). Somit sollte es in der Tat möglich sein, den genetischen Algorithmus zu verbessern, indem man die menschlichen Strategien mit einbaut.

## 7 Nutzung der menschlichen Strategien

Nun soll die menschliche Intuition bzw. die hierbei verwendeten Strategien genutzt werden um neue Probleme zu lösen. Dafür wurden zwei Ansätze getestet: (i) das Fall-basierte Schließen (case-based reasoning, kurz: CBR) und (ii) die Anpassung des genetischen Algorithmus.

### 7.1 CBR-Algorithmus

Im Folgenden wird ein CBR-basierten-Ansatz (auf Basis von [BK08]) als alternative Heuristik eingesetzt um menschliche Intuition nutzbar zu machen. Damit nutzt man im Gegensatz zum genetischen Algorithmus, in dem die menschliche Intuition in aggregierter Form genutzt wird, das Einzelfallwissen der menschlichen Intuition. Ein CBR-Ansatz funktioniert allgemein so: Wenn ein neues Problem vorliegt, wird in einer Datenbank mit Problem/Lösungs-Paaren das ähnlichste Problem gesucht (retrieve). Dessen Lösung wird genommen (reuse) und für das neue Problem (revise) konvertiert. Die neue, angepasste Lösung wird dann üblicherweise als Lösung für das neue Problem zurückgegeben und in die Datenbank eingefügt (retain).

Dieser Ansatz wurde mit Hilfe von selbst entwickelten Verfahren auf das Job-Shop-Problem übertragen. Im Gegensatz zum herkömmlichen CBR-Verfahren wird das neue Problem-Lösungs-Paar nicht zur Datenbank hinzugefügt, da diese ausschließlich Problem-Lösungs-Paare enthalten soll, welche direkt vom Menschen erzeugt worden sind.

Der so entwickelte CBR-Ansatz wurde nun mit dem genetischen Algorithmus verglichen. Dafür wurden Probleme unterschiedlicher Länge gewählt: mit 55 Aufträgen (als ein sehr komplexes Problem) oder mit 9 Aufträgen (wie maximal beim Gamification-Ansatz). In beiden Fällen wurden für den genetischen Algorithmus die folgenden vier Varianten verwendet:

geneAlgo1: 1 Lösungsindividuum pro Generation, 1 neues Individuum pro Iteration
geneAlgo2: 2 Lösungsindividuen pro Generation, 1 neues Individuum pro Iteration
geneAlgo3: 20 Lösungsindividuen pro Generation, 1 neues Individuum pro Iteration
geneAlgo4: 20 Lösungsindividuen pro Generation, 2 neues Individuum pro Iteration

Die erste Variante dient als Beispiel für den einfachsten genetischen Algorithmus bzw. den Hill-Climbing-Ansatz. Bei den weiteren Varianten wurde die Anzahl an Lösungsindividuen erhöht, so dass der Suchraum einfacher (und schneller) erschlossen werden kann und sich der Algorithmus nicht so leicht in einem lokalen Optimum verfängt.

Das Problem mit 55 Aufträgen wurde vom CBR-Ansatz und den vier genetischen Algorithmen jeweils 20mal gelöst. Die Ergebnisse sind in Tab. 1 gezeigt. Die Ergebnisse des Vergleichs der vier genetischen Algorithmen und des CBR-Ansatzes mit den Problemen mit 9 Aufträgen sind in Tab. 2 gezeigt.

| | beste gefundene Lösung | Durch-schnittlich gefundene Lösung | Anzahl gebrauchter Iterationen für Lösung | Laufzeit dafür [s] | Iterationen um beste Lösung vom CBR-Algorithmus zu finden | Laufzeit dafür [s] |
|---|---|---|---|---|---|---|
| geneAlgo1 | 320 | 332,00 | 502,55 | 119,01 | 44,10 | 11,96 |
| geneAlgo2 | 320 | 333,75 | 471,75 | 112,72 | 41,55 | 11,83 |
| geneAlgo3 | 320 | 333,50 | 756,25 | 197,22 | 108,70 | 35,49 |
| geneAlgo4 | 320 | 328,25 | 617,60 | 297,50 | 46,10 | 31,51 |
| Durch-schnitt | 320 | 331,88 | 587,04 | 181,61 | 60,11 | 22,70 |
| CBR-Algo | 380 | 426,25 | | 0,51 | | |
| Verhältnis | 1,19 | 1,28 | | 356,10 | | 44,50 |
| | > 19 % schlechter | **> 28 % schlechter** | | **> 356,10mal so schnell** | | > 44,50mal so schnell |

