Diplomarbeit

Validierung zweier Optimierungsverfahren zur taktzeitoptimierten Aufgabenzuordnung am Beispiel eines Mensch-Roboter-Arbeitssystems

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines

Diplom-Ingenieurs

unter der Leitung von

Univ.-Prof. Dr.-Ing. Dipl.-Ing. Sebastian Schlund
(E330 Institut für Managementwissenschaften, Bereich: Human Centered Cyber Physical Production and Assembly Systems)

Maximilian Papa, MSc
(Fraunhofer Austria Research GmbH)

eingereicht an der Technischen Universität Wien

Fakultät für Maschinenwesen und Betriebswissenschaften

von

Andra Vartolomei

[redacted]

Wien, im Dezember 2022

_____
Andra Vartolomei

Ich habe zur Kenntnis genommen, dass ich zur Drucklegung meiner Arbeit unter der Bezeichnung

Diplomarbeit

nur mit Bewilligung der Prüfungskommission berechtigt bin.

Ich erkläre weiters an Eides statt, dass ich meine Diplomarbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen selbstständig ausgeführt habe und alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, genannt habe.

Weiters erkläre ich, dass ich dieses Diplomarbeitsthema bisher weder im In- noch Ausland (einer Beurteilerin/einem Beurteiler zur Begutachtung) in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Wien, im Dezember 2022

_____

Andra Vartolomei

Danksagung

Ich danke an dieser Stelle vorerst all jenen, die an mich geglaubt haben und mir die Kraft gegeben haben, selbst an mich zu glauben und weiterzumachen, wenn auch der Weg zu dieser Arbeit und dem Studienabschluss manchmal sehr lang und schwer schien.

Meinen Betreuern Titanilla Komenda-Haring, Maximilian Papa und Sebastian Schlund möchte ich für die aufgebrachte Geduld und die unterstützenden Worte danken.

Zusätzlich will ich meiner großen Familie – jener, in die ich geboren bin, der, die ich geheiratet habe und der, die ich selbst gründen durfte – danken, dass sie mir alle immer die Unterstützung, den Raum, die Kraft und die Unermüdlichkeit entgegenbringen konnten, um diesen Abschnitt endlich als abgeschlossen bezeichnen zu können.

Kurzfassung

Bei der Auslegung und im laufenden Betrieb von Mensch-Roboter-Arbeitssystemen in der Produktion treffen Planerinnen und Planer auf Herausforderungen aus diversen Bereichen. Diese Arbeit befasst sich mit der taktzeitoptimierten Aufgabenzuordnung in einem solchen System. In einem Arbeitsumfeld, in dem Ressourcen unerwartet ausfallen, der Fertigstellungsdruck jedoch weiterhin bestehen bleibt, bedarf es eines Ressourcenzuweisungswerkzeugs, welches in kürzester Zeit eine neue Aufgabenzuweisung ermöglicht.

Ein solches Werkzeug wird mit dieser Arbeit, ausgehend von einer umfangreichen systematischen Literaturrecherche zum aktuellen Stand der Technik, entworfen. Basierend auf den Erkenntnissen ausgewählter Arbeiten wird eine passende mathematische Formulierung vorgestellt, welche mithilfe von MATLAB sowohl exakt als auch metaheuristisch implementiert wird.

Zum Abschluss der Arbeit werden die implementierten Algorithmen anhand eines Beispiels getestet und die erhaltenen Ergebnisse verglichen, um der Benutzerin und dem Benutzer die jeweiligen Einsatzgebiete, aber auch eventuelle Einschränkungen der Methoden zu erläutern.

Abstract

During the design as well as the ongoing operation of human-robot-work systems in manufacturing, planners encounter challenges in diverse fields. This paper deals with the cycle-time optimized task allocation in such a system. In a work environment where resources unexpectedly fail but completion pressure remains, there is a need for a resource allocation tool that can reassign tasks in the shortest possible time.

Such a tool is designed in this thesis, based on an extensive systematic literature review of the current state of the art. Based on the findings of selected works, a suitable mathematical formulation is presented, which is implemented exactly as well as metaheuristically using MATLAB.

Finally, the implemented algorithms are tested by means of an example and the obtained results are compared in order to show the user the respective fields of application as well as possible limitations of the methods.

Inhaltsverzeichnis

1 Einleitung

1.1 Eingrenzung des Themas

Die Zusammenarbeit von Mensch und Roboter am selben Arbeitsplatz ist mit den schnellen technischen Fortschritten ein Bereich, welcher viel Potential birgt und entsprechend intensiv und vielfältig analysiert wird. Von Ergonomie (Faber, et al., 2015) und Arbeitssicherheit (Behrens, 2018) bis hin zur Ermittlung der optimalen Bewegungsabfolgen (Müller, et al., 2019) und zur psychologischen Akzeptanz der Maschine als Arbeitskollegen (Glück, 2022) wird das System laufend in wissenschaftlichen Arbeiten analysiert.

Ein Bereich mit Optimierungspotential, welcher in dieser Arbeit im Fokus liegt, ist die Aufgabenzuweisung in Mensch-Roboter-Arbeitssystemen. Die Kollaboration der Ressourcen Mensch und Roboter am selben Arbeitsplatz ist von einer Vielzahl an Aspekten geprägt, von denen in dieser Arbeit die Faktoren Kosten und Zeit im Fokus stehen.

Bei der Gestaltung eines gemeinsamen Arbeitsplatzes für Mensch und Roboter kommen die Faktoren Kosten und Zeit zum Einsatz, wenn es um die Anschaffung, Auslegung und den Betrieb dieser Arbeitsstation geht. Bei Robotern sind die Anschaffungs- und Instandhaltungskosten zu berücksichtigen, während ein Mensch entsprechende monatliche Gehaltskosten verursacht. Bei der Auslegung des Arbeitsplatzes müssen den verfügbaren Menschen beziehungsweise Robotern bestimmte Arbeitsschritte zugewiesen werden, wobei es einen Unterschied macht, ob ein Mensch oder Roboter die Tätigkeit durchführt, da andere Bearbeitungszeiten einzuplanen sind. Wie zu erwarten, haben sowohl Mensch als auch Roboter verschiedene Stärken in der Bearbeitung unterschiedlicher Aufgaben am Arbeitsplatz (Glück, 2022) und durch eine geschickte Kombination können die Gesamtzeit minimiert und die Bearbeitungskosten reduziert werden.

Genau diese optimierte Kombination wird im Rahmen dieser Arbeit mithilfe ausgewählter mathematischer Formulierungen und entsprechender Programmierung in zwei Algorithmen ermittelt, wobei angestrebt wird, dem Benutzer der Anwendung so viele Freiheitsgrade wie möglich zu überlassen, sodass unterschiedliche Arbeitsplätze analysiert werden können.

Um ein Arbeitssystem bestehend aus Mensch und Roboter hinsichtlich der Taktzeit und Bearbeitungskosten beurteilen und optimieren zu können, wird zuerst der aktuelle Wissensstand untersucht und anschließend eine entsprechende mathematische Formulierung aufgestellt. Die Komponenten Zeit und Kosten sind hier im Fokus mit dem Ziel, die Aufgaben so zuzuweisen, dass sowohl die Gesamtbearbeitungszeit als auch die -kosten minimiert werden.

Da Mensch und Roboter ganzzahlige Einheiten darstellen, muss diese Einschränkung bei der Umsetzung entsprechend mitberücksichtigt werden. Diese Restriktion zu ganzzahligen Werten erschwert die Ermittlung einer optimalen Lösung erheblich. Zur Auswertung der Zielfunktion werden dementsprechend zwei Optimierungsverfahren herangezogen, da mit steigender Anzahl an Variablen auch die Kombinationsmöglichkeiten und die Berechnungszeit rasant zunehmen. Aus diesem Grund findet im Rahmen dieser Arbeit ein Wechsel zwischen den gewählten Optimierungsverfahren statt – sobald das exakte Optimierungsverfahren für eine schnelle Entscheidung eine zu lange Rechenzeit benötigt, wird mithilfe des metaheuristischen Optimierungsverfahrens eine nahezu optimale Lösung derselben Zielfunktion ermittelt.

1.2 Grundlegende Fragestellungen

Um die Ausrichtung der vorliegenden Arbeit zu verdeutlichen, werden folgende Forschungsfragen im Rahmen dieser Arbeit beantwortet:

1. Wie lautet eine Zielfunktion zur taktzeitoptimierten Aufgabenzuordnung in Mensch-Roboter-Arbeitssystemen?
2. Bis zu welcher Ressourcen- und Aufgabenanzahl kann zur taktzeitoptimierten Planung noch ein exaktes Optimierungsverfahren angewendet werden, das ein globales Minimum ermittelt?
3. Was sind in diesem Zusammenhang Entscheidungskriterien, um ein lokales Minimum als beste Lösung zu akzeptieren?

1.3 Forschungsziele

Der Schwerpunkt dieser Arbeit liegt auf der Bestimmung einer Zielfunktion und ihrer Nebenbedingungen zur taktzeitoptimierten Aufgabenzuordnung in einer Mensch-Roboter-Kollaboration. Unter Anwendung dieser Zielfunktion wird abhängig von der Anzahl der zu planenden Arbeitsschritte beziehungsweise der verfügbaren Ressourcen mit Hilfe eines der gewählten Optimierungsverfahren ein globales oder lokales Minimum für die Aufgabenzuordnung an die verfügbaren Ressourcen identifiziert. Um die Zielfunktion zu erstellen, wird eine umfassende Literaturrecherche durchgeführt, im Rahmen derer der aktuelle Wissensstand hinsichtlich der Optimierungsmöglichkeiten dargestellt wird. Die Zielfunktionen der ausgewählten wissenschaftlichen Arbeiten werden analysiert und als Grundlage herangezogen, um die gewünschte Zielfunktion und deren Nebenbedingungen für diese Diplomarbeit zu formulieren.

Im Praxisteil dieser Arbeit wird die Zielfunktion in den zwei ausgewählten Optimierungsverfahren implementiert und getestet, um festzustellen, für welchen Umfang an Arbeitsschritten beziehungsweise Ressourcen ein globales oder lokales

Optimum zufriedenstellend ist, beziehungsweise welcher der Algorithmen dementsprechend zur Anwendung kommen sollte.

Konkrete Ziele dieser Diplomarbeit sind:

- Theoretische Darstellung des Themenbereichs und Erläuterung der in dieser Arbeit verwendeten Begriffe.
- Ermittlung des aktuellen Standes der Technik zu den ausgewählten Optimierungsverfahren im Bereich der Taktzeitoptimierung und Betrachtung der hier eingesetzten Zielfunktionen mithilfe einer vergleichenden systematischen Literaturrecherche.
- Entwicklung einer Zielfunktion und entsprechender Nebenbedingungen unter Berücksichtigung der hierzu dargestellten theoretischen Grundlagen und der durchgeführten Literaturrecherche.
- Implementierung der entwickelten Zielfunktion im ausgewählten exakten und metaheuristischen Optimierungsverfahren.
- Testung der entwickelten Algorithmen mit einer steigenden Anzahl an Arbeitsschritten. Die erzielten Ergebnisse der Algorithmen werden anschließend miteinander verglichen.

1.4 Aufbau der Arbeit

In diesem ersten Kapitel der Arbeit werden das zu untersuchende Thema beschrieben und die darauf aufbauenden Forschungsfragen und -ziele festgehalten. In Kapitel 2 ist der wissenschaftliche Rahmen des *Operations Research* schrittweise detailliert dargestellt und die Zuordnung der vorliegenden Arbeit im Rahmen der Theorie wird durchgeführt. Im Kapitel 3 werden das Vorgehen der systematischen Literaturrecherche und die daraus resultierenden Ergebnisse präsentiert. Das 4. Kapitel stellt die erarbeitete mathematische Formulierung zur Lösung der Aufgabe dar. In Kapitel 5 wird die Implementierung dieser mathematischen Aufgabe mithilfe der ausgewählten Algorithmen vorgestellt. Im Kapitel 6 werden die Ergebnisse der Algorithmen anhand eines Fallbeispiels präsentiert und im Kapitel 7 werden weitere Schritte zur Entwicklung vorgeschlagen.

2 Theoretische Grundlagen

In den nachfolgenden Abschnitten wird auf das Fachgebiet, dem das Thema dieser Arbeit zuzuordnen ist, schrittweise eingegangen, sodass ausgehend vom Allgemeinen zum vorliegenden Fall ein Überblick geschaffen wird.

2.1 Operations Research

Gerdts und Lempio definieren Operations Research wie folgt: *„Operations Research befasst sich mit der Modellierung der qualitativen und quantitativen Analyse und der algorithmischen Lösung von Entscheidungsproblemen. Hauptanwendungsgebiete sind die Analyse und Optimierung vernetzter Systeme in Wirtschaftsbetrieben, in der Städte- und Verkehrsplanung, in der Volkswirtschaft und in der Technik. Daher ist Operations Research eine ausgezeichnete Motivationsquelle für die mathematische Optimierung, gilt es doch, vor jeder Entscheidung mögliche Entscheidungsalternativen abzuwägen, zu bewerten und die bestmögliche auszuwählen.“* (2011, p. 1)

Singla (vgl. 2021) schreibt, es stehen für das Operations Research diverse Definitionen zur Verfügung, denen als Gemeinsamkeit abgewonnen werden kann, dass es sich um eine wissenschaftliche Methode handelt, welche zur Entscheidungsfindung quantifizierbare Werkzeuge zur Verfügung stellt. Die Quintessenz ist, dass es sich einerseits um eine wissenschaftliche Methode handelt, wodurch impliziert wird, dass ein systematisches Verfahren zur Lösung angewendet wird und andererseits, dass mathematisch quantifizierbare Werkzeuge zum Einsatz kommen, welche bei ähnlichen Problemen erneut eingesetzt werden können, um praktikable Ergebnisse zu erhalten.

2.1.1 Anwendungsgebiete

Die Anwendungsmöglichkeiten sind vielfach, Koop und Moock (vgl. 2018) begründen dies damit, dass mithilfe mathematischer Darstellungen der Realität in den meisten Bereichen Probleme veranschaulicht, analysiert und im Anschluss optimiert werden können. Somit entstehen strukturelle Ähnlichkeiten trotz unterschiedlicher Anwendungsbereiche, da diverse Fragestellungen in eine standardisierte Form gebracht werden (vgl. Kathöfer & Müller-Funk, 2017).

Aufgabenbereiche, welche in Koop & Moock (vgl. 2018) oder Kathöfer und Müller-Funk (vgl. 2017) erwähnt werden, sind:

- Zuordnungsprobleme,
- Produktionsplanung (Scheduling),
- Mischprobleme,
- Transportprobleme,

- Routenplanung,
- Maschinenbelegungsplanung,
- Investitionsplanungsprobleme.

Eine Nebenerscheinung der großen Vielfalt an Anwendungsmöglichkeiten des Operations Research ist, dass eine klare Abgrenzung, welche Bereiche und Aufgaben dem Operations Research zugeschrieben werden können und welche nicht, schwer möglich ist. Kathöfer und Müller-Funk (vgl. 2017) führen weiter aus, dass diese Schwierigkeiten sowohl inhaltlich als auch methodisch bestehen. Die Überschneidungen *sind „zu Teildisziplinen der Betriebswirtschaftslehre wie Produktion und Logistik (oder – zeitgemäßer – dem Operations Management) ebenso fließend wie zu Formalwissenschaften wie der Wahrscheinlichkeitstheorie und Statistik oder konvexen Analysis“* (Kathöfer & Müller-Funk, 2017, p. 12).

Im Rahmen dieser Arbeit wird im Bereich der Produktion die Ressourcenzuweisung mathematisch modelliert und anschließend optimiert. Dies stellt entsprechend ein Entscheidungsproblem dar, welches mithilfe eines mathematischen Modells erfasst werden kann, um anhand gezielt gewählter Algorithmen gelöst zu werden und somit die – den Kriterien entsprechend – bestmögliche Lösungsvariante auszuwählen. Im Bereich des Operations Research wird diese Aufgabe den Produktionsplanungsproblemen zugewiesen. Im englischsprachigen Raum ist das Produktionsplanungsproblem als *Production Scheduling* oder auch in der abgekürzten Version als *Scheduling* bekannt und wird im Abschnitt 2.2 vertieft.

2.1.2 Planungsprozess im Operations Research

Operations Research wird charakterisiert durch eine sowohl logische als auch systematische Herangehensweise, um eine objektive Basis für die Entscheidungsfindung darzustellen, indem (vgl. Kathöfer & Müller-Funk, 2017):

- das Problem formuliert und formalisiert wird,
- die notwendigen Verfahren durchgeführt werden und zum Schluss
- die Ergebnisse validiert werden.

Der Ablauf ist schematisch in Abbildung 1 dargestellt. Ausgangspunkt ist eine reale Situation, welche im ersten Schritt mitsamt ihrem Umfeld beobachtet wird. Aus der betrachteten Ist-Situation wird die zu lösende Aufgabe definiert, indem die laut Göllmann et al. (vgl. 2017) benötigten Bausteine zur Lösung eines Optimierungsproblems formuliert werden:

- Optimierungsziel: zu optimierende Ergebnisgrößen,
- Optimierungsvariablen: Eingangsgrößen, welche variiert werden,
- Nebenbedingungen: Restriktionen der Eingangsgrößen.