Tab. 1: Ergebnisse des Vergleiches von CBR-Algorithmus und genetischen Algorithmus bei einem Problem der Länge 55.

| | durchschnittlich gefundene Lösung | Anzahl gebrauchter Iterationen für Lösung | Laufzeit dafür [s] |
|---|---|---|---|
| geneAlgo1 | 78,96 | 20,49 | 5,92 |
| geneAlgo2 | 78,74 | 19,71 | 6,08 |
| geneAlgo3 | 77,86 | 27,85 | 13,79 |
| geneAlgo4 | 77,90 | 15,11 | 14,36 |
| Durchschnitt | 78,37 | 20,79 | 10,04 |
| CBR-Algo | 89,33 | | 0,07 |
| Verhältnis | 1,14 | | 143,39 |
| | **> 14 % schlechter** | | **> 143,39mal so schnell** |

Tab. 2: Ergebnisse des Vergleiches vom CBR-Algorithmus und genetischen Algorithmus bei Problemen der Länge 9 (Durchschnitt für 70 unterschiedliche Probleme).

Dass der CBR-Algorithmus eher schlechtere Lösungen findet, lässt sich bei der Länge 55 vermutlich dadurch erklären, dass keine so großen Probleme in der Datenbank gespeichert sind. Ein weiterer Grund könnte darin bestehen, dass der Mensch wahrscheinlich nicht immer die optimale Lösung beim Spielen des Spiels gefunden hat, und daher in der Datenbank auch Problem-Lösungs-Paare vorhanden sind, welche nicht optimal sind. Hervorzuheben ist aber, dass der CBR-Ansatz deutlich schneller zu einer leicht schlechteren Lösung gekommen ist, wohingegen die genetischen Algorithmen in diesem Fall deutlich langsamer sind. Dies liegt daran, dass der genetische Algorithmus viele Lösungen ausprobiert und durchrechnet, während der CBR-Algorithmus nur eine Lösung konvertieren muss. Zusammenfassend kann man sagen, dass mit menschlicher Intuition, eingesetzt in Form eines CBR-Ansatzes, nicht unbedingt die optimale Lösung gefunden wird, aber man aufgrund des Algorithmus deutlich schneller auf eine sehr gute Lösung kommt. Dies ist für viele Anwendungen von einem Job-Shop-Problem bereits ausreichend – vor allem im Hinblick auf die wesentlich geringere Berechnungszeit.

## 7.2 Anpassung des genetischen Algorithmus mit menschlichem Wissen

Um den genetischen Algorithmus mit den Strategien aus der menschlichen Intuition zu verbessern, mussten aus der Datenbank entsprechende Regeln abgeleitet werden. Um möglichst einfache, generische und kompakte Entscheidungsregeln zu erhalten, werden immer zwei Aufträge miteinander verglichen. Dafür werden die Merkmale der Aufträge hierfür in abstrakter Form beschrieben (bezeichnet als Bedingung). Die Reihenfolge, in der diese Aufträge gemäß Datenbank, also des Menschen, am besten abgearbeitet werden, ist dann die Schlussfolgerung basierend auf der Bedingung.

Mit den in Kapitel 4 beschriebenen 10 Aufträgen ergeben sich 45 (= 10 * 9 / 2) Paare an Aufträgen. Mit jedem dieser Paare wird dann in den gesammelten Daten geschaut, ob es ein Problem gibt, das die beiden Aufträge enthält. Wenn dieses der Fall ist, wird mit diesen beiden Aufträgen eine Datenzeile (siehe Abb. 3) bestehend aus Bedingung und Schlussfolgerung erzeugt.

```
j1_KKE ^ j2_ES ^ gemein_3.1 ^ j2_kuerzer  => j1_j2
```

Abb. 3: Eine beispielshafte Datenzeile mit Bedingung und Schlussfolgerung. Die Bedingung ist vor dem Folgepfeil und besteht aus einigen Parametern, welche die beiden Aufträge abstrakt beschreiben und verbinden. Hinter dem Folgepfeil befindet sich die Schlussfolgerung, also welchen der beiden Aufträge der Mensch als erstes gestartet hat.

Dieses wurde mit allen 45 Kombination von zwei Aufträgen für jedes Problem in der Datenbank durchgeführt, in der diese beiden Aufträge vorkommen, woraus sich insgesamt 2583 Datenzeilen ergeben haben. Die Datei mit diesen Datenzeilen wurde dann mittels eines Wissensextraktions-Algorithmus [AK17] weiterverarbeitet, der eine mehrstufige Wissensbasis zurückgibt, wobei auf der ersten Stufe eine allgemeine Regel steht und auf den tieferen Stufen die Ausnahmen der Regel bzw. Ausnahmen von Ausnahmen, etc. D.h. dieser Algorithmus komprimiert die Datenzeilen zu Regeln. Dabei werden die Bedingungen zu sogenannten Prämissen und die entsprechende

Schlussfolgerung zur Konklusion. So wurde eine Datei mit den komprimierten Regeln erhalten, die in Abb. 4 gezeigt ist.

Die so erhaltenen Regeln müssen nun in den genetischen Algorithmus eingearbeitet werden. Dafür wurde der Mutationsoperator so verändert, dass vor einem Vertauschen der beiden Aufträge diese mit den erhaltenen Regeln überprüft werden. Dazu wird mit Hilfe eines Inferenzalgorithmus [AK16] kontrolliert, ob die beiden zur Mutation ausgewählten Aufträge schon in der richtigen Reihenfolge vorliegen. Ist dieses nicht der Fall, so werden diese getauscht, d.h. die Mutation erfolgt, und die neue Reihenfolge wird der Individuenliste hinzugefügt. Ansonsten erfolgt keine Mutation und es wird zufällig ein anderes Auftragspaar ausgewählt. Somit ist die Mutation abhängig von den Regeln, welche aus den gesammelten Daten über die menschliche Intuition erzeugt wurden.

```
j2_j1 [0.75]

j1_ES => j1_j2 [1.0]
gemein_1.3_2.4 => j1_j2 [1.0]
j1=j2 => j1_j2 [0.78]
j1_KKES => j1_j2 [0.98]
j2_KK => j1_j2 [0.98]
j2_SL => j1_j2 [0.86]
j1_KKE => j1_j2 [0.83]
j2_KL => j1_j2 [0.98]

gemein_2.2 ^ j2_SL => j2_j1 [1.0]
gemein ^ j1_S => j1_j2 [0.99]
gemein_1.1 ^ j1_S => j1_j2 [1.0]
gemein ^ j1_SL => j1_j2 [0.89]
gemein ^ j1_L => j2_j1 [0.87]
j2_L ^ j2_kuerzer => j2_j1 [0.92]
```

```
j1=j2 ^ j2_SL => j2_j1 [1.0]
gemein_1.1_2.2_3.3 ^ j1_KKE => j2_j1 [1.0]
gemein_2.2 ^ j2_kuerzer => j2_j1 [1.0]
gemein_2.2 ^ j1_SL => j1_j2 [1.0]
gemein_2.2 ^ j1_kuerzer => j1_j2 [1.0]
j1_KKE ^ j2_KKES => j2_j1 [1.0]
j1_KKE ^ j2_KE => j2_j1 [1.0]
j1_KKE ^ j1_kuerzer => j2_j1 [1.0]
gemein ^ j2_kuerzer => j2_j1 [0.87]
gemein ^ j2_KE => j1_j2 [0.86]
j1_kuerzer ^ j2_KKE => j2_j1 [0.84]

gemein_1.1 ^ j2_S ^ j2_kuerzer => j2_j1 [1.0]
gemein ^ j2_SL ^ j2_kuerzer => j1_j2 [1.0]
gemein ^ j1_SL ^ j1_kuerzer => j2_j1 [1.0]
```

Abb. 4: Die Regeln aus dem Wissensextraktions-Algorithmus.