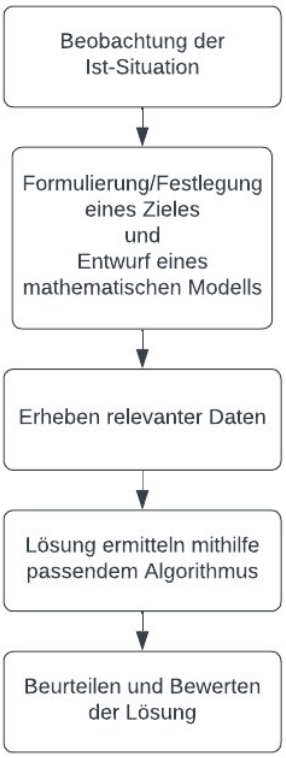

Abbildung 1: Prozessdarstellung im Operations Research, angelehnt an (Koop & Moock, 2018)

Es entsteht ein Modell der Ist-Situation, indem die Optimierungsgrößen in einer Zielfunktion verknüpft werden, welche es gilt, zu minimieren oder maximieren, um die optimalen Parameter des Systems zu finden, während gleichzeitig die definierten Nebenbedingungen eingehalten werden (vgl. Göllmann, et al., 2017). Im Rahmen des Operations Research werden hierzu mathematische Gleichungen angewandt, um Systeme, Prozesse oder ganze Umgebungen darzustellen. Es werden hierzu Beziehungen zwischen den Variablen anhand bereits bestehender Modelle formuliert, diese werden angepasst, um den Gegebenheiten zu genügen oder, falls es notwendig ist, können Gleichungen von Grund auf neu erstellt werden.

Um das formulierte Modell zu testen beziehungsweise zu verwenden, bedarf es richtiger Daten, da mit einer kleinen, mangelnden oder fehlerhaften Datenbank auch das beste Modell keine korrekten Ergebnisse erzeugen kann.

Mit den ausgewählten Daten wird unter Anwendung eines entsprechenden Optimierungsalgorithmus eine Lösung für das mathematische Modell ermittelt. Das Modell wird getestet, um eventuelle Einschränkungen oder Fehlverhalten zu identifizieren. Falls Fehler auftreten, müssen Anpassungen am Modell vorgenommen oder ein anderer Algorithmus gewählt werden.

Am Ende dieses Schrittes steht allenfalls ein passendes Modell beziehungsweise dessen Formulierung als Algorithmus fest. Abschließend wird die Lösung bewertet.

Operations Research kann auch als Anwendung mathematischer und statistischer Modelle auf Probleme der Entscheidungsfindung betrachtet werden (Kandiller, 2007). Entsprechend kann bezüglich der Typologie der Rechenmodelle auf unterschiedliche mathematische Strukturen zurückgegriffen werden. Von zentraler Bedeutung im Operations Research sind beispielsweise das *Lineare*, *Nichtlineare* und *Dynamische Programmieren*.

Das *Lineare Programmieren*, auch *Lineare Optimierung* genannt, stellt ein wichtiges Teilgebiet dar, da hiermit eine Vielzahl an praktischen Anwendungen mathematisch formuliert werden kann. Das in dieser Arbeit betrachtete Modell wird mithilfe der Sonderform der *Ganzzahligen Linearen Optimierung* formuliert. Wie der Name schon besagt, sind sowohl die Zielfunktion als auch ihre Nebenbedingungen linear. Im Sonderfall der *Ganzzahligen Linearen Optimierung* (englisch, *Integer Linear Program*) besteht die zusätzliche Bedingung, dass die Variablen ganzzahlig sind (vgl. Pinedo, 2016). Dieser Ansatz wird in der vorliegenden Arbeit bei der Formulierung des mathematischen Modells angewandt, da die Variablen, die gesucht sind, ganzzahlig sind und die Ressourcenzuweisung linearen Gesetzen unterliegt.

2.2 Scheduling

Wie bereits in 2.1.1 erwähnt, wird die vorliegende Arbeit dem Aufgabenbereich der *Produktionsplanung*, englischsprachig *Scheduling*, zugeordnet.

Scheduling ist ein Entscheidungsprozess, der in vielen Produktions- und Dienstleistungsbereichen regelmäßig zur Anwendung kommt und sich sowohl mit der Zuteilung von Ressourcen an Aufgaben als auch der Terminplanung dieser befasst, während eines oder mehrere Ziele verfolgt werden, (vgl. Pinedo, 2016).

Laut Pinedo (vgl. 2016) wird *Scheduling* nicht strikt nur im Bereich der Produktionsplanung betrieben, sondern es werden auch Zeitpläne eines Bauunternehmens, Abflug- und Landepläne eines Flughafens oder die Planung von Aufgaben in einem Zentralprozessor eruiert.

Die Produktionsplanung in der Fertigung stellt keine eigenständige Sparte dar, sondern sie befindet sich in einer kontinuierlichen Wechselwirkung mit anderen Fachbereichen. Bestellungen, welche vom Unternehmen entgegengenommen werden, müssen in firmeninternen Aufträgen mit Fertigstellungszeitpunkten erfasst werden. Die entsprechenden Ressourcen werden hierfür eingeplant und die Abarbeitungsreihenfolge durch diese festgelegt. Es besteht jedoch die Wahrscheinlichkeit, dass die vorgesehenen Ressourcen für dringlichere Aufgaben umdisponiert werden müssen, dass Maschinen defekt werden oder bei der Bearbeitung Verzögerungen auftreten. Die Produktionsplanung wird jedoch nicht nur vom *shop floor* beeinflusst, sondern auch von anderen Bereichen des Unternehmens. Die mittel- und langfristige *Programmplanung* des Unternehmens spielt ebenfalls eine

wesentliche Rolle, da eine langfristige Auslastung der Bereiche im Fokus ist (vgl. Pinedo, 2016).

Lopez und Roubellat (vgl. 2008) erläutern, dass durch den hohen Marktdruck die Unternehmensleistung in einer *technologischen* und einer *organisatorischen* Dimension ausgebaut wird.

Die *technologische Dimension* hat als Ziel, die Anforderungen an Qualität und den Wunsch nach niedrigeren Betriebskosten für diese Produkte zu erfüllen, wobei der schnelle technologische Fortschritt und ein erhöhter Individualisierungsgrad dazu führen, dass Unternehmen sich von Massenproduktion entfernen und flexible beziehungsweise schnell anpassbare Produktionssysteme einführen.

Die *organisatorische Dimension* strebt eine Verbesserung sowohl hinsichtlich der Reaktionsfähigkeit auf Schwankungen als auch der Bearbeitungs- und Lieferzeiten an und spielt eine immer wichtigere Rolle, da immer kürzere Reaktionszeiten seitens der Unternehmen erwartet werden. Um dies zu erreichen, werden unterschiedliche Werkzeuge angewandt, in diesem Zusammenhang ist *Scheduling* – also die Produktionsplanung – von Interesse. Mit ihr werden Menschen auch als technische Ressourcen eingeplant, sodass Kundenwünsche, die im Produktionsprogramm des Unternehmens festgehalten sind, erfüllt werden.

In Anbetracht des dynamischen Marktes ist die Produktionsplanung eine immer komplexere Aufgabe, da eine Vielzahl an Aufträgen eingeplant werden muss, welche von begrenzt verfügbaren Ressourcen (sei es Mensch oder Roboter) durchgeführt werden können.

An dieser Stelle wird die Unterscheidung bei der Art der darzustellenden Modelle erläutert. Im Scheduling kann – laut Pinedo (vgl. 2016) – zwischen *deterministischen* und *stochastischen* Modellen unterschieden werden.

In den *deterministischen* Scheduling Modellen gilt, dass eine endliche Anzahl an Aufgaben eingeplant wird, wobei eine oder mehrere Zielfunktionen ermittelt werden. In *stochastischen* Scheduling-Modellen wird ebenfalls eine bestimmte Anzahl an Tätigkeiten geplant, wobei aber weitere Details wie die Bearbeitungszeit, oder auch Start- und Fertigstellungszeiten nicht unbedingt bekannt sind, sondern erst nach Abschluss genaue Werte dafür feststehen.

In dieser Arbeit wird in weiterer Folge auf die Klasse *deterministischer* Scheduling-Modelle eingegangen.

2.2.1 Resource Constrained Project Scheduling Problem (RCPSP)

Scheduling-Probleme sind in den Fünfzigerjahren in den Fokus der Wissenschaftlerinnen und Wissenschaftler geraten. Motivator hierzu waren die

Anwendungsgebiete der *Projektplanung* und *Maschinenplanung*. Im Rahmen der *Maschinenplanung* wurden unterschiedliche Möglichkeiten analysiert, abhängig von der Art und Anzahl der Maschinen und den Jobeigenschaften. Im Rahmen der *Projektplanung* wurden Vorgängerbeziehungen in Betracht gezogen. Einschränkungen hinsichtlich der Verfügbarkeit der Ressourcen wurden erst in späterer Folge berücksichtigt und haben sich zum Bereich der *Resource Constrained Project Scheduling Problems (RCPSP)* entwickelt. Das *RCPSP* ist mittlerweile ein Grundbestandteil der komplexen *Scheduling*-Probleme und einen Spezialfall in diesem Bereich stellen Probleme der *Maschinenplanung* dar (Brucker & Knust, 2012).

2.2.2 Terminologie im Scheduling

In diesem Abschnitt werden die grundlegen Begriffe erläutert. Diese sind in der *Produktionsplanung* beziehungsweise dem *RCPSP* (und auch in der *Maschinenplanung*) von Bedeutung.

Wie der nachfolgenden Literaturrecherche zu entnehmen ist, handelt es sich bei den angeführten Begriffen und den entsprechenden Bereichen um überlappende Themengebiete, welche – abhängig von den Autoren und Fachgebieten – sich leicht abgewandelter Begriffe bedienen. Im nachfolgenden werden die wichtigsten Begriffe der Literatur der angeführten Bereiche erläutert.

Ressourcen (vgl. Lopez & Roubellat, 2008)

Eine *Ressource* i stellt einen Menschen oder einen Roboter dar, welche zur Ausführung von Arbeitseinheiten zur Verfügung steht. Hierbei ist zu beachten, dass deren Verfügbarkeit zu jedem gegebenen Zeitpunkt begrenzt ist.

Man kann zwischen *erneuerbaren* und *nicht-erneuerbaren* Ressourcen unterscheiden. Erstere stehen bei erneutem Einsatz in derselben Menge zur Verfügung (Mensch, Roboter, Raum, …), wenn auch zu jedem Zeitpunkt nur eine gewisse Menge vorhanden ist. Bei *nicht-erneuerbaren Ressourcen* handelt es sich um Ressourcen wie Rohstoffe oder Geld, deren Verfügbarkeit über die Zeit summiert begrenzt ist.

Jobs (vgl. Lopez & Roubellat, 2008)

Ein *Job* j (eine Tätigkeit beziehungsweise Aufgabe) stellt die grundlegende Arbeitseinheit dar, welche eine *Startzeit* st_{ji} und eine *Fertigstellungszeit* et_{ji} hat. Die Ausführung wird von einer *Bearbeitungszeit* pt_{ji} (es gilt $et_{ji} = st_{ji} + pt_{ji}$) unter dem Einsatz der *Ressource* i beschrieben.

Jobs können *präemptiv* oder *nicht-präemptiv* durchgeführt werden, worunter zu verstehen ist, dass die Bearbeitung eines Jobs unterbrochen oder nicht unterbrochen

werden kann. Ein Vorteil präemptiver Jobs ist, dass durch die Durchführung in Teilaufgaben eine stärkere Auslastung der Ressourcen erzielt werden kann.

Aufgabenstruktur

Es werden n Jobs $J_j, (j = 1, \dots, n)$ von m Ressourcen $R_i, (i = 1, \dots, m)$ bearbeitet. Weitere Eigenschaften der Jobs, welche definiert werden können, sind: notwendige Bearbeitungsschritte zur Fertigstellung, die Bearbeitungszeiten durch die jeweiligen Ressourcen, Gewichtung der Jobs, Start- oder Fertigstellungszeitpunkte oder auch eine Kostenfunktion, welche die Kosten misst, wenn der Job rechtzeitig fertiggestellt wird (vgl. Hammer, et al., 1979).

Wenn ein *Auftrag* eine Reihe an Jobs für die Fertigstellung benötigt, dann bezeichnet das Paar (i, j) den Job j, der von der Ressource i durchgeführt wird.

Vorgängerrelationen

Vorgängerrelationen stellen Bedingungen dar, nach denen bestimmte Jobs abgeschlossen sein müssen, bevor die Bearbeitung eines anderen Jobs gestartet werden kann. Es gibt verschiedene Spezialformen, welche beispielsweise *Ketten* beschreiben, bei denen jeder Job maximal einen Vorgänger und maximal einen Nachfolger hat (Brucker & Knust, 2012).

2.2.3 Einteilung in der Maschinenplanung

Scheduling-Probleme werden mit dem *Triplett* $\alpha \mid \beta \mid \gamma$ beschrieben (vgl. Pinedo, 2016). Diese sind wie folgt zu verstehen.

Maschinenumgebung α (vgl. Pinedo, 2016)

Die *Maschinenumgebung* α enthält einen einzigen Eintrag. Zur Auswahl stehen folgende Möglichkeiten.

Die einfachste Systemkonfiguration und gleichzeitig die Vereinfachung aller anderen Modelle ist das *Einmaschinenmodell [1]*, bei dem eine einzige Ressource zur Verfügung steht. Dieses Modell kann zur Lösung von *Bottleneck*-Produktionsengpässen angewandt werden, indem ein Modell für die Planung der betroffenen Maschine ermittelt wird.

Eine Erweiterung ist das *parallele Maschinenmodell [Pm]*, bei dem i identische Ressourcen zur Verfügung stehen.

Hier kann der Sonderfall differenziert werden, der *parallele Ressourcen, jedoch unterschiedliche Bearbeitungsgeschwindigkeiten [Qm]* aufweist. Mit der Geschwindigkeit v_i der Ressource i kann die Bearbeitungszeit $pt_{ji} = pt_j / v_i$ ermittelt

werden. Wenn alle Ressourcen dieselbe Geschwindigkeit v_i haben, dann handelt es sich um identische parallele Maschinen, wie vorhin beschrieben.

Mit *unabhängigen parallelen Maschinen [Rm]* entsteht eine weitere Generalisierung des vorherigen Falls, da i unterschiedliche Ressourcen zur Verfügung stehen. Jede Ressource i bearbeitet mit der Geschwindigkeit v_{ji} den Job j. Die Bearbeitungszeit kann ermittelt werden mit $pt_{ji} = pt_j/v_{ji}$ (vorausgesetzt, dass alle Jobs von derselben Ressource durchgeführt werden). Wenn wiederum die Geschwindigkeit $v_{ji} = v_i$ unabhängig von den Tätigkeiten ist, dann liegt wieder der vorherige Fall vor.

Shop Scheduling stellt Spezialfälle des *Schedulings* dar, wobei deren Gliederung sich auf die Art der Jobabfolge auf den Maschinen bezieht. Sie wird unterteilt in *flow shop, open shop* und *job shop* (vgl. Blazewicz, et al., 2019).

Bei einem Auftrag im *flow shop [Fm]* wird eine sortierte Abfolge an Jobs durchgeführt. Für jeden Auftrag werden alle Jobs in derselben Reihenfolge von bestimmten in Serie geschalteten Ressourcen durchgeführt (die j-te Job erfolgt auf der i-ten Maschine), womit die Arbeitsgangfolge für jeden Auftrag identisch ist. Meist erfolgt die Bearbeitung der Jobs nach dem *FIFO-Prinzip (First In First Out)*, sodass ein Überholen in der Warteschleife ausgeschlossen ist (vgl. Blazewicz, et al., 2019).

Im Gegensatz zu *flow shop* ist im *open shop [Om]* die Reihenfolge der Jobs auf den Maschinen willkürlich (vgl. Blazewicz, et al., 2019).

Im *job shop [Jm]* sind die Ressourcenreihenfolgen für die einzelnen Jobs vorgegeben, die Abfolge kann jedoch von einem Auftrag zum nächsten unterschiedlich sein (vgl. Blazewicz, et al., 2019).

Die Sonderfälle *flexible flow shop [FFc]* und *flexible job shop [FJc]* sind analog zu flow shop und job shop zu verstehen. Der Unterschied besteht darin, dass statt Jobs Phasen c in Serie geplant werden, in denen gleiche parallel geschaltete Ressourcen verfügbar sind, die alle den Job j durchführen können (der nur von einer Ressource durchgeführt werden muss).

Jobeigenschaften β

In den *Jobeigenschaften* β sind Einzelheiten zu den Bearbeitungsmerkmalen und Einschränkungen hinterlegt. Es können keine, einer oder sogar mehrere Einträge vorhanden sein.

Einige der möglichen Einträge im Parameter β sind:

- *Freigabezeiten* $[r_j]$, welche die möglichen „ab“-Startzeiten der Jobs darstellen
- *Präemption [prmp]*
- *Vorgängerrelation [prec]*

Zielfunktion γ (vgl. Pinedo, 2016)

Im dritten Feld γ ist die zu minimierende *Zielfunktion* hinterlegt, meist als ein einziger Eintrag.

Mögliche reguläre Leistungskennzahlen für die zu minimierende Zielfunktionen sind:

- die gesamte Bearbeitungszeit [$C_{\max} = \max(C_j)$], wodurch der Auslastungsgrad der Maschinen gemessen wird.
- die maximale Verspätung [$L_{\max} = \max(L_j)$].
- die gewichtete Gesamtfertigstellungszeit [$\sum w_j C_j$], die einen Hinweis auf die gesamten Bestandskosten liefert, die durch den Zeitplan verursacht werden.