Um zu überprüfen, ob der so angepasste genetische Algorithmus besser ist als der ursprüngliche, wurde wieder ein festgelegtes Problem mit immer den gleichen 55 Aufträgen 20 Mal von den beiden Algorithmen über 1000 Iterationen gelöst. Dabei wurden für die Individuen pro Liste zwei Längen getestet (1 und 20), wobei in beiden Fällen pro Zyklus ein neues Lösungsindividuum erzeugt wurde. Zum Vergleich der beiden Algorithmen wird überprüft, wie sich die durchschnittliche Zeit zur Herstellung der Aufträge mit jeder Iteration verändert. Dieses ist in Abb. 5 gezeigt.

Es ist zu erkennen, dass in beiden Fällen der Graph bei dem angepassten genetischen Algorithmus (grün) zu Beginn schneller absinkt. Mit steigender Anzahl der Iterationen wird dann aber die Lösung des ursprünglichen genetischen Algorithmus besser, d.h. der rote Graph schneidet den Grünen (jeweils in den Ausschnitten gezeigt) und findet letztendlich eine bessere Lösung. Die Erwartung war, dass der angepasste genetische Algorithmus schneller ist und die bessere Lösung findet.

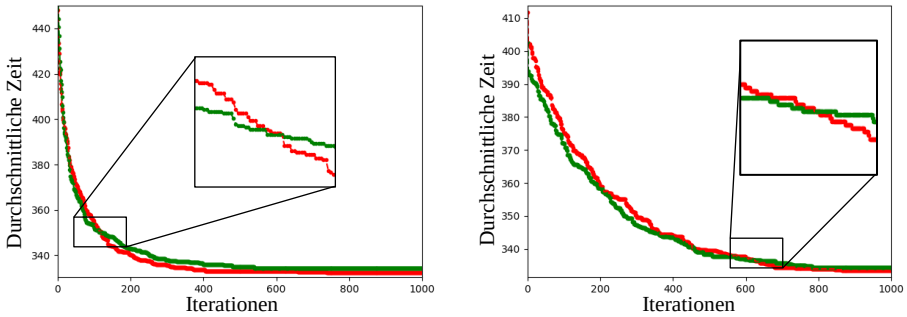


Abb. 5: Graphen des genetischen Algorithmus mit (grün) und ohne (rot) Intuition. Rechts: Individuenanzahl 1; links: Individuenanzahl 20.

Der angepasste genetische Algorithmus kann nur so gut sein wie die durch Intuition abgeleiteten Regeln, die von den Spieldaten ausgehen. Um diese zu überprüfen, wurden noch einmal die erzeugten Datensätze der Testspieler betrachtet. Für jedes Problem, das von einem Menschen gelöst worden ist, wurde überprüft, ob dies die beste Lösung ist. Dafür wurde jedes dieser Probleme in den ursprünglichen genetischen Algorithmus gegeben. Dann wurde die Lösung mit der Lösung vergleichen, welche vom Menschen erspielt wurde. Dabei hat sich in der Tat gezeigt, dass der Mensch in knapp 30% der Fälle eine nicht so gute Lösung wie der genetische Algorithmus gefunden hat, d.h. das eingesetzte menschliche Wissen ist nicht perfekt. Der damit angepasste genetische Algorithmus findet aber trotzdem – gerade zu Beginn – schneller eine gute Lösung, aber letztendlich nicht die optimale. Dies beschreibt sehr genau, was ein Mensch mittels Intuition leistet: Diese weist den Menschen schnell in die Richtung für eine sehr gute Lösung, die aber nicht unbedingt die optimale sein muss.