NP-Schwere

Die Klasse der NP-Aufgaben (NP steht für nichtdeterministisch polynomielle Zeit und bezeichnet eine fundamentale Komplexitätsklasse) wird dadurch charakterisiert, dass für eine (exakte) Lösung mit allen bekannten deterministischen Algorithmen ein exponentieller Rechenaufwand erforderlich ist und es wird vermutet, dass es keine effizienten Algorithmen zur Lösung dieser Klasse gibt (vgl. Garey, et al., 1979).

Dementsprechend stoßen exakte Lösungsverfahren an ihren Grenzen bei der Lösung solcher Aufgaben, sodass sie von heuristischen oder metaheuristischen Verfahren abgelöst werden müssen, um in einem sinnvollen Zeitraum eine Lösung zu ermitteln.

Im Rahmen dieser Arbeit soll genau dieser Wechsel von einem exakten zu einem metaheuristischen Lösungsansatz genauer betrachtet werden, damit eine exakte Lösung der vorliegenden Aufgabe so lang wie möglich angeboten wird.

2.3 Algorithmen

Wie von Kathöfer und Müller-Funk (vgl. 2017) betont wird, stellt die standardisierte Fragestellung im Planungsprozess nicht die Hauptaufgabe des Operations Research dar, sondern die anschließend angewandten Lösungsverfahren.

Dieses Schritt-für-Schritt definierte Vorgehen, welches zeigt, wie eine Aufgabe zu lösen ist – ähnlich einem Kochrezept –, wird als Algorithmus bezeichnet.

Eigenschaften

Ein Algorithmus erfüllt folgende Eigenschaften (vgl. von Rimscha, 2008):

- Er muss *allgemein gültig* sein, sodass er nicht nur eine bestimmte Fragestellung beantwortet, sondern ganz allgemein für gleichartige Aufgaben passt.
- Er muss *ausführbar* sein, indem er eine endliche Anzahl klarer Anweisungen verständlich und in einer sinnvollen Reihenfolge enthält.

- Er muss *endlich* sein und nach einer gewissen Anzahl an Schritten zu einer Lösung kommen, oder – falls es sich um ein Verfahren handelt, welches unendlich lange läuft – es muss ein künstlicher Abbruch geplant werden.

2.3.1 Klassifizierung von Algorithmen (vgl. Nahrstedt, 2018)

Algorithmen können abhängig von der *Maschinenfähigkeit deterministisch* oder *nichtdeterministisch* sein – je nachdem, ob bei gleicher Eingabe immer das gleiche Ergebnis geliefert wird oder nicht. In praktischen Anwendungen werden Algorithmen oft auf Determiniertheit eingeschränkt.

Eine weitere Gliederung kann abhängig von der *Problemstellung* stattfinden, indem zwischen *Entscheidungsalgorithmen* und *Optimierungsalgorithmen* unterschieden wird.

Entscheidungsprobleme bestehen aus Fragestellungen, die entweder mit „ja“ oder „nein“ beantwortet werden können (vgl. Dangelmaier, 2021).

Optimierungsalgorithmen können mit einer Vielfalt an *Methoden* umgesetzt werden, welche sich einteilen lassen in:

- *Exakte* Algorithmen, welche die optimale Lösung liefern. Diese können jedoch schnell ineffizient sein, da sie alle Lösungen der vorliegenden Aufgabe ermitteln, um anschließend die optimale Lösung zu definieren. Somit kommt es bei einer steigenden Anzahl an möglichen Kombinationen zu sehr hohen Rechenzeiten und -leistungen, sodass diese Verfahren schnell verworfen werden.
- *Heuristische* Algorithmen sind Suchverfahren, die spezielle Regelstrategien anwenden, um somit schneller eine Lösung zu finden (vgl. Grimme & Bossek, 2018).
  Deren Nachteil besteht darin, dass sie gezielt auf das behandelte Problem konfiguriert werden und somit fast unmöglich für andere Aufgaben abwandelbar sind (vgl. Jarboui, et al., 2013). Beispiele wichtiger Heuristiken sind *Dekomposition*, *Induktives Vorgehen* oder *Inkrementalanalyse* (vgl. Zimmermann, 2008)
- *Metaheuristische* Algorithmen, (vgl. Jarboui, et al., 2013), welche auf eine große Spannbreite an Optimierungsproblemen angewandt werden können. Im Gegensatz zu heuristischen Algorithmen haben sie den Vorteil, dass sie nicht in lokalen Optima hängen bleiben und somit bessere Chancen haben, ein globales Optimum zu erreichen.
  In den vergangenen Jahren konnten sich folgende Typen etablieren (vgl. Jarboui, et al., 2013):

- *Lokale Suchmethoden* (beispielswese *Simulated Annealing* oder *Tabu Search*), welche ausgehend von einer bestehenden Lösung die Nachbarschaft absuchen, um weitere Lösungen zu finden
  und
- *Evolutionäre Algorithmen* (wie der *Genetische Algorithmus*), welche bei jeder Iteration eine Lösungsauswahl entwickeln.

2.3.2 Ausgewählte Algorithmen

Im Rahmen dieser Arbeit wird die Optimierung eines Ressourcenzuweisungsproblems durchgeführt, einerseits mithilfe eines *exakten* und anschließend eines *metaheuristischen* Algorithmus.

Die Implementierung erfolgt mithilfe der Programmiersprache MATLAB[1], welche zwei Optimierungspakete zur Verfügung stellt: einerseits die *Optimization Toolbox*, welche unter anderem optimale Lösungen für gemischt-ganzzahlige lineare Programme findet und andererseits die *Global Optimization Toolbox*, zum Finden globaler Lösungen von Problemen mit mehreren Minima.

Da es sich in der vorliegenden Aufgabe um eine ganzzahlige Optimierungsaufgabe (ILP, also integer linear programming) handelt, kommen nur entsprechende *Solver* (Sammelbezeichnung für mathematische Computerprogramme, welche Aufgaben numerisch lösen) in Frage.

Aus den bei MATLAB verfügbaren Solvern wird für die exakte Lösung die Lineare Programmierung (*intlinprog)* eingesetzt. Für die metaheuristische Lösung stehen die Solver *genetic algorithm (ga)* oder *surrogate optimization (surrogateopt)* zur Verfügung, siehe Abbildung 2, wobei für die Umsetzung in dieser Arbeit der Genetische Algorithmus zum Einsatz kommt.

Solver	MILP	Solver	MILP
linprog	×	patternsearch	×
intlinprog	✓	ga	✓
quadprog	×	particleswarm	×
coneprog	×	simuannealbnc	×
lsqlin	×	surrogateopt	✓
lsqnonneg	×	gamultiobj	✓
lsqnonlin	×	paretosearch	×
fminunc	×		
fmincon	×		

Abbildung 2: Verfügbare Solver in MATLAB und deren Eignung zur Lösung gemischt-ganzzahliger Optimierungsprobleme (MILP)

[1] https://de.mathworks.com/products/matlab.html

3 Systematische Literaturrecherche

In diesem Kapitel wird die Durchführung der systematischen Literaturrecherche zum Thema Optimierungsverfahren mit dem Schwerpunkt Taktzeitoptimierung beschrieben.

Ziel der Literaturrecherche ist es, unterschiedliche Zielfunktionen zur Optimierung der Taktzeit in Mensch-Roboter-Arbeitssystemen zu identifizieren und aus diesem Wissen heraus eine geeignete Zielfunktion mit entsprechenden Nebenbedingungen für das vorliegende Optimierungsproblem zu entwickeln.

3.1 Vorgehensweise bei der systematischen Literaturrecherche

Das Vorgehen bei der Durchführung einer Literaturrecherche wird in diversen Quellen beschrieben, unter anderem von Brereton et. al. in (2006) oder Kitchenham in (2004), wobei allesamt folgende Grundbestandteile aufweisen: Festlegung eines Forschungsthemas, Formulierung einer oder mehrerer zusammenhängender Forschungsfragen, Entwicklung eines Überprüfungsprotokolls zur Datenfindung, Durchführung der eigentlichen systematischen Literaturrecherche anhand des entwickelten Rechercheprotokolls und schließlich die Dokumentation und Interpretation der Ergebnisse.

Nachfolgend werden die Phasen der systematischen Literaturrecherche, welche dieser Arbeit zugrunde liegen, dargestellt und es werden sowohl die geplanten Maßnahmen als auch deren Umsetzung eruiert.

3.1.1 Phase I – Planen der Literaturrecherche

Schritt 1: Forschungsbereich spezifizieren

Mit der Festlegung des Forschungsthemas wird der Rahmen für die zu beantwortenden Forschungsfragen bestimmt.

Umsetzung: Thema dieser systematischen Literaturrecherche ist die „taktzeitoptimierte Aufgabenzuordnung“.

Schritt 2: Ziel der Literaturrecherche bestimmen

Es wird die Verfassung der Literaturrecherche begründet und ihr Ziel klar definiert.

Umsetzung: Diese Literaturrecherche wird erstellt, um das State-of-the-art der taktzeitoptimierten Aufgabenzuordnung zu ermitteln und anhand der Ergebnisse eine Zielfunktion für die vorliegende Aufgabenstellung zu definieren.

Schritt 3: Forschungsfragen spezifizieren

Ziel einer systematischen Literaturrecherche ist die Beantwortung der hierfür festgelegten Forschungsfragen. Die Forschungsfragen müssen aussagekräftig und präzise formuliert sein, sodass aus ihnen die anschließend benötigten Schlagworte abgeleitet werden können.

Umsetzung: Folgende Forschungsfragen wurden im Rahmen dieser Arbeit definiert:

- Wie lautet eine Zielfunktion zur taktzeitoptimierten Aufgabenzuordnung in einem Mensch-Roboter-Arbeitssystemen?
- Bis zu welcher Ressourcen- und Aufgabenanzahl kann zur taktzeitoptimierten Planung noch ein exaktes Optimierungsverfahren angewendet werden, welches ein globales Minimum ermittelt?
- Was sind in diesem Zusammenhang Entscheidungskriterien, um ein lokales Minimum als beste Lösung zu akzeptieren?

Schritt 4: Entwicklung eines Überprüfungsprotokolls

Zur Durchführung der systematischen Literaturrecherche wird ein Überprüfungsprotokoll definiert, in dessen Rahmen die anzuwendenden Methoden und Verfahren festgelegt sind.
Methoden zur Datenfindung, die zur Anwendung kommen, beziehen sich auf die Suchstrategie, den Suchraum und die Suchbegriffe.
Die Auswahl relevanter beziehungsweise zutreffender wissenschaftlicher Arbeiten erfolgt anhand primärer und semantischer Kriterien.

Umsetzung:

Datenfindung

- *Suchstrategie*: Es wird eine webbasierte systematische Literaturrecherche durchgeführt.
- *Suchraum*: Google Scholar Suchmaschine
- *Enthaltene Schlüsselwörter:* der Stufe 1 bis 3 (siehe Tabelle 1 und Tabelle 2)

Datenauswahl

Primäre Auswahlkriterien

- Veröffentlichungszeitraum: 2015 – 2020
- Sprache: englisch
- Art der Publikation: wissenschaftliche Arbeiten (Journale und Konferenzberichte), Bücher (beziehungsweise ausgewählte Kapitel)
- Verfügbarkeit: nur Online-Ressourcen mit Volltext-Option
- Sortierung: nach Relevanz

Semantische Auswahlkriterien

- Bezug zum Forschungsthema: Ja/Nein (Bewertung während des Screenings)
- Eignung zur Beantwortung der Forschungsfragen: Ja/Nein (Bewertung während des Screenings)

Die festgelegten Schlüsselwörter der Tabelle 1 und Tabelle 2, anhand derer die systematische Literaturrecherche durchgeführt wird, beziehen sich in der Stufe 1 auf den gewünschte Lösungsansatz – die *Lineare ganzzahlige Optimierung* und den *Genetischen Algorithmus* – und auch auf die Hauptanforderung an die Zielfunktion, die Taktzeitoptimierung.

In Stufe 2 und 3 werden in beiden Suchen dieselben Schlüsselwörter zur weiteren Verfeinerung angewandt.

In der zweiten Stufe werden somit die nächsten Kriterien an die Lösungsansätze der wissenschaftlichen Arbeiten jeweils hinzugefügt: die *Ressourcenzuweisung*, die *Aufgabenzuweisung* und das *Scheduling*.

In der dritten und letzten Stufen werden die jeweiligen Ergebnisse um die Suchbegriffe *Mensch Roboter* beziehungsweise *Roboter und Multiagent* verfeinert.

Mit diesem dreistufigen Suchverfahren können wissenschaftliche Arbeiten identifiziert werden, welche die geplanten Lösungsansätze – *Lineare Ganzzahlige Optimierung* und *Genetischen Algorithmus* – einsetzen, um ein Ressourcenzuweisungsproblem in einer Mensch-Roboter-Kollaboration zu lösen.

3.1.2 Phase II – Durchführung der Literaturrecherche

Schritt 5: Durchführung der Literaturrecherche

Umsetzung: Zuerst wird eine webbasierte Suche anhand der primären Auswahlkriterien durchgeführt. Die somit eingegrenzten wissenschaftlichen Arbeiten werden, soweit sie den semantischen Einschlusskriterien entsprechen, heruntergeladen, gelesen und weiter bewertet.

3.1.3 Phase III – Dokumentation und Interpretation der Literaturrecherche

Schritt 6: Dokumentation und Interpretation der Literaturrecherche

Die identifizierten wissenschaftlichen Arbeiten der vorgelagerten Literaturrecherche werden dokumentiert und interpretiert, sodass die Forschungsfragen beantwortet werden können.

Umsetzung: Die Dokumentation und Interpretation der durchgeführten Literaturrecherche wird in den nachfolgenden Unterkapiteln vorgelegt.

3.2 Ergebnisse der systematischen Literaturrecherche

Die Forschungsfragen und auch die Ziele der Arbeit, welche im Rahmen der Planungsphase I (Schritt 1 – 3) definiert werden, sind bereits am Anfang dieser Arbeit in den Abschnitten 1, 1.2 und 1.3 dargestellt.

Phase II, die „Durchführung der systematischen Literaturrecherche“, erfolgt im Rahmen einer webbasierten Suche über die Suchmaschine Google Scholar, welche der allgemeinen Literaturrecherche von wissenschaftlichen Dokumenten dient. Diese Suchmaschine wird bevorzugt, da eine verfeinerte Suche entsprechend der primären Auswahlkriterien möglich ist. Weitere Onlinedatenbanken werden nicht angewandt, da davon auszugehen ist, dass alle relevanten Fachjournale bereits mit Google Scholar zugänglich sind. Die verlinkten Plattformen werden mit einem aktiven Studentenkonto durchsucht und es werden nur jene wissenschaftlichen Arbeiten den primären und semantischen Suchkriterien unterzogen, die als Volltext zum Zeitpunkt der Recherche verfügbar sind.

Die Interpretation der Literaturrecherche, welche in Phase III dargestellt wird, wird in Abschnitt 3.3 präsentiert.

Tabelle 1: Ergebnisse Literaturrecherche ganzzahlig-lineare Optimierung

Schlüsselwörter Stufe 1		Schlüsselwörter Stufe 1 + Schlüsselwörter Stufe 2		Schlüsselwörter Stufe 1 + Schlüsselwörter Stufe 2 + Schlüsselwörter Stufe 3	
integer linear programming & cycle time optimization	50	resource allocation	9	human robot	1
				robot & multi-agent	2
		task allocation	2	human robot	1
				robot & multi-agent	1
		scheduling	48	human robot	2
				robot & multi-agent	3

In Tabelle 1 sind die Ergebnisse der Literaturrecherche zum Lösungsansatz **Ganzzahlige Lineare Optimierung** erfasst.

Mit den Schlüsselwörtern der Stufe 1 wird der Ergebnisraum auf 50 Arbeiten eingeschränkt. Die Stufe 1 zusammen mit den jeweiligen Begriffen der Stufe 2 der Schlüsselwörter ergibt 59 Ergebnisse. Wenn die Schlüsselwörter der Stufe 3 schrittweise hinzugefügt werden, bleiben noch 10 wissenschaftliche Arbeiten über, welche jedoch noch nicht von Duplikaten bereinigt sind.

Es besteht die Vermutung, dass im Rahmen der Ganzzahligen Linearen Optimierung deutlich weniger Arbeiten verfügbar sind, da dieses Verfahren die optimale Lösung sucht, jedoch – wie bereits erwähnt – hierfür Abschläge bei der Ermittlungszeit mit sich bringt.

Tabelle 2: Ergebnisse Literaturrecherche genetischer Algorithmus

Schlüsselwörter Stufe 1		Schlüsselwörter Stufe 1 + Schlüsselwörter Stufe 2		Schlüsselwörter Stufe 1 + Schlüsselwörter Stufe 2 + Schlüsselwörter Stufe 3	
genetic algorithm & cycle time optimization	84	resource allocation	10	human robot	1
				robot & multi-agent	4
		task allocation	4	human robot	3
				robot & multi-agent	3
		scheduling	58	human robot	5
				robot & multi-agent	9

Tabelle 2 enthält die Ergebnisse der Literaturrecherche anhand der primären Auswahlkriterien für den **genetischen Algorithmus**.

Nach Anwendung der primären Auswahlkriterien und der Schlüsselwörter der Stufe 1 sind 84 Suchergebnisse verfügbar. Durch die zusätzliche Anwendung der Schlüsselwörter der Stufe 2 werden 72 wissenschaftliche Arbeiten identifiziert. Die zusätzlichen Schlüsselwörter der Stufe 3 schränken die Ergebnisse auf insgesamt 25 wissenschaftliche Arbeiten ein.

An dieser Stelle wird darauf hingewiesen, dass noch keine Datenbereinigung stattgefunden hat und innerhalb der 25 Arbeiten noch Duplikate möglich sind.

Anhand einer zentralen Erfassung der gefundenen wissenschaftlichen Arbeiten konnten insgesamt 89 Duplikate ausgeschlossen werden. Diese hohe Anzahl an mehrfachem Erscheinen einer Arbeit kann damit erklärt werden, dass durch das Hinzufügen der Schlüsselwörter der Stufen 2 und Stufe 3 der Pool an Arbeiten weiter eingeschränkt wird und relevante Arbeiten, welche den eingeschränkten Suchkriterien entsprechen, weiterhin aufscheinen. Es kann somit behauptet werden, dass die

ausgewählten Schlüsselwörter, welche stufenweise verfeinert werden, relevante Arbeiten des Bereichs abdecken.

Auf diesen Pool an wissenschaftlichen Arbeiten werden zusätzlich zu den primären Auswahlkriterien auch die semantischen Auswahlkriterien angewandt, sodass 5 relevante Publikationen in weiterer Folge auf deren eingesetzte Zielfunktionen untersucht werden.

3.3 Anwendbare Erkenntnisse aus der systematischen Literaturrecherche für die definierten Forschungsziele

Die relevanten Publikationen, welche anhand der systematischen Literaturrecherche identifiziert wurden, sind im Anhang 8.1 dargestellt.

In Tabelle 14 kommen Harvey Balls zum Einsatz, um darzustellen in welchem Ausmaß eine Übereinstimmung mit den verwendeten Schlüsselwörtern festgestellt werden konnte. In dieser Tabelle ist auch ersichtlich, inwiefern die semantischen Auswahlkriterien auf die jeweiligen Arbeiten zutreffen.

In Tabelle 15 sind die Eckdaten der Arbeiten zusammengefasst und die Zielfunktionen und Nebenbedingungen werden hier erläutert. Im rechten Bereich dieser Tabelle ist ersichtlich, in welchem Ausmaß sich die Nebenbedingungen der jeweiligen Arbeit mit den Nebenbedingungen (2) – (5) dieser Arbeit decken.

Vergleich der identifizierten Ansätze

Die Arbeit von Lee et al. (2019) minimiert die Anzahl nicht durchgeführter Room Service-Bestellungen eines Hotels, welche von Robotern getätigt werden. In der Arbeit von Yao et al. (2019) werden Logistiktransporte, welche ebenfalls von Robotern durchgeführt werden, optimiert, indem die aufsummierten Kosten für Transporte und Wartezeiten minimiert werden. Die Arbeit Nejads et al. (2019) und auch die vorgehende Arbeit (Nejad, et al., 2018) desselben Hauptverfassers, bezieht sich auf die Durchführung und entsprechende Optimierung eines pick-and-place-Roboters an einem Arbeitsplatz, mithilfe dessen eine Zykluszeitminimierung durch die Optimierung der Transportabfolgen erzielt werden soll, wobei in der Arbeit von 2018 (Nejad, et al.) zusätzlich die Produktionskosten der Maschinen optimiert beziehungsweise minimiert werden. Die Arbeit von Bogner et al. (2018) beschreibt einen Mensch-Roboter-Arbeitsplatz zur Herstellung von Platinen, an dem ebenfalls die Minimierung der Gesamttaktzeit angestrebt wird.

Eine Gemeinsamkeit der fünf ausgewählten Arbeiten ist die Optimierung anhand von Boole'schen Optimierungsvariablen, welche 2-,3- oder auch 4-dimensional mathematisch formuliert sind und den Wert 1 annehmen, falls zu gegebenem Zeitpunkt jene Ressource den Job durchführt und ansonsten den Wert Null beibehalten.

Die angeführten wissenschaftlichen Publikationen sind allesamt mit der Minimierung der jeweiligen Zielfunktionen befasst.

Vier der ausgewählten Arbeiten befassen sich mit Multiagent-Aufgabezuteilung bei Robotern und nur die Arbeit von Bogner et al. (2018) berücksichtigt die Zusammenarbeit von Mensch und Roboter am selben Arbeitsplatz.

In den Arbeiten (Nejad, et al., 2018), (Nejad, et al., 2019) und (Bogner, et al., 2018) wird die Durchführungszeit minimiert. Bei (Nejad, et al., 2019) werden explizit die Wartezeiten berücksichtigt und der Unterschied zu der Arbeit desselben Hauptverfassers Nejad (2018) besteht darin, dass hier auch auf die Kostenminimierung eingegangen wird. Die Kosten werden auch in der Zielfunktion der Arbeit (Yao, et al., 2019) minimiert. Einzig in der Arbeit (Lee, et al., 2019) steht eine Minimierung nicht erteilter Jobs im Fokus.

Hinsichtlich der angewandten Nebenbedingungen haben alle ausgewählten Arbeiten die nicht-präemptive Durchführung aller geplanten Jobs gemeinsam, beziehungsweise die Bedingung, dass jeder Job zeitgleich nur von einer Ressource durchgeführt werden kann.

Ableitbare Merkmale

Die Minimierung der Gesamtbearbeitungszeit wird auch in dieser Arbeit angestrebt.

Diese wird unter den Nebenbedingungen nicht-präemptiver Durchführung und der Bearbeitung jedes Jobs von einer Ressource zeitgleich geplant. Mit dieser Nebenbedingung wird implizit auch gewährleistet, dass jeder Job nur ein Mal durchgeführt wird.

Da in weiterer Folge nur die Arbeit von Bogner et al. (2018) mit dieser Arbeit vergleichbar ist, werden aus diesem Paper auch die Nebenbedingungen bezüglich der Vorgängerrelationen übernommen.

4 Entwicklung einer geeigneten Zielfunktion

Ausgehend von den theoretischen Grundlagen des Bereichs *Resource Constrained Project Scheduling Problem (RCPSP)* und der durchgeführten systematischen Literaturrecherche mit Hinblick auf die Taktzeitoptimierung in der Mensch-Roboter-Kollaboration werden in weiterer Folge zwei Ansätze für eine geeignete Zielfunktion mit Nebenbedingungen präsentiert.

Die Zielfunktion und auch die Nebenbedingungen werden auf einen beliebigen Arbeitsplatz in der Produktion ausgelegt, für welchen der Anwender die Anzahl an Ressourcen – Menschen als auch Roboter – und durchzuführender Jobs in einem gewissen Rahmen frei festlegen kann. Im Zuge der Auswertung in Kapitel 6 wird zur Ermittlung von Ergebnissen ein konkretes Fallbeispiel verwendet.

Für diesen Arbeitsplatz wird eine kosten- und taktzeitoptimierende Zielfunktion aufgestellt, für welche – bei stetig steigender Anzahl der verfügbaren Ressourcen und auch der durchzuführenden Jobs – so lang wie möglich eine exakte Lösung mittels *Linearer Programmierung* ermittelt wird. Mit steigender Anzahl an Ressourcen und Jobs wird jedoch eine exakte Lösung nicht mehr in zumutbarer Zeit auffindbar sein. Ab diesem Zeitpunkt kann die Aufgabe schneller mithilfe des *genetischen Algorithmus* gelöst werden.

4.1 Konzeptionelle Überlegungen einer möglichen Zielfunktion

Die generalisierte Annahme bezüglich des Arbeitsplatzes bietet den Vorteil, dass sie als „Fertiglösung“ für unterschiedliche Konstellationen angewandt werden kann, sofern Kosten, Bearbeitungszeiten und Vorgängerrelationen der verfügbaren Ressourcen bekannt sind.

Folgende Zielfunktion

$$\min\left(\sum_{j=1}^{J}\sum_{i=1}^{R}\sum_{t=0}^{T} PT_{ji} * x_{jit} * c_i * p_p\right) \tag{1}$$

und ihre Nebenbedingungen werden aufgestellt:

$$\sum_{i\in R}\sum_{t\in T} x_{jit} = 1\,, \forall j \in J \tag{2}$$

$$\sum_{j\in J}\sum_{l=t-PT_{ji}+1} x_{jil} \leq 1, \forall i \in R, \forall t \in T \tag{3}$$

$$\sum_{i \in R} \sum_{t \in T} (t + PT_{ki}) * x_{kit} \leq \sum_{i \in R} \sum_{t \in T} t * x_{jit}, \forall j \in J \; und \; \forall k \in V_j \tag{4}$$

$$x_{jit} \in \{0,1\} \tag{5}$$

wobei

$PT_{ji} \ldots Bearbeitungszeit\ der\ Tätigkeit\ j\ durch\ die\ Ressource\ i$

$x_{jit} \ldots Boole'sche\ Entscheidungsvariable \begin{cases} 1, wenn\ Tätigkeit\ j\ von\ Ressource\ i \\ zu\ Zeitpunkt\ t\ gestartet\ wird \\ 0, sonst \end{cases}$

$c_i \ldots Kosten\ je\ Zeiteinheit\ (Sekunde) für\ Ressource\ i$

$p_p \ldots Pönalefaktor\ für\ Prozentintervall\ p$

Die zu minimierende Zielfunktion (1)(10) setzt sich zusammen aus der Summe aller – mithilfe der Boole'schen Variablen – selektierter Bearbeitungszeiten, welche mit dem jeweils der Ressource entsprechenden Kostensatz multipliziert und um einen Pönalefaktor skaliert werden.

Im letzten Bereich der Tabelle 15 wird mit Harvey Balls dargestellt, in welchem Ausmaß die hier angewandten Nebenbedingungen in der Literatur der systematischen Recherche anzutreffen sind.

Nebenbedingung (2) gewährleistet, dass jeder durchzuführende Job einmal von einer Ressource gestartet und ohne Unterbrechung durchgeführt wird, während die Nebenbedingung (3) dafür sorgt, dass jede Ressource nur an einem Job zeitgleich arbeitet.

Nebenbedingung (4) stellt sicher, dass Vorgängertätigkeiten, welche in der Vorgängermatrix definiert sind, abgeschlossen werden, bevor eine Nachfolgetätigkeit gestartet wird.

Mit Bedingung (5) wird das binäre Format der Boole'schen Entscheidungsvariable festgelegt.

Seitens der Anwender sind folgende Eingaben notwendig, um die Optimierung entsprechend zu betreiben:

- Anzahl der verfügbaren Ressourcen und Jobs,
- Bearbeitungszeiten der jeweiligen Ressourcen für alle Jobs,
- Kosten der verfügbaren Ressourcenarten,
- Vorgängermatrix – welche Jobs vor anderen durchzuführen sind.

Der erwähnte Pönalefaktor sollte den Neuheitsfaktor dieser Arbeit darstellen, da er exzessive Wechsel zwischen den Ressourcen unterbindet.

Wenn nur mit Hinblick auf die minimalen Bearbeitungszeiten jeder Job von einer anderen Ressource durchgeführt wird, dann werden unter Umständen alle Ressourcen bei der Bearbeitung der Jobs eingeplant. Hiermit wäre die gesamte Bearbeitungszeit tatsächlich minimiert, aus ressourcenplanerischer Sicht würde dies jedoch – bei Arbeitszeiten im Minutenbereich – eine Belegung aller Ressourcen bedeuten und dies wäre entsprechend ineffizient. Um dieser übermäßigen Minimierung durch Wechsel entgegenzuwirken, wird der Pönalefaktor eingeführt, welcher übermäßige Wechsel sanktionieren soll.

Mithilfe der Variable a_{ji} in (6) werden Wechsel der eingesetzten Ressource zwischen aufeinanderfolgenden Jobs gezählt. Wenn die Anzahl der Wechsel einen gewissen Prozentsatz der gesamten Anzahl an durchzuführenden Tätigkeiten überschreitet, dann wird das Ergebnis der gerade ermittelten Zielfunktion mit einem Gewichtungsfaktor multipliziert und somit skaliert.

$$a_{ji} = \sum_{t=0}^{T} x_{jit}\,, \forall i \in R \; und \; \forall j \in J \tag{6}$$

$$w_i = \sum_{j=1}^{J} |a_{ji} - a_{(j+1)i}|\,, \forall i \in R \tag{7}$$

$$p_p \; ist \begin{cases} 0{,}6 \; wenn \; w_i \in [71\%, 100\%] \\ 0{,}8 \; wenn \; w_i \in [35\%, 70\%] \\ 1 \; wenn \; w_i \in [0\%, 34\%] \end{cases} \tag{8}$$

Der zugrundeliegende Gedanke des Pönalefaktors besteht darin, in einem ersten Schritt (6) die zeitliche Dimension der Optimierungsvariable zu entfernen, sodass eine zweidimensionale Boole'sche Matrix entsteht, in welcher die Spalten Ressourcen und die Zeilen die jeweiligen Jobs darstellen. Es kommen Einser vor, wenn die entsprechende Ressource den jeweiligen Job durchführt. In einem nachfolgenden Schritt (7) werden die absoluten Differenzen zwischen aufeinanderfolgenden Zeilen dieser neuen Matrix aufsummiert, womit bekannt ist, ob aufeinanderfolgende Jobs von unterschiedlichen Ressourcen durchgeführt werden – falls die absolute Differenz 1 ist – oder nicht andernfalls.

In (8) werden Grenzen festgelegt, welcher Pönalefaktor in der Zielfunktion zu tragen kommt. Dies geschieht in Abhängigkeit von der Anzahl an Wechsel (prozentual dargestellt) zwischen den Ressourcen, welche aufeinanderfolgende Jobs durchführen.

Dieser Ansatz muss jedoch aus zwei Gründen verworfen werden.

Einerseits erweist sich die lineare Implementierung des Pönalefaktors aufgrund der anzuwendenden Struktur der zu implementierenden Optimierungsvariable als unmöglich. Die mathematische Formulierung der Zielfunktion ist für den Leser/die Leserin zwar leichter verständlich, in der Implementierung in MATLAB kann jedoch keine dreidimensionale Optimierungsvariable angewandt werden, sondern es wird eine zweidimensionale Matrix erstellt, in der indexiert nacheinander alle Kombinationen der Startzeiten der Jobs für jede Ressource aufgelistet sind. Somit kann der – mathematisch korrekte Ansatz der Formeln (6) – (8) nicht codiert werden.

Ein weiterer Grund, weshalb diese Form der Pönalisierung von Nachteil ist, besteht darin, dass eine strikte aufeinanderfolgende Abarbeitung der Jobs vorausgesetzt werden muss, um auch entsprechend die Wechsel aufeinanderfolgender Tätigkeiten zu pönalisieren. Diese Voraussetzung schränkt jedoch in der Praxis sehr stark ein, da nur eine bestimmte Bearbeitungsreihenfolge der Jobs möglich ist. Mithilfe der Vorgängermatrix und der entsprechenden Vorgängerbeziehungen in Nebenbedingung (4) kann – durch die parallele Bearbeitung von Jobs – die Gesamtbearbeitungszeit reduziert werden.

4.2 Implementierte Zielfunktion und Nebenbedingungen

Nach weiteren Überlegungen wurde die Entscheidung getroffen, die Zielfunktion als Summe der gewichteten normalisierten Faktoren: Gesamtbearbeitungszeit, Summe aller Startzeiten und der anfallenden Ressourcenkosten aufzustellen (10). Dieser Ansatz ist in der Arbeit Tsarouchis et al. (2017) anzutreffen.

Die Normalisierung beziehungsweise Skalierung des Wertebereichs dient dazu, die unterschiedlichen Anteile der Zielfunktion auf den Bereich zwischen 0 und 1 zu skalieren und somit trotz unterschiedlicher Größenordnung vergleichbar zu machen.

Die Bestandteile der Zielfunktion (10) werden alle minimiert und somit kommt folgende Form der Skalierung zum Einsatz:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{9}$$

In (9) wird der normalisierte Wert x' aus dem ursprünglichen Wert x erzeugt, indem von diesem der minimale Wert des vorliegenden Datenbereichs subtrahiert und anschließend durch den vorliegenden Wertebereich dividiert wird und somit x' im Intervall [0,1] denselben Stellenwert wie im ursprünglichen Intervall $[\min(x), \max(x)]$ aufweist.