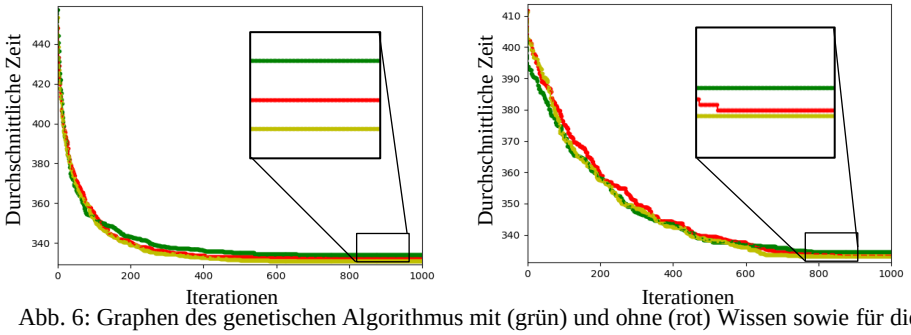


Abb. 6: Graphen des genetischen Algorithmus mit (grün) und ohne (rot) Wissen sowie für die kombinierte Form (gelb). Rechts: Individuenanzahl 1; links: Individuenanzahl 20.

Um nun die Vorteile der menschlichen Intuition mit denen des genetischen Algorithmus zu kombinieren, wurde die Idee verfolgt, ab einer bestimmten Anzahl von Iterationen auf den genetischen Algorithmus ohne Wissen umzuschalten. Gemäß den Graphen in Abb. 5 ist der Punkt, wo der ursprüngliche Algorithmus den angepassten überholt, bei ca. 110

Iterationen (1 Individuum in der Liste) bzw. 605 Iterationen (20 Individuen in der Liste). Diese wurden als „Umschaltpunkte“ festgelegt und der so angepasste genetische Algorithmus wie zuvor auf das Problem mit 55 Aufträgen angewendet (gelbe Kurve in Abb. 6). Hier wurde das erwartete Ergebnis erhalten: die gelbe Kurve sinkt zu Beginn vergleichbar schnell ab wie die Grüne, wird dann aber nicht von der Roten überholt, sondern kommt auf den durchschnittlich besten Wert.

## 8 Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, Heuristiken unter Einbeziehung von intuitiven Strategien zu verbessern, die man aus dem Vorgehen von Menschen bei der Problemlösung ableitet. Hierzu wurde als Basis ein genetischer Algorithmus eingesetzt, der Job-Shop-Probleme löst. Um an die menschlichen Strategien zu kommen, wurde ein Computerspiel aus dem Zeitmanagement-Genre entwickelt, das einem Job-Shop-Problem entspricht (Gamification). Dies wurde von den Menschen gespielt, wobei die Vorgehensweisen der einzelnen spielenden Personen aufgezeichnet wurde.

Diese gesammelten Daten über die menschliche Intuition wurden auf zwei Arten algorithmisch nutzbar gemacht. Zum einen mit Hilfe des CBR-Algorithmus, der mit dem genetischen Algorithmus verglichen wurde. Dabei hat sich gezeigt, dass die menschliche Intuition sehr schnell auf eine sehr gute Lösung kommt, aber nicht unbedingt die beste Lösung findet. Zum anderen wurde der genetische Algorithmus unter Einbeziehung der menschlichen Strategien basierend auf Intuition verbessert. Dafür wurden aus den menschlichen Daten Regeln extrahiert und mit diesen Regeln wurde dann der Mutationsschritt des genetischen Algorithmus abgewandelt. Dieses hat den genetischen Algorithmus dahingehend verbessert, dass bereits nach wenigen Iterationen schneller eine gute Lösung gefunden wird. Allerdings holt der ursprüngliche genetische Algorithmus mit der Zeit auf und kommt letztendlich zu einer besseren Lösung. Dieses spiegelt wieder, dass menschliche Intuition schnell zu einer guten Lösung führen kann, aber die Rechenleistung eines Computers – bei genügender Zeit – am Ende eine bessere Lösung findet.

Um das Beste beider „Welten“ zu kombinieren, wurde dann der genetische Algorithmus so programmiert, dass die Mutationen zunächst basierend auf den menschlichen Regeln erfolgen, gegen Ende aber wieder – wie ursprünglich – rein zufällig erfolgen. Dadurch ergab sich ein Algorithmus, der zu Beginn schneller zu einer guten Lösung kommt und diese letztendlich noch weiter verbessert. Bei der Kombination von CBR-Algorithmus und genetischem Algorithmus liegen erste Ergebnisse vor, die auf ein ähnliches Ergebnis kommen.

Somit ist es grundsätzlich möglich, eine Heuristik unter Einbeziehung von Strategien basierend auf menschlicher Intuition zu verbessern. Hierbei ist die Qualität der aus menschlicher Intuition generierten Daten sehr wichtig. Daher könnte es möglich sein, den Algorithmus noch weiter zu verbessern, wenn man noch mehr menschliches Wissen

höherer Qualität ansammelt. Man könnte nun auch untersuchen, ob eine Verbesserung basierend auf Strategien menschlicher Intuition ebenfalls für andere Arten von Problemen und/oder Heuristiken funktioniert.

# 9 Quellen

[AK17] Apeldoorn, D.; Kern-Isberner, G.: Towards an Understanding of What is Learned: Extracting Multi-Abstraction-Level Knowledge from Learning Agents. In: Rus, V., Markov, Z. (Hrsg.) Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference. AAAI Press, Palo Alto, California (2017)

[AK16] Apeldoorn, D.; Kern-Isberner, G.: When Should Learning Agents Switch to Explicit Knowledge? In: Benzmüller, C., Sutcliffe, G., Rojas, R. (Hrsg.) GCAI 2016. 2nd Global Conference on Artificial Intelligence. EPiC Series in Computing, vol. 41, pp. 174–186. EasyChair Publications (2016)

[BK08] Beierle, C.; Kern-Isberner, G.: Methoden wissensbasierter Systeme: Grundlagen, Algorithmen, Anwendungen. Vieweg+Teubner, Wiesbaden (2008)

[FM93] Forrest, S.; Mitchell, M.: Relative Building-Block Fitness and the Building Block Hypothesis. Foundations of Genetic Algorithms Volume 2, 109–126 (1993)

[HKS14] Hamari, J.; Koivisto, J.; Sarsa, H.: Does Gamification Work? – A Literature Review of Empirical Studies on Gamification. In: 2014 47th Hawaii International Conference on System Science. IEEE, New York, S. 3025-3034, 2014.

[He03] Henning, A.: Praktische Job-Shop Scheduling-Probleme. Jena 2003.

[JBS17] Joseph, K.; J. Banks, C.; A. Shah, J.: Collaborative Planning with Encoding of Users' High-level Strategies. Thirty-First AAAI Conference of Artifical Intelligence. Association for the Advancement of Artificial Intelligence, Palo Alto, 2017.

[KUG18] Kyritsis, M.; Ud Din, S.; Gulliver, S.: Quotient Algorithm: A Simple Human-Inspired Heuristic for Addressing the Travelling Salesperson Problem. 2018 Fifth HCT Information Technology Trends (ITT), Dubai, United Arab Emirates, 2018, pp. 47-49

[MHZ10] Mönch, L. unter Mitarbeit von Habla, C., Zimmermann, J.: Entscheidungsmethoden in Unternehmensweiten Softwaresystemen. FernUniversität in Hagen, Hagen, 2010.

[Re17] Reichert, R.: Scheduling Algorithmen: Einleitung in das Job-Shop Scheduling. SwissEduc (2017). Webseite (aufgerufen am 30.11.18): https://www.swisseduc.ch/informatik/theoretische_informatik/scheduling/einleitung.html

[Wu02] Wu, Y.; Huang, W.; Lau, S.; Wong, C.; Young, G.: An effective quasi-human based heuristic for solving the rectangle packing problem. European Journal of Operational Research, Volume 141, Issue 2, pp. 341-358, 2002.