In der Zielfunktion (10) besteht die Möglichkeit, die normalisierten Minima der drei Bestandteile mit einem Gewichtungsfaktor zu multiplizieren, sodass – in Abhängigkeit vom individuellen Fokus – der Nutzwert den Anforderungen der jeweiligen

Arbeitsstation entspricht. In dieser Arbeit wird, ähnlich zur Arbeit Tsarouchis, (2017), vorerst eine gleichverteilte Gewichtung von $g_z = 0{,}33$ gewählt.

$$\min\ (g_1 * F'_1 + g_2 * F'_2 + g_3 * F'_3) \tag{10}$$

wobei

$F'_1, F'_2, F'_3 \ldots normalisierte\ Werte\ folgender\ Teilziele\ darstellen$

$$F_1 = min\ [C_{max} + (R_H * ST_H + R_R * ST_R)] \tag{11}$$

$$F_2 = min \sum_{j=1}^{J} S_{ji}\,, i \in R \tag{12}$$

$$F_3 = min(\sum_{j=1}^{J}\sum_{i=1}^{R} x_{ji} \times c_i \times PT_{ji} + \sum_{i=1}^{R} c_i \times STB_i) \tag{13}$$

unter den Nebenbedingungen:

$$C_{max} \geq C_{ji}, \forall j \in J\ und\ \forall i \in R \tag{14}$$

$$\sum_{i=1}^{R} x_{ji} = 1\,, \forall j \in J \tag{15}$$

$$\sum_{j \in J}\ \sum_{t-PT_{ji}+1} x_{ji} \leq 1, \forall i \in R, \forall t \in T \tag{16}$$

$$C_{ji} - S_{ji} \geq \sum_{i=1}^{R} \left(x_{ji} \times PT_{ji}\right), \forall j \in J \tag{17}$$

$$x_{ji}, x_{jij^*i^*} \in \{0,1\} \tag{18}$$

wobei

$g_z, g_1, g_2, g_3 \ldots Gewichtungsfaktor\ (allgemein\ beziehungsweise\ f\ddot{u}r\ die\ einzelnen\ Bestandteile\ der\ Zielfunktion)$

$F_1, F_2, F_3 \ldots Bestandteile\ der\ Zielfunktion$

$C_{max} \ldots Fertigstellungszeitpunkt\ der\ Bearbeitung\ aller\ Jobs$

$i, i^* \in R \ldots verf\ddot{u}gbare\ Ressourcen$

$j, j^* \in J \ldots alle\ durchzuf\ddot{u}hrenden\ Jobs$

$R_H, R_R \ldots Gesamtanzahl\ eingesetzter\ Ressourcen\ Mensch\ beziehungsweise\ Roboter$

ST_H, ST_R … $Rüstzeit\ für\ Ressource\ Mensch\ beziehungsweise\ Roboter$

S_{ji} … $Startzeit\ Bearbeitung\ Job\ j\ durch\ Ressource\ i$

STB_i … $Wartezeit\ der\ Ressource\ i\ zwischen\ Jobs$

C_{ji} … $Fertigstellungszeitpunkt\ des\ Jobs\ j\ durch\ die\ Ressource\ i$

PT_{ji} … $Bearbeitungszeit\ des\ Jobs\ j\ durch\ die\ Ressource\ i$

c_i … $Kosten\ für\ die\ Ressource\ i\ je\ Zeiteinheit$

x_{ji} … $1\ wenn\ Job\ j\ von\ Ressource\ i\ durchgeführt\ wird, 0\ sonst$

$x_{jij^*i^*}$ … $1\ wenn\ Job\ j^*\ durch\ Ressource\ i^*\ vor\ Job\ j\ durch\ Ressource\ i, 0\ sonst$

Der in (11) dargestellte erste Bestandteil der Zielfunktion minimiert die maximale Bearbeitungszeit, welche zur Durchführung aller Jobs gemäß (14) benötigt wird. Der Ansatz der Minimierung der Gesamtbearbeitungszeit C_{max} ist ein übliches Optimierungsziel im RCPSP und wird auch in den Arbeiten von Tsarouchi et al. (2017), Monguzzi et al. (2022) und Ren et al. (2022) angewandt.
Die Gesamtbearbeitungszeit wird in dieser Arbeit um die Rüstzeiten je eingesetzter Ressourcen Mensch und Roboter erweitert, um somit einen ressourcenminimierenden Umgang zu erreichen. Dieser Ansatz stellt, soweit bekannt, einen Neuheitsfaktor dar.

In (12) wird die Summe aller Startzeiten der Bearbeitung aller Jobs minimiert. Mit dieser Komponente, welche auch in Monguzzi et al. (2022) zur Anwendung kommt, wird erzielt, dass die Jobs so früh wie möglich eingeplant und somit Stillstandzeiten reduziert werden.

Die dritte und somit letzte Komponente (13) der zu minimierenden Zielfunktion stellt – in dieser Form – den Neuheitswert dieser Arbeit dar.

Hier wird der Kostenfaktor minimiert, wobei nicht nur die Bearbeitungszeiten PT_{ji} der Ressourcen berücksichtigt werden, sondern auch etwaig anfallende Stillstandzeiten STB_i einer Ressource, wenn diese davor und danach eingeplant ist. Hiermit wird versucht, die Ressourcen durchgehend so gut wie möglich auszulasten, sodass Stillstände zwischen Jobs eines Planungsintervalls so niedrig wie möglich gehalten werden. Eventuelle Stillstände vor einem Einsatz im betrachteten Zyklus werden nicht berücksichtigt, da hier angenommen werden kann, dass diese Ressource entweder in einem vorgelagerten Zyklus oder an einer anderen Arbeitsstation eingesetzt werden kann.

Es ist anzumerken, dass dieser dritte Bestandteil der Zielfunktion nicht in der linearen Programmierung zum Einsatz kommt, da dies – wie auch im Fall des Pönalefaktors in

4.1– im linearen Kontext aufgrund des Aufbaus der Optimierungsvariablen nicht umgesetzt werden kann. Nichtsdestotrotz wird diese Komponente als wertvolle Erweiterung der zu minimierenden Zielfunktion betrachtet, sodass sie im Rahmen des metaheuristischen Ansatzes implementiert wird.

Durch die Berücksichtigung der drei Komponenten der Zielfunktion – gesamte Bearbeitungszeit, Startzeit aller Jobs und anfallende Ressourcenkosten – wird der ermittelte Zeitplan gleichzeitig sowohl im Hinblick auf die Taktzeit als auch auf die Kosten minimiert.

Die Nebenbedingungen sind im Vergleich zu jenen in Abschnitt 4.1 marginal abgewandelt. Dies musste durchgeführt werden, da von der mathematischen dreidimensionalen Sichtweise der Optimierungsvariable abgewichen wird, wie bereits in Bezug auf den Pönalefaktor und dessen Nebenbedingungen in den Gleichungen (6) – (8) erläutert.

Mit der Nebenbedingung (14) wird die Gesamtbearbeitungszeit zur Durchführung aller Jobs nach dem Fertigstellungszeitpunkt aller Jobs festgelegt.

Gleichung (15) gewährleistet, dass jeder Job genau einer Ressource zugewiesen wird.

Gleichung (16) ist ebenso wie die Gleichung (3) dafür zuständig, dass eine Ressource nicht gleichzeitig mehrfach eingeplant wir. Dazu wird für jede Ressource geprüft, dass sich die Bearbeitungszeit des eingeplanten Jobs mit keiner weiteren Bearbeitungszeit für einen anderen Job (durch dieselbe Ressource) überschneidet.

Mit der Nebenbedingung (17) wird die Bearbeitungszeit PT_{ji} eingehalten, da zwischen Fertigstellung- und Startzeit eines Jobs mindestens die Bearbeitungszeit selbst vergehen muss.

Die formelle Gleichung (18) legt fest, dass die mathematisch dargestellten Optimierungsvariablen die Funktion einer ja/nein-Rückmeldung erfüllen.

5 Implementierung der Algorithmen

Im Rahmen der Implementierung wird ein konkretes Fallbeispiel angewandt, damit die Umsetzung veranschaulicht werden kann.

Für die umgesetzte Aufgabe stehen als Ressourcen jeweils 3 Menschen und 3 Roboter zur Verfügung und es müssen 7 Jobs eingeplant werden.

Die Eingabe erfolgt über eine Excel-Tabelle. Für das vorliegende Fallbeispiel sind in Abschnitt 8.2 die dazugehörigen Abbildungen dargestellt.

Für die Eingaben zu den Ressourcen, wie in Tabelle 16 ersichtlich, können im unteren Bereich Angaben zur Arbeitsumgebung eingegeben werden, aus denen die Kosten für die Ressourcen berechnet werden.

Ebenfalls über die Excel-Tabelle werden die Bearbeitungszeiten für die Ressourcen Mensch und Roboter (Tabelle 17) eingegeben, beziehungsweise die entsprechende Vorgängermatrix ausgefüllt (Tabelle 18).

Die Bearbeitungszeiten sind jeweils für Mensch und Roboter gleich. Die Anzahl der verfügbaren Menschen oder Roboter über die Eingabe ermöglicht es entsprechend, eine größere Mannschaft der jeweiligen Ressourcen zur Verfügung zu haben.

Da in der Zeile „Task 04“ alle Werte Null sind kann aus der Vorgängermatrix (Tabelle 18) gelesen werden, dass kein anderer Task Vorgänger sein muss und somit dieser Job beliebig eingeplant werden kann. Im Gegensatz hierzu sind in der Zeile „Task 07“ Einser in den ersten sechs Spalten, was heißt, dass die Jobs 01 – 06 alle abgeschlossen sein müssen, bevor dieser letzte Job gestartet werden kann. Die Diagonale der Vorgängermatrix muss durchgehend mit dem Wert Null belegt sein, da ein Job nicht sich selbst als Vorgänger haben kann.

Zur Ermittlung der Kosten für die Ressourcen werden die Eingaben in Tabelle 16 ausgehend von Erfahrungswerten in der Produktion erstellt.
Bei einem Einschichtbetrieb kann eine durchschnittliche Produktivität der gewerblichen Mitarbeitern von 85% angenommen werden, womit beispielsweise Urlaubszeiten und Krankenstände berücksichtigt werden.
Die Bruttokosten des Dienstgebers für diesen Mitarbeiter belaufen sich auf ungefähr 60.000€/Jahr, gemäß dem Kollektivvertrag für Eisen- und Metallverarbeitenden Gewerbe 2022 (Wirtschaftskammer Österreich, 2022).
Ein industrieller Roboter kann mit einer Produktivität von circa 95% ausgelegt werden, wobei Service und Instandhaltungszeiten berücksichtigt werden. Die Anschaffungskosten belaufen sich auf 350.000€, von denen der reine Anschaffungspreis 30% darstellt und die Differenz von der Peripherie (Kameras, Sensoren, Software) und Integration aller Komponenten (beispielsweise

Programmierung oder Dokumentation) verursacht wird (Klaerner, 2021). Laut der Absetzung für Abnutzung (AfA) in Österreich kann dieser Betrag in einer Laufzeit von 6 Jahren abgeschrieben werden. Für Service und Instandhaltung fallen üblicherweise jährliche Kosten von 25% der AfA an.

Mit den getätigten Eingaben errechnen sich höhere Kosten für die Roboterminute von 5,16 Cent (€) im Vergleich zur Ressource Mensch.

Das in (9) beschriebene Verfahren benötigt minimale und maximale Werte, welche für die Normalisierung herangezogen werden.
In dieser Arbeit werden die Minima und Maxima mit nachfolgendem Vorgehen ermittelt:

- Minimierung der maximalen gesamten Bearbeitungszeit (11): Die untere Grenze wird mit 1 festgelegt, die obere Grenze stellt die Summe der maximalen Bearbeitungszeiten der Jobs dar. Die Rüstzeiten werden hier nicht berücksichtigt, da sie einen Störfaktor darstellen und entsprechend gering gehalten werden sollen, um die Minimierung zu bestärken.
- Minimierung der Summe der Startzeiten (12): Als untere Grenze dient die Anzahl der durchzuführenden Jobs. Als obere Grenze gilt dieselbe Anzahl, multipliziert mit der maximalen Bearbeitungszeit.
- Minimierung Arbeitszeiten und Stillstandzeiten (13): Die aufsummierten minimalen und maximalen Bearbeitungszeiten je Job werden mit der günstigsten beziehungsweise teuersten Ressource multipliziert.

Für eine laufende Ermittlung der minimalen und maximalen Werte der Bestandteile der jeweils betrachteten Zielfunktion wird ein verhältnismäßig unrealistischer Rechenaufwand benötigt, der nicht gerechtfertigt ist. Die angewendeten Grenzwerte skalieren die jeweiligen Anteile der Zielfunktion zufriedenstellend, sodass diese Lösung akzeptiert wird.

In den nachfolgenden Abschnitten 5.1 bis 5.3 werden die Hauptbestandteile der implementierten Algorithmen beschrieben. Innerhalb der dazugehörigen Codes (8.3 bis 8.5) werden weitere implementierte Funktionen abgerufen, welche im Abschnitt 0 angehängt sind. Da es sich in weiterer Folge um Hilfsfunktionen zur Visualisierung der Daten oder ähnliches handelt, werden diese nicht weiter erläutert.

5.1 Lineare Optimierung

Der implementierte Code der linearen Optimierung ist in 8.3 verfügbar.

Gemischt-ganzzahlige-Aufgaben werden Solver-basiert in MATLAB in folgender Form bereitgestellt:

$$\min \boldsymbol{f} * \boldsymbol{x} \tag{19}$$

$$\boldsymbol{x}\ hat\ ganzzahlige\ Werte \tag{20}$$

$$A * \boldsymbol{x} \leq \boldsymbol{b} \tag{21}$$

$$A_{eq} * \boldsymbol{x} = \boldsymbol{b_{eq}} \tag{22}$$

$$\boldsymbol{l}_b \leq \boldsymbol{x} \leq \boldsymbol{u}_b \tag{23}$$

Entsprechend (21) müssen alle Ungleichungen zur Matrix A und zum Vektor $\boldsymbol{b}$ zusammengefasst werden. Die Gleichungen, welche die zu minimierende Zielfunktion einschränken, werden in der Matrix A_{eq} und der rechten Seite $\boldsymbol{b_{eq}}$ nacheinander aufgelistet (22). In $\boldsymbol{l}_b$ und $\boldsymbol{u}_b$ werden die Grenzwerte, die $\boldsymbol{x}$ annehmen kann, definiert.

Im ersten Schritt wird die Zielfunktion (10) mit ihren Nebenbedingungen mithilfe der linearen Programmierung von MATLAB. Somit kann das Optimum der Zielfunktion mit der exakten Lösung der Aufgabe ermittelt werden.

Hier wird darauf hingewiesen, dass dieser Lösungsansatz nicht die Anteile der Rüstzeiten (11) und Stillstandzeiten (13) berücksichtigt, wodurch diese Gleichungen folgende Form annimmt:

$$F_1 = min\ C_{max} \tag{24}$$

$$F_3 = min(\sum_{j=1}^{J}\sum_{r=1}^{R} x_{jr} \times c_r \times PT_{jr}) \tag{25}$$

Die maximale Bearbeitungszeit [t_{max}], welche in weiterer Folge benötigt wird (im vorliegenden Beispiel 19 Zeiteinheiten), errechnet sich als die maximal mögliche Bearbeitungszeit bei der Bearbeitung durch eine der verfügbaren Ressourcen. Mit diesem Ansatz kann davon ausgegangen werden, dass die maximale (sinnvolle) Bearbeitungszeit im Rahmen der Optimierung nicht überschritten wird, oder dass gute Lösungen nicht berücksichtigt werden.

Implementiert wird die Optimierungsvariable als Tabelle 3, welche alle möglichen Kombinationsvarianten von Job-Ressource-Startzeit enthält. Diese wird aus den vorab aus Excel importierten Daten erstellt.

Wie der Tabelle 3 zu entnehmen ist, wird im Index *[ix]* eine fortlaufende Nummerierung vorgenommen, während alle Tätigkeiten *[task]* mit allen Ressourcen *[resource]* zu allen möglichen Startzeiten (bis zur maximalen Bearbeitungszeit) chronologisch kombiniert werden.

$$ix_{max} = j_n * i_m * t_{max} \tag{26}$$

In (26) ist die Berechnungsformel für ix_{max} – der maximalen Anzahl an Optimierungsvariablen – dargestellt.

Tabelle 3: Lineare Optimierungsvariable (MATLAB Tabelle) für 6 Ressourcen (3 Menschen und 3 Roboter), 7 Jobs und einer sich ergebenden maximalen Bearbeitungszeit von 19 Zeiteinheiten

ix	task	resource	time
1	1	1	1
2	1	1	2
3	1	1	3
4	1	1	4
5	1	1	5
6	1	1	6
7	1	1	7
9	1	1	9
10	1	1	10
11	1	1	11
12	1	1	12
13	1	1	13
14	1	1	14
15	1	1	15
16	1	1	16
17	1	1	17
18	1	1	18
19	1	1	19
20	1	2	1
21	1	2	2
22	1	2	3
23	1	2	4
24	1	2	5
25	1	2	6
26	1	2	7
765	7	5	5
766	7	5	6
767	7	5	7
768	7	5	8
769	7	5	9
770	7	5	10
771	7	5	11
772	7	5	12
773	7	5	13
774	7	5	14
775	7	5	15
776	7	5	16
777	7	5	17
778	7	5	18
779	7	5	19
780	7	6	1
781	7	6	2
782	7	6	3
783	7	6	4
784	7	6	5
785	7	6	6
786	7	6	7
787	7	6	8
788	7	6	9
789	7	6	10
790	7	6	11
791	7	6	12
792	7	6	13
793	7	6	14
794	7	6	15
795	7	6	16
796	7	6	17
797	7	6	18
798	7	6	19

Im vorliegendem Beispiel ist

$$ix_{max} = 7 * 6 * 19 = 798 \tag{27}$$

Mit diesem notwendigen Vorgehen bei der Erstellung der Optimierungsvariablen ist ersichtlich, dass mit steigender Anzahl an Ressourcen und Jobs und bei langen Bearbeitungszeiten die Anzahl der zu berücksichtigenden Optimierungsvariablen rasant zunimmt.

Nach Aufbereitung der Eingaben über die Excel-Tabelle und Erzeugung der Index-Tabelle werden die Nebenbedingungen implementiert.

Für die Nebenbedingung (15) werden mittels for-Schleife für alle Jobs die zugehörigen Indices durchlaufen und entsprechend die Zellen in der Gleichungsmatrix A_{eq} mit 1 belegt. Auf der rechten Seite $\boldsymbol{b}_{eq}$ steht ebenfalls der Wert 1. Somit darf die Summe der jeweiligen Zeile jedes Jobs nicht den Wert 1 der rechten Seite (also einmal zugewiesen) überschreiten.

Die Implementierung der Ungleichungen der Nebenbedingung (16) ist jedoch nicht mehr so einfach in Worte zu fassen. Hier werden drei for-Schleifen ineinander geschachtelt.
Die erste Schleife definiert den „aktiven“ Job, dessen Bearbeitungszeit – abhängig von der angewandten Ressource – ebenfalls erfasst wird.
In einer zweiten for-Schleife werden alle möglichen Zeiten für den gewählten Fall durchlaufen. Die Indices der „aktiven“ Jobs werden identifiziert und die notwendigen Zeiten für deren Bearbeitung geblockt. Dies passiert, indem alle „anderen“ Jobs mit der dritten for-Schleife durchlaufen werden und – wenn ein „anderer“ Job dieselbe Ressource in der blockierten Zeit verwenden würde – für der entsprechenden Index eine Ungleichung in der Matrix A hinzugefügt wird, beziehungsweise der entsprechende Wert in Vektor b auf 1 gestellt.

Die Vorgängerbeziehungen – welche über die excel-Tabelle als Matrix importiert werden – werden mit der Erweiterung der Ungleichungsbeziehungen festgelegt.

Hierzu werden vier for-Schleifen verschachtelt durchlaufen. Mit der ersten Schleife werden alle Job durchlaufen und die entsprechenden Vorgänger gespeichert. Die zweite Schleife durchläuft alle vorhin gespeicherten Vorgänger des Jobs. Da unterschiedliche Bearbeitungszeiten der Jobs durch unterschiedliche Ressourcen möglich sind, geht die dritte Schleife durch alle Ressourcen und speichert die Bearbeitungszeit der aktuellen Ressource. Mit der vierten Schleife werden alle möglichen Zeiten durchlaufen und die Indices des aktuellen Jobs und der aktuellen Zeit identifiziert. Für die Indices, für welche die Vorgänger noch bearbeitet werden oder welche später stattfinden, wird eine neue Zeile der Ungleichungsmatrix A und in der entsprechenden rechten Seite der Wert 1 hinzugefügt.

Mit den soeben beschriebenen zwei Code-Abschnitten wird die Nebenbedingung (17) eingehalten.

Die Bedingung für binäre Variablen in Gleichung (18) wird mit dem Parameter *intcon* für alle Indices und der gleichzeitigen Anwendung der unteren $\boldsymbol{l}_b$- und oberen $\boldsymbol{u}_b$-Schranken für diese (mit den Werten 0 und 1) implementiert.

Anschließend wird die Zielfunktion (10) implementiert.

Dies erfolgt ebenfalls über eine dreifache for-Schleife über alle Jobs, Ressourcen und Zeiten, innerhalb derer die Spalten-Vektoren für die Bestandteile der Teilzielfunktionen (11) und (12) erstellt werden. Der Spaltenvektor für (11), welcher die Minimierung der Gesamtsumme der Startzeiten erzielt, hat ix_{max} Positionen. Diese sind deckungsgleich mit der vierten Spalte der Indices-Matrix aus Tabelle 3 zu verstehen und stellen den Zeitpunkt dar, an dem gestartet wird, da diese „Kosten" bei der Minimierung der Gesamtstartzeiten benötigt werden.

Der Spaltenvektor (12) hat ebenfalls ix_{max} Einträge mit dem jeweiligen Produkt aus Ressourcenkosten und Bearbeitungszeiten der jeweiligen Ressource. Falls die entsprechenden Indices zum Tragen kommen, wird für (12) der entsprechende Preis des Jobs durch die ausgewählte Ressource zur Summe addiert.

Der dritte Bestandteil der Zielfunktion minimiert die Gesamtbearbeitungszeit. Hier werden in dem mit Nullen belegten Spaltenvektor (ebenfalls mit ix_{max} Einträgen) nur Zellen belegt, wenn sie dem letzten Dummy-Job entsprechen.

Die Zielfunktion (10) wird nach den for-Schleifen als Summe der normalisierten und gewichteten Teilziele definiert.

Nach Datenimport, Implementierung der Nebenbedingungen und der Zielfunktion kann der Solver *intlinprog* ausgeführt werden.

Der Output besteht aus der Laufzeit des Solvers, dem Zeitplan mit den zugewiesenen Jobs an die Ressourcen und Startzeiten und einer grafischen Darstellung dieses Zeitplans.

5.2 Lineare Optimierung mit for-Schleife und genetischem Algorithmus

In einem nächsten Entwicklungsschritt wird dieselbe lineare Lösung implementiert, jedoch mit einem vorgelagerten Schritt zur Ermittlung aller möglichen Kombinationen an Ressourcen, die getestet werden müssen und sollen. Ziel dieser Vorab-Kombinationsaufbereitung ist es, im Rahmen der Optimierung die Anzahl der eingeplanten Ressourcen zu minimieren.

Durch die Bereitstellung der verfügbaren Ressourcen-Kombinationen wird bewirkt, dass alle möglichen Kombinationen an Ressourcen getestet und bewertet werden,

jedoch mit dem Nachteil, dass $2^{Ressourcen}$–Kombinationen ermittelt und bewertet werden müssen, worunter die Laufzeit entsprechend verlängert wird.

Der nächste logische Schritt ist somit, dass nicht alle Kombinationen an Ressourcen von einer for-Schleife erzeugt werden sondern mit einem genetischen Algorithmus. Dessen Optimierungsvariablen sind binär und entsprechen der Gesamtanzahl an Ressourcen.
Die lineare Implementierung der Zielfunktionsberechnung bleibt weiterhin erhalten, wie in Abschnitt 5.1 dargestellt.

Der Vorteil des genetischen Algorithmus wird hier ausgeschöpft, da – im Gegensatz zu for-Schleifen – nicht alle Kombinationen der Ressourcenzuweisung erstellt und getestet werden müssen, sondern der Algorithmus selbst von einer Eltern- zu Kindergeneration ungünstige Kombinationen nicht weiterverfolgt.

Durch das Ersetzen der kombinatorischen Erzeugung aller Ressourcenzuweisungen für die Jobs durch den vorgelagerten genetischen Algorithmus wird keine Verbesserung der Laufzeit erzielt (wie in Abschnitt 6.1.2 anhand des Fallbeispiels ersichtlich wird), wobei weiterhin ein lineares Programm gelöst wird. Die Änderungen, welche am Code aus Abschnitt 8.3 durchgeführt werden, sind in 8.4 ersichtlich.

5.3 Doppelter genetischer Algorithmus

Da weiterhin im linearen Programm nicht alle gewünschten Anteile der Zielfunktion berücksichtigt werden können – die Stillstandzeiten und Rüstzeiten werden in den Abschnitten 5.1 und 5.2 nicht ermittelt und berücksichtigt – beziehungsweise noch Verbesserungspotential bei der Iterationszeit zu erwarten ist, wird das lineare Programm als nächstes auch durch einen genetischen Algorithmus ersetzt. Dieser ist im Anhang 8.5 eingefügt.

Es entsteht ein verschachtelter genetischer Algorithmus, der folgendermaßen funktioniert:
Der erste (hauptsächliche) genetische Algorithmus entscheidet, welcher Job von welcher Ressource durchzuführen ist. Diese Information wird an den zweiten (verschachtelten) genetischen Algorithmus übermittelt, welcher die Ressourcenzuweisung nicht mehr beeinflusst, sondern – unter Berücksichtigung der Nebenbedingungen – ermittelt zu welchem Zeitpunkt der Job zielfunktionsoptimierend eingeplant wird.

Dies stellt nur einen groben Überblick dar, daher wird an dieser Stelle auf die Details der Bearbeitung eingegangen.

Der Datenimport erfolgt genau wie in den Abschnitten 5.1 und 5.2.

Der hauptsächliche genetische Algorithmus hat als Optimierungsvariablen die Anzahl der durchzuführenden Jobs, welchen Ressourcen zugewiesen werden müssen. Diese sind ganzzahlige Optimierungsvariablen und deren untere und obere Grenze sind 1 sowie die verfügbare Anzahl an Ressourcen.
Der Unterschied zwischen dem vorgelagerten genetischen Algorithmus in 5.2 und hier besteht darin, dass im ersten die grundsätzliche Verfügbarkeit der Ressource entschieden wird und hier aus allen verfügbaren Ressourcen deren Zuweisung zu den Jobs durchgeführt wird.
Dieser erste genetische Algorithmus gewährleistet auch, dass die Nebenbedingung (15) eingehalten wird.

Die erzeugte Belegungskombination wird dem verschachtelten Algorithmus gemeinsam mit anderen notwendigen Daten bereitgestellt.

In diesem werden zuerst die Nebenbedingungen für die Vorgängerbeziehungen als Ungleichungen erzeugt, indem mit for-Schleifen alle Ressourcen durchlaufen werden. Für diese werden jeweils die zugewiesenen Jobs durchlaufen und deren Vorgänger gefunden. Für jeden der ermittelten Vorgänger werden die durchführenden Ressourcen identifiziert und – da die Bearbeitungszeit ressourcenabhängig ist – eine neue Zeile in der Ungleichungsmatrix erzeugt, wobei die rechte Seite der Ungleichung der ermittelten Bearbeitungszeit entspricht. Mit diesem Schritt wird die Nebenbedingung (17) gewährleistet.

Um zu gewährleisten, dass eine Ressource nicht mehrfach zum selben Zeitpunkt belegt werden kann (wie in der Nebenbedingung (16) festgehalten), wird im verschachtelten genetischen Algorithmus eine nichtlineare Bedingung eingeführt. Hierfür werden mit einer for-Schleife alle Ressourcen – und für jede Ressource alle ihr zugewiesenen Jobs – nacheinander betrachtet. Für jeden Job wird aus der bekannten Startzeit und Bearbeitungszeit die Fertigstellungszeit berechnet. Diese belegten Zeiten werden nacheinander einem Zeilenvektor zugewiesen. Gleichzeitig wird die Gesamtbearbeitungszeit der Ressource ermittelt.
Die Mehrfachbelegungslogik besteht darin, dass für jede Ressource die Anzahl der eindeutigen Werte im erstellten Zeilenvektor gleich der Gesamtbearbeitungszeit durch diese Ressource sein muss. Wenn es weniger eindeutige Werte gibt, dann heißt das, dass die Ressource mehrfach belegt ist und somit ist die Ungleichheitsbedingung verletzt wird.
Diese Ungleichungen werden zeilenweise in einer Matrix gespeichert.

Die Nebenbedingung (18) wird im Rahmen dieser Implementierung nicht benötigt.

Die zu optimierende Zielfunktion des verschachtelten genetischen Algorithmus, welche auch die Zielfunktion des ersten Algorithmus darstellt, entspricht (10), wobei im Anteil der Gleichung (13) auch die Stillstandzeiten berücksichtigt werden können.

Die Implementierung von Gleichung (13) ist mit einem ähnlichen Ansatz zur soeben beschriebenen Mehrfachbelegung möglich. Für jede Ressource werden die zugewiesenen Jobs durchlaufen und die belegten Zeiten in einem Vektor – durch Aufzählung der Zeitpunkte – festgehalten, beziehungsweise abschließend steigend sortiert. In einem zweiten Vektor werden auch die entstandenen Zeitfenster zwischen einzelnen Jobs belegt. Durch Abgleich der zwei Vektoren sind die Stillstandzeiten je Ressource bekannt.

Die gesamte Zielfunktion (10) setzt sich aus den normalisierten und gewichteten Anteilen – Startzeitminimierung, Gesamtbearbeitungszeit, Kosten für Bearbeitungszeiten, Stillstandzeiten und Rüstzeiten – zusammen.

Es werden Änderungen an den Optionen des genetischen Algorithmus durchgeführt, um den Suchraum realistisch zu minimieren und Rechenzeit zu begrenzen.

Angepasst wurden die Werte folgender Optionen, bezugnehmend auf die Standardangaben von MATLAB (2022):

- *MaxStallGenerations* auf 5 [Standard 50]: Mit diesem Wert wird gesteuert, über wie viele Schritte der genetische Algorithmus prüft, ob er Fortschritte macht. Für die realistischen Dimensionen des Arbeitsplatzes (bis zu 10 Ressourcen und 26 Jobs) ist der angepasste Wert ausreichend.
- *UseParallel* auf *true* [statt *fals*]: Hiermit wird die parallele Anwendung von mehreren Prozessoren erlaubt.
- *Display* auf *iter* [statt *final*]: Hiermit werden Iterationsbewertungen angezeigt.
- *MaxTime* auf 600‘‘ [statt *inf*]: Hiermit wird nach maximal 600 Sekunden die letzte Iteration als Lösung angenommen.
- *PopulationSize* auf 50 [50 für ≤5 Optimierungsvariablen | Ressourcen und 200 für mehr Optimierungsvariablen | Ressourcen]: Nach einigen Versuchen mit 4 bis 8 Ressourcen konnte keine Verbesserung durch eine größere Bevölkerung erzielt werden, jedoch wurden wesentlich längere Rechenzeiten festgestellt.
- *MaxGenerations* auf 40 [statt 100*Anzahl Optimierungsvariablen | Ressourcen]: Nach Testläufen konnte festgestellt werden, dass nicht mehr als 20 Generationen benötigt wurden, um Ergebnisse zu erbringen.

6 Resultate und Auswertung

6.1 Resultate der implementierten Algorithmen

In diesem Abschnitt werden anhand des Fallbeispiels die Ergebnisse der jeweiligen Algorithmen präsentiert.

Das Fallbeispiel betrachtet – wie bereits in Kapitel 5 erwähnt – einen Arbeitsplatz mit 7 Jobs (exklusive Dummy-Job), für welche jeweils 3 Menschen und 3 Roboter zur Verfügung stehen.

Vom Anwender werden über die Excel-Datei die notwendigen Bearbeitungszeiten der Jobs durch die Ressourcen Mensch und Roboter eingegeben, beziehungsweise werden auch die Vorgängerbeziehungen der Jobs eingepflegt.

Durch die Unterscheidung in die Ressourcenarten Mensch und Roboter besteht hier die Möglichkeit, die jeweiligen Kosten – in Abhängigkeit von Erfahrungswerten – zu definieren.

Ebenfalls einzupflegen sind die Gewichtungsfaktoren für die Teilziele (11) – (13) und auch die Rüstzeiten für die Ressourcen, wobei letztere nur in Abschnitt 6.1.3 zum Einsatz kommen.

Die hier angewandten Eingaben sind in Abschnitt 8.2 dargestellt.

6.1.1 Lineare Optimierung

Mit der angewendeten Struktur der Optimierungsvariable – wie bei der Implementierung in Abschnitt 6.1.1 erläutert – ist ein ressourcenschonender Umgang nicht möglich.

Konkret heißt das, dass bei der Entscheidung zwischen zwei Ressourcen, welche gleich viel kosten, nicht berücksichtigt wird, ob eine der zwei bereits eingesetzt wird oder nicht. Wenn entsprechend beide Ressourcen verfügbar sind, ist es für das lineare Programm irrelevant, welche der zwei Ressourcen gewählt wird, solange Vorgängerbeziehungen und Verfügbarkeit nicht verletzt werden. Eine Folge kann somit sein, dass durch die Anteile der Zielfunktion mehr Ressourcen eingebunden werden, als mit Hausverstand sinnvoll wäre, wie im vorliegenden Fallbeispiel ersichtlich ist.

Dies könnte einerseits behoben werden, indem die Zielfunktion nichtlineare Anteile enthalten würde, dies ist jedoch nicht im Sinne der hier betrachteten Aufgabenstellung.

Ein weiterer Lösungsansatz könnte darin bestehen, weitere Optimierungsvariablen zu erstellen, wie von Kalvelagen (2016) erklärt, jedoch wurde diese Option nicht weiter in

Betracht gezogen, da der dafür nötige Aufwand für die Implementierung nicht gerechtfertigt ist.

Um auch in der linearen Optimierung Rüstzeiten zu berücksichtigen und somit die übermäßige Einbindung von Ressourcen zu reduzieren, sind ebenfalls nichtlineare Anteile oder zusätzliche Optimierungsvariablen notwendig und davon wird ebenfalls in dieser Arbeit abgesehen.

Der Solver *intlinprog* von MATLAB verwendet Heuristiken, um den Lösungsraum zu verkleinern. Über die Optionen des Abrufs kann deren Einsatz unterdrückt werden, dies führt jedoch zu erhöhten Rechenzeiten und gleichbleibendem Ergebnis der Zielfunktion.

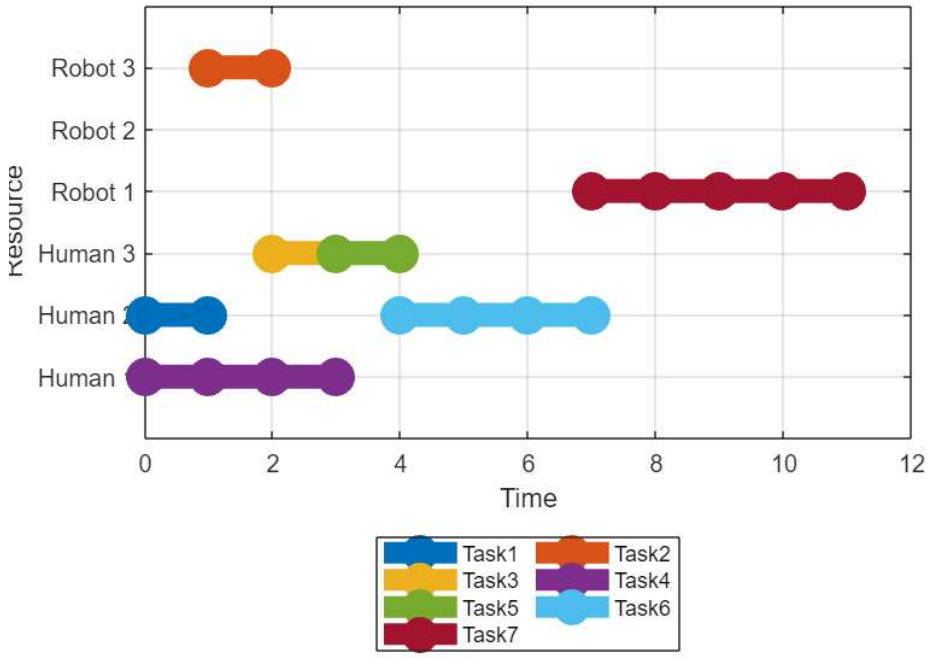

Abbildung 3: Zeitplan Fallbeispiel, gelöst mit ganzzahliger linearer Optimierung

In Tabelle 4 ist die Lösung der ganzzahligen linearen Optimierung dargestellt. In der ersten Spalte sind die Indices der Optimierungsvariable aufgelistet. Die zweite bis vierte Spalte dien dem Leser zur Erläuterung, welcher Job von welcher Ressource mit welcher Startzeit durch diese Indices beschrieben werden. In der fünften Spalte werden die Kosten, also die Werte der jeweiligen Zielfunktion für den entsprechenden Index, angeführt und in der letzten Spalte sind die jeweiligen Bearbeitungszeiten durch die ausgewählte Ressource ersichtlich.

Als letzter Job, hier 8, wird auch der Dummy-Job dargestellt, welcher notwendig ist, um die maximale Bearbeitungszeit zu ermitteln. Die in dieser Zeile angeführten Kosten setzen sich aus den Anteilen der Zielfunktionen der Startzeitminimierung (12) und der Bearbeitungszeit C_{max} (13) zusammen.

Es fallen keine Bearbeitungskosten an, da der Job eine Bearbeitungszeit gleich Null hat und Stillstand- beziehungsweise Rüstzeiten in dieser Implementierung nicht enthalten sind.
In der grafischen Darstellung des Zeitplans wird für das erleichterte Verständnis auf die Anzeige des Dummy-Jobs verzichtet.

Die Rechenzeit für die lineare Optimierung von 7 Jobs und 6 verfügbaren Ressourcen liegt bei **123,76 Sekunden**.

Die ermittelte Zielfunktion hat den Wert **2,61**. Dieser Wert kann in weiterer Folge nur mit den Ergebnissen aus Abschnitt 6.1.2 verglichen werden, da dieselbe Zielfunktion optimiert wird.

Tabelle 4: Lösung Fallbeispiel mit ganzzahliger linearer Optimierung

	ix	task	resource	time	cost	processingTimes
1	20	1	2	0	-0.2016	1
2	211	2	6	1	-0.1606	1
3	269	3	3	2	-0.1665	1
4	343	4	1	0	0.3360	3
5	498	5	3	3	-0.1489	1
6	594	6	2	4	0.4061	3
7	749	7	4	7	0.8214	4
8	905	8	6	11	1.7193	0

Zum Vergleich wird an dieser Stelle die Umsetzung der Zielfunktion und Nebenbedingungen nicht geändert. Anstatt des Abrufs *intlinprog* (für die ganzzahlige lineare Optimierung) wird der Abruf *ga* für den genetischen Algorithmus mit MATLAB Standardeinstellungen durchgeführt, wobei die Verwendung der Parallelverarbeitung gestattet wird.
Nach einer Laufzeit von 60 Minuten wurde die Ausführung gestoppt, da kein Ergebnis geliefert werden konnte.

Um einen Richtwert zu erzeugen, wird dieselbe Abfrage erneut durchgeführt, jedoch mit der zusätzlichen Option *MaxTime*, welche nach jeder Iteration einen Stopp erzwingen würde, sofern diese in der eingegebenen Laufzeit eintritt. Der Laufzeitwert wird mit 600 Sekunden festgelegt.
Dieser Versuch scheitert jedoch ebenfalls, was heißt, dass innerhalb von 10 Minuten keine Iteration abgeschlossen werden kann.

In einem dritten und letzten Versuch wird eine weitere Option *PopulationSize* hinzugenommen, mit welcher die Bevölkerungsgröße abweichend von MATLABs Standardwert beschränkt wird.
Standardmäßig wird im *ga* für gemischt ganzzahlige Optimierungsaufgaben eine Bevölkerungsgröße gemäß Gleichung (28) angenommen. Die Optimierungsvariable

[nvars] in Gleichung (28) ist im vorliegenden Fall gleich ix_{max} aus Gleichung (27), also *798*. Somit wird von MATLAB im vorliegenden Fall der Standardwert 100 für die Bevölkerungsgröße angewandt.
In diesem letzten Versuch wird dieser Wert von 100 auf 20 reduziert, jedoch findet trotzdem keine vollständige Iteration im Zeitraum von 10 Minuten statt.

$$\{\min(\max(10 * \text{nvars}, 40), 100)\} \tag{28}$$

Somit konnte veranschaulicht werden, wieso lediglich die Anwendung eines anderen Solvers nicht ausreicht, um die Vorteile des genetischen Algorithmus zu nutzen, sondern dass die Struktur der Optimierungsvariablen geändert werden musste.

6.1.2 Lineare Optimierung mit for-Schleife und genetischem Algorithmus

Die Änderungen an der implementierten ganzzahligen linearen Optimierung sind in Abschnitt 5.2 dargestellt.

Die Problematik in Bezug auf den schonenden Umgang mit den verfügbaren Ressourcen besteht weiterhin, da die Implementierung weiterhin im Kern dieselben Möglichkeiten und Einschränkungen mit sich bringt. Es besteht nicht die Möglichkeit, Stillstandzeiten oder Rüstzeiten zu berücksichtigen, sodass für den Algorithmus eine verfügbare Ressource – unabhängig ihrer weiteren Verwendung – denselben „Wert“ hat.

Um diese Beeinträchtigung etwas abzufangen, wird dem Hauptteil des Algorithmus aus 5.1 eine for-Schleife beziehungsweise alternativ dazu ein genetische Algorithmus zur Ressourcenfindung vorgelagert. Dieser vorgelagerte Abschnitt liefert dem Hauptteil der Implementierung Ressourcenkombinationen für die Jobzuweisung, alle weiteren Schritte bleiben unverändert.

for-Schleife

Unter Anwendung einer for-Schleife werden die erzeugten Ressourcenkombinationen an die lineare Optimierung übertragen. Für dieselben Angaben des Fallbeispiels – 7 Jobs und 6 Ressourcen (jeweils 3 Menschen und 3 Roboter) – konnte in einer Laufzeit von **2208,9 Sekunden**, also knappe 37 Minuten, die optimale Lösung mit dem Wert **2,61** der Zielfunktion ermittelt werden.

Der Zeitunterschied zur Rechenzeit in 6.1.1 lässt sich damit erklären, dass *intlinprog* selbst Heuristiken anwendet, um den Lösungsraum zu reduzieren.
Im vorliegenden Fall wird *intlinprog* jedoch, wie aus (29) ersichtlich, 63 Mal aufgerufen, da die Kombination aus nur Nullen – also keine Ressource – nicht weiterverfolgt wird.

Entsprechend werden vom Solver Reduktionsschritte vorgenommen, dies erfolgt jedoch 63 Mal.

$$2^{Anzahl\ Ressourcen} - 1 = 2^6 - 1 \tag{29}$$

Der Output dieser Optimierung ist eine Tabelle mit den sortierten Ergebnissen der Zielfunktionen und der dazugehörigen Ressourcenkombination, siehe Tabelle 5.

Tabelle 5: Ergebnisse der linearen Optimierung mit vorgelagerter for-Schleife

	resourceSelections						resources	objective
1	1	1	1	1	1	1	"Human 1 Human 2 Human 3 Robot 1 Robot 2 Robot 3"	2.6053
2	0	1	1	0	1	1	"Human 2 Human 3 Robot 2 Robot 3"	2.6053
3	0	1	1	1	0	1	"Human 2 Human 3 Robot 1 Robot 3"	2.6053
4	0	1	1	1	1	0	"Human 2 Human 3 Robot 1 Robot 2"	2.6053
5	0	1	1	1	1	1	"Human 2 Human 3 Robot 1 Robot 2 Robot 3"	2.6053
6	1	0	1	0	1	1	"Human 1 Human 3 Robot 2 Robot 3"	2.6053
7	1	0	1	1	0	1	"Human 1 Human 3 Robot 1 Robot 3"	2.6053
8	1	0	1	1	1	0	"Human 1 Human 3 Robot 1 Robot 2"	2.6053
9	1	0	1	1	1	1	"Human 1 Human 3 Robot 1 Robot 2 Robot 3"	2.6053
10	1	1	0	0	1	1	"Human 1 Human 2 Robot 2 Robot 3"	2.6053
11	1	1	0	1	0	1	"Human 1 Human 2 Robot 1 Robot 3"	2.6053
12	1	1	0	1	1	0	"Human 1 Human 2 Robot 1 Robot 2"	2.6053
13	1	1	0	1	1	1	"Human 1 Human 2 Robot 1 Robot 2 Robot 3"	2.6053
14	1	1	1	0	1	1	"Human 1 Human 2 Human 3 Robot 2 Robot 3"	2.6053
15	1	1	1	1	0	1	"Human 1 Human 2 Human 3 Robot 1 Robot 3"	2.6053
16	1	1	1	1	1	0	"Human 1 Human 2 Human 3 Robot 1 Robot 2"	2.6053
17	0	1	1	0	0	1	"Human 2 Human 3 Robot 3"	2.6053
18	0	1	1	0	1	0	"Human 2 Human 3 Robot 2"	2.6053
19	0	1	1	1	0	0	"Human 2 Human 3 Robot 1"	2.6053
20	1	0	1	0	0	1	"Human 1 Human 3 Robot 3"	2.6053
21	1	0	1	0	1	0	"Human 1 Human 3 Robot 2"	2.6053
22	1	0	1	1	0	0	"Human 1 Human 3 Robot 1"	2.6053
23	1	1	0	0	0	1	"Human 1 Human 2 Robot 3"	2.6053
24	1	1	0	0	1	0	"Human 1 Human 2 Robot 2"	2.6053
25	1	1	0	1	0	0	"Human 1 Human 2 Robot 1"	2.6053
26	1	1	1	0	0	1	"Human 1 Human 2 Human 3 Robot 3"	2.6053
27	1	1	1	0	1	0	"Human 1 Human 2 Human 3 Robot 2"	2.6053
28	1	1	1	1	0	0	"Human 1 Human 2 Human 3 Robot 1"	2.6053
29	0	0	1	0	1	1	"Human 3 Robot 2 Robot 3"	2.6757
30	0	0	1	1	0	1	"Human 3 Robot 1 Robot 3"	2.6757
60	0	0	0	1	0	0	"Robot 1"	5.1319
61	0	0	1	0	0	0	"Human 3"	5.6563
62	0	1	0	0	0	0	"Human 2"	5.6563
63	1	0	0	0	0	0	"Human 1"	5.6563

Die ersten 28 Ergebnisse haben denselben Wert wie die Zielfunktion und es werden hierfür zwischen 3 und 6 Ressourcen eingesetzt.
Die Begründung hierfür ist dieselbe wie für die lineare Optimierung aus 6.1.1 – im Rahmen der implementierten linearen Optimierung ist es nicht möglich, die Ressourceneffizienz in der Zielfunktion darzustellen.
Die – aus Sicht des Planers/der Planerin – beste Option ist eine mit einer minimalen Anzahl an Ressourcen (später unterstützt durch Berücksichtigung von Rüstzeiten) und minimierten Stillstandzeiten (zwischen bereits geplanten Jobs).

Nach Anwendung der linearen Optimierung verbleiben 9 von 28 Lösungen mit jeweils 3 eingesetzten Ressourcen, welche dasselbe Ergebnis von Zielfunktion und

Jobzuweisung haben. Diese sind alle möglichen Kombinationen von 2 Menschen und einem Roboter, wie in Abbildung 4 ersichtlich. Dieses Ergebnis wurde auch mit der Implementierung aus 5.1 erzielt.
Hiermit wird bestätigt, dass die lineare Optimierung die optimale Lösung der Zielfunktion findet. Durch die lange Laufzeit und die Unmöglichkeit, Teile der gewünschten Zielfunktion zu implementieren, stellt sie jedoch keine zufriedenstellende Option dar.

Abbildung 4: Alle Lösungen der linearen Optimierung mit vorgelagerter for-Schleife

vorgelagerter genetischer Algorithmus

Bei der Anwendung des genetischen Algorithmus, dessen binäre Optimierungsvariablen die Anzahl verfügbarer Ressourcen darstellen, wird dieselbe Lösung **2,61** der Zielfunktion in **1913,6 Sekunden** – also knapp 32 Minuten – ermittelt.

Wie auch vorhin anhand der for-Schleife beschrieben, ist dies ein Ansatz, welcher nicht weiterverfolgt wird.

6.1.3 Doppelter genetischer Algorithmus

Die letzte Implementierung erfolgt mit einem verschachtelten genetischen Algorithmus, wie in 5.3 beschrieben.
In diesem sind alle gewünschten Anteile (11) – (13) der Zielfunktion (10) enthalten.

Der Datenimport ist derselbe wie bislang, zusätzlich werden an dieser Stelle aber auch sowohl die Gewichtungen für die Anteile der Zielfunktion als auch die Rüstzeit berücksichtigt.

Im vorliegenden Fallbeispiel haben alle Anteile der Zielfunktion dieselbe Gewichtung von 0,33. Die Rüstzeit wird für die Ressource Mensch und Roboter gleich 2,5 Zeiteinheiten angenommen. Diese Eingaben können über den Excel-Datenimport geändert werden.

Nach einer Rechenzeit von **289,42 Sekunden** ist die Optimierung mittels genetischem Algorithmus für 7 Jobs und 6 Ressourcen (jeweils 3 Menschen und 3 Roboter) abgeschlossen.

Das Ergebnis der Zielfunktion ist **4,59** wobei darauf hingewiesen werden muss, dass dieses nicht mit den vorherigen Ergebnissen in 6.1.1 und 6.1.2 vergleichbar ist, da hier auch die Anteile der Stillstand- und Rüstzeiten berücksichtigt werden.

Der Zeitplan dieser Optimierung ist in Abbildung 5 ersichtlich.

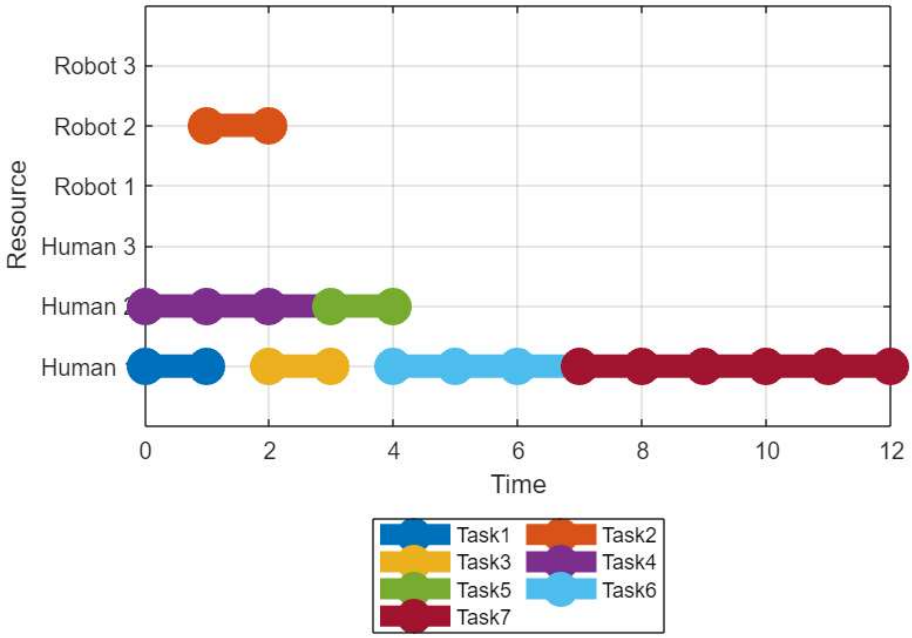

Abbildung 5: Zeitplan Optimierung mit genetischem Algorithmus

Um einen Vergleich mit der linearen Optimierung zu ermöglichen, wird an dieser Stelle auch ein Testlauf mit der vereinfachten Form der Zielfunktion durchgeführt. Dafür werden die Rüst- und Stillstandzeiten in den jeweiligen Anteilen (11) und (13) aus der Implementierung entfernt, sodass die entsprechenden vereinfachten Formen (24) und (25) verwendet werden.
Die Änderungen am Algorithmus werden nicht im Anhang dargestellt, da diese Vereinfachung des genetischen Algorithmus nur zum Vergleich der Laufzeiten und Ergebnisse der Zielfunktion für das vorliegende Fallbeispiel dienen soll, jedoch nicht

mit den Ergebnissen der systematischen Literaturrecherche und dem Ansinnen dieser Arbeit im Einklang stehen.

Mit der Vereinfachung des genetischen Algorithmus für das vorliegende Fallbeispiel wurde innerhalb **142,57 Sekunden** ein Wert der Zielfunktion von **2,61** ermittelt. Der dazugehörige Zeitplan ist in Abbildung 6 ersichtlich.

Wie zu erwarten, findet auch in diesem Fall kein ressourcenschonender Umgang statt und es werden 3 Menschen und 2 Roboter eingesetzt.

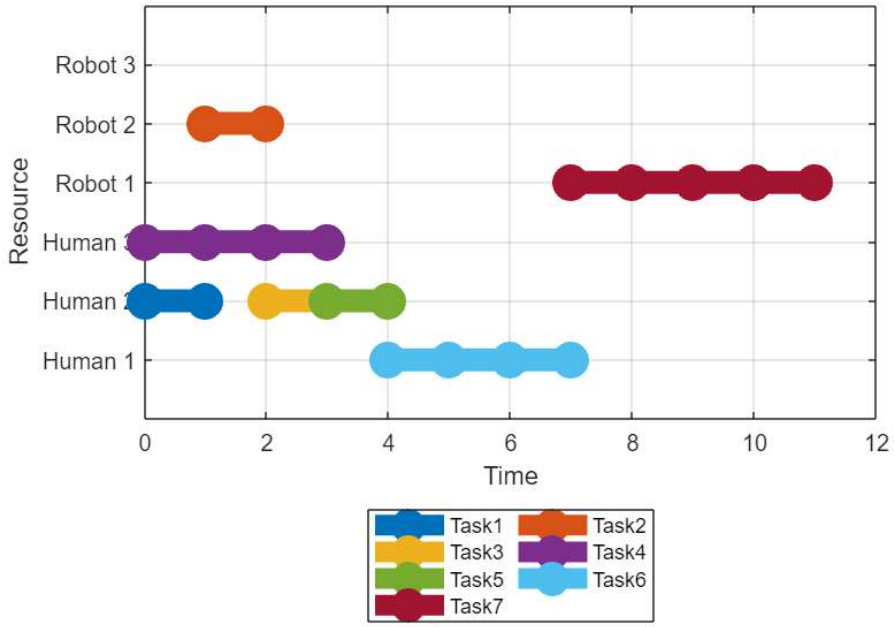

Abbildung 6: Zeitplan genetischer Algorithmus mit vereinfachter Zielfunktion

6.2 Vergleich der implementierten Algorithmen

In diesem Abschnitt werden die implementierten Algorithmen zuerst am Fallbeispiel verglichen und anschließend werden sie in Bezug auf die Forschungsfragen getestet.

6.2.1 Vergleich der Ergebnisse am Fallbeispiel

Im Abschnitt 6.1 wurden anhand eines gewählten Fallbeispiels sowohl die Ergebnisse des exakten als auch des metaheuristischen Lösungsverfahrens erläutert.
In Tabelle 6 sind die Ergebnisse zusammengefasst.

An dieser Stelle wird darauf hingewiesen, dass in der Implementierung des exakten Optimierungsverfahrens nicht alle Bestandteile der Zielfunktion umgesetzt sind, sondern vereinfachte Formen zum Einsatz kommen.

Für die Anteile der Zielfunktion werden – für das vereinfachte Verständnis – sowohl die gewünschten ((11) und (13)) als auch die vereinfachten Formeln ((24) und (25)) angeführt, welche für die Implementierung der linearen Optimierung notwendig sind.

Um einen Vergleich der Ergebnisse der linearen Optimierung mit jenen des doppelten genetischen Algorithmus zu ermöglichen, wurde die Zielfunktion insofern vereinfacht, als dass sie mit jener der linearen Optimierung übereinstimmt.
Die Ergebnisse dieses zusätzlichen Testlaufs sind in der vorletzten Zeile der Tabelle 6 mit einem Asterisk versehen.

Tabelle 6: Vergleich der Ergebnisse der Optimierungsverfahren am Fallbeispiel

	Implementierte Anteile der Zielfunktion					Ergebnis Zielfunktion	Rechenzeit [Sekunden]
	F1		F2	F3			
	(11)	(24)	(12)	(13)	(25)		
lineare Optimierung		●	●		●	2,61	123,76
lineare Optimierung mit for-Schleife		●	●		●	2,61	2.208,90
lineare Optimierung mit vorgelagertem genetischen Algorithmus		●	●		●	2,61	1.913,60
Doppelter genetischer Algorithmus*		●	●		●	2,61	142,57*
Doppelter genetischer Algorithmus	●		●	●		4,59	289,42

Das Fallbeispiel ist mit 7 Jobs und 6 Ressourcen (3 Menschen und 3 Roboter) absichtlich am Grenzwert sinnvoller Rechenzeiten zwischen der exakten und metaheuristischen Optimierung ausgelegt.

Es ist ersichtlich, dass die lineare Optimierung 6.1.1 für einen kleinen Arbeitsplatz das minimale Optimum der Zielfunktion in einer zumutbaren Zeit findet. Im Vergleich zum genetischen Algorithmus mit der vereinfachten Zielfunktion kann eine ähnliche Rechenzeit festgestellt werden. Es ist anzumerken, dass bei wiederholtem Testlauf auf derselben Arbeitsumgebung trotzdem leicht abweichende Rechenzeiten entstehen können.
Bei dem Vergleich der Implementierung mittels for-Schleife mit jener durch vorgelagerten genetischen Algorithmus (6.1.2) kann festgestellt werden, dass die optimale Lösung gefunden wird, wobei das Ergebnis bei dem Einsatz des genetischen Algorithmus schneller eintritt. Diese Feststellung überrascht jedoch nicht, da hier die Vorteile des genetischen Algorithmus zum Einsatz kommen.
Wenn die implementierten genetischen Algorithmen (mit der gewünschten beziehungsweise vereinfachten Zielfunktion) miteinander verglichen werden (6.1.3), ist erkennbar, dass durch die Berechnung einer komplexeren (der gewünschten) Zielfunktion mehr Rechenzeit benötigt wird.

6.2.2 Vergleich der Algorithmen in Bezug auf die Problemstellung

An dieser Stelle wird die lineare Optimierung aus 5.1 im Vergleich zu dem in 5.3 beschriebenen doppelten genetischen Algorithmus getestet. Der genetische Algorithmus berechnet die Zielfunktion (10) mit ihren Bestandteilen (11) bis (13), da diese das Kernelement dieser wissenschaftlichen Arbeit darstellen. Die lineare

Programmierung dient entsprechend als Vergleichswert hinsichtlich der benötigten Rechenzeiten.

Die Varianten der linearen Optimierung mit for-Schleife und genetischem Algorithmus werden nicht weiterverfolgt, da die erzielten Rechenzeiten im Fallbeispiel nicht zufriedenstellend sind.

Die Testläufe wurden auf einem Intel(R) Core (TM) i7-7700HQ CPU @ 2.80GHz 2.81 GHz mit 16GB RAM und x64 basierten Prozessor durchgeführt.

Lineare Optimierung

In Tabelle 9 sind die Rechenzeiten der linearen Optimierung mit steigender Anzahl an durchzuführenden Jobs beziehungsweise hierzu verfügbarer Ressourcen dargestellt. Bei einer ungeraden Anzahl an Ressourcen ist ein Menschen mehr als die Anzahl an Robotern verfügbar.

Im vorliegenden Testlauf wird mit steigender Anzahl der durchzuführenden Jobs in entgegengesetzter Richtung (also von unten nach oben) eine neue Zeile hinzugefügt (Tabelle 7).

Dasselbe Prinzip gilt auch für die Vorgängermatrix (Tabelle 8). Dieses Vorgehen wurde für die Erstellung der Matrizen für die Testläufe gewählt, um immer einen vordefinierten Dummy-Job zu haben.

Tabelle 7: Bearbeitungszeiten für 11 einzuplanende Jobs

	Humans	Robots
Task 01	5	1
Task 02	3	3
Task 03	2	3
Task 04	3	1
Task 05	1	3
Task 06	5	1
Task 07	1	2
Task 08	3	3
Task 09	1	1
Task 10	3	3
Task 11	5	4
Σ	32	25

Entsprechend kann man sich – falls gewünscht – verkleinerte Arbeitsumgebungen von der unteren rechten Ecke der Tabellen Tabelle 7 und Tabelle 8 ermitteln.

Tabelle 8: Vorgängermatrix für 11 einzuplanende Jobs

	Task 01	Task 02	Task 03	Task 04	Task 05	Task 06	Task 07	Task 08	Task 09	Task 10	Task 11
Task 01	0	0	0	0	0	0	0	0	0	0	0
Task 02	1	0	0	0	0	0	0	0	0	0	0
Task 03	1	1	0	0	0	0	0	0	0	0	0
Task 04	1	1	1	0	0	0	0	0	0	0	0
Task 05	0	0	0	0	0	0	0	0	0	0	0
Task 06	1	1	1	1	1	0	0	0	0	0	0
Task 07	1	1	1	1	1	1	0	0	0	0	0
Task 08	0	0	0	0	0	0	0	0	0	0	0
Task 09	1	1	1	1	1	1	1	1	0	0	0
Task 10	1	1	1	1	1	1	1	1	1	0	0
Task 11	1	1	1	1	1	1	1	1	1	1	0

Wie aus Tabelle 9 für die lineare Optimierung ersichtlich, steigt die Rechenzeit bei steigender Anzahl an Jobs schnell an.

Tabelle 9: Rechenzeiten der linearen Optimierung bei zunehmender Größe des Arbeitsplatzes

		Rechenzeit [Sekunden]							
		Anzahl durchzuführender Jobs							
		4	5	6	7	8	9	10	11
Anzahl verfügbarer Ressourcen	3	4,3	3,8	8,2	15,2	39,6	86,3	169,2	419,3
	4	2,4	4,0	18,7	40,6	94,4	170,5	575,2	994,4
	5	4,8	7,9	34,7	74,5	150,4	341,5	1545,0	2174,7
	6	7,8	13,4	60,2	123,1	273,5	610,0	2039,9	2908,3
	7	9,8	24,4	109,3	104,0	433,8	930,9	3023,1	7202,1
	8	17,9	36,7	137,9	136,4	279,0	1288,9	7200,8	7202,5
	9	20,4	40,4	200,5	419,7	934,9	1955,4	7400,3	10451,2
	10	29,6	59,4	284,9	612,6	1205,7	2879,4	8300,6	14576,7
	11	50,2	110,4	393,8	754,0	1575,9	3155,8	8407,5	>18000,0
Rechenzeit in Minuten				0' - 5'	5' - 10'	10' - 20'	20' - 60'	60' - 180'	>180'

Die Anzahl der verfügbaren Ressourcen beeinflusst die Laufzeit nicht so stark. Bei den angewandten Vorgängerbeziehungen, mit denen die meisten Jobs nacheinander einzuplanen sind, sind (ab einem verfügbaren Ressourcenpool von 3 Menschen und 3 Robotern) auch keine großen Veränderungen in der Zuweisungsordnung der Jobs festzustellen.

Aus dieser Feststellung und auch aus der Tatsache heraus, dass maximal 11 Jobs betrachtet werden, wird der verfügbare Pool an Ressourcen nicht auf mehr als 11 Ressourcen erhöht, da dies nicht mehr sinnvoll scheint. Ebenfalls ist die Betrachtung von mehr als 11 Ressourcen, welche nur zwei Typenklassen angehören, nicht zielführend, wenn die Anzahl der Ressourcen jene der Jobs übersteigt.

Im Hinblick auf die Anzahl der einzuplanenden Jobs zeigt sich mit der linearen Optimierung schnell, dass die Rechenzeit eine kurzfristige Entscheidung innerhalb 10 Minuten nicht mehr unterstützen würde.

An einem Arbeitsplatz stellen 10 Jobs und 5 verfügbare Ressourcen eine realistische Arbeitsgröße dar. Für diesen Arbeitsplatz wäre aber keine schnelle Ergebnisfindung mit der linearen Optimierung möglich, da die Rechenzeit bereits 25‘ beträgt.
Ein weiterer Nachteil ist bei der Einschränkung bezüglich der Größenordnung der Bearbeitungszeiten festzustellen. Die Testläufe werden mit einstelligen Bearbeitungszeiten durchgeführt. Bei steigenden Bearbeitungszeiten steigt auch die mögliche Anzahl an Indices der Optimierungsvariablen rasant an, wie aus der Formel (26) zu entnehmen ist. Somit können Arbeitsumgebungen, deren Bearbeitungszeiten sehr stark variieren und nicht auf einen gemeinsamen Nenner skaliert werden können, nicht mit der linearen Optimierung in zumutbarer Zeit iteriert werden.

Doppelter genetischer Algorithmus

Für die Testläufe des doppelten genetischen Algorithmus gelten die Eingaben der Tabellen Tabelle 10 und Tabelle 11. Die Matrix der Vorgänger kann für weniger als 20 Jobs von unten nach oben beziehungsweise nach links skaliert werden. Dieses Format der Matrix wurde für die Testläufe gewählt, um immer einen Dummy-Job für die Implementierung als Referenz zu haben. Dieses Vorgehen ist bei einer Anwendung durch den Planer/die Planerin nicht notwendig.

Tabelle 10: Vorgängermatrix für 20 einzuplanende Jobs

	Task 01	Task 02	Task 03	Task 04	Task 05	Task 06	Task 07	Task 08	Task 09	Task 10	Task 11	Task 12	Task 13	Task 14	Task 15	Task 16	Task 17	Task 18	Task 19	Task 20
Task 01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 02	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 03	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 04	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 05	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 06	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 07	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 08	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 09	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 10	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Task 11	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
Task 12	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
Task 13	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
Task 14	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Task 15	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
Task 16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
Task 17	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
Task 18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
Task 19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
Task 20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Dasselbe Prinzip der Skalierung gilt an dieser Stelle auch für die Bearbeitungszeiten der Tabelle 11. Bei dieser Tabelle wird die entsprechende Anzahl an Zeilen von oben entfernt.

Tabelle 11: Bearbeitungszeiten bei 20 einzuplanenden Jobs

	Humans	Robots
Task 01	5	3
Task 02	4	3
Task 03	1	4
Task 04	1	3
Task 05	2	4
Task 06	5	1
Task 07	1	1
Task 08	3	4
Task 09	4	1
Task 10	5	1
Task 11	3	3
Task 12	2	3
Task 13	3	1
Task 14	1	3
Task 15	5	1
Task 16	1	2
Task 17	3	3
Task 18	1	1
Task 19	3	3
Task 20	5	4
Σ	58	49

In Tabelle 12 sind ausgewählte Rechenzeiten für den doppelten genetischen Algorithmus veranschaulicht. In dieser werden für die zu planenden Jobs maximal die selbe Anzahl an Ressourcen berücksichtigt, da im Fallbeispiel nur die zwei Typen Mensch und Roboter betrachtet werden. Weiters werden maximal 5 Menschen und 5 Roboter bereitgestellt, da davon auszugehen ist, dass bei 20 Jobs nicht eine größere Mannschaft benötigt wird.

Es ist ersichtlich, dass der genetische Algorithmus Schwierigkeiten bei der Berechnung der Zielfunktion mit einer geringen Anzahl an verfügbaren Ressourcen oder Jobs hat, da der Algorithmus eine größere Ausgangsbevölkerung benötigt, um aus dieser weitere Generationen erzeugen zu können.

Im Vergleich dieser Ergebnisse mit jenen des linearen Algorithmus kann festgestellt werden, dass mit steigender Arbeitsplatzgröße der lineare Algorithmus rasch ansteigende Rechenzeiten beansprucht, während der genetische Algorithmus noch immer unter dem Schwellwert von 15‘ iteriert.

Tabelle 12: Rechenzeiten für den doppelten genetischer Algorithmus bei zunehmender Größe des Arbeitsplatzes

		Rechenzeit [Sekunden]											
		Anzahl durchzuführender Jobs											
		4	5	6	7	8	9	10	12	14	16	18	20
Anzahl verfügbarer Ressourcen	3	549,6	351,2	146,8	145,5	286,1	439,9	309,0	197,8	941,6	446,9	1233,2	1245,7
	4	548,4	346,7	202,2	216,2	129,0	388,9	370,7	681,7	505,9	715,2	946,5	1216,0
	5		238,5	323,5	222,3	370,8	303,0	254,8	975,5	543,2	469,3	813,7	1053,9
	6			245,5	159,5	271,4	326,7	250,4	249,5	429,7	483,1	795,8	936,0
	7				425,9	289,5	358,2	253,1	271,9	592,8	563,5	797,4	876,3
	8					207,9	283,8	447,4	333,5	422,2	706,1	804,0	712,9
	9						799,3	448,8	631,2	938,6	968,3	1214,8	1243,2
	10							338,1	428,6	537,2	706,8	1236,6	1142,4
Rechenzeit in Minuten				0' - 5'	5' - 10'	10' - 20'	20' - 60'	60' - 180'	>180'				

Wenn der Fall von 9 zu planenden Jobs und 7 verfügbaren Ressourcen betrachtet wird, ist ersichtlich, dass der genetische Algorithmus eine Rechenzeit von 358‘‘ (also knapp 6‘, Tabelle 12) benötigt, während die lineare Optimierung bereits den Schwellwert von 15‘ überschreitet (Tabelle 9).

Aus den Zielwerten in Tabelle 11 kann in der gegebenen Form, Aufbereitung und Zusatzinformation nur eine Aussage zur Funktionsweise des genetischen Algorithmus getroffen werden. Aus der Auswertung ist erkennbar, dass die Werte der Zielfunktion für dieselbe Anzahl an Jobs und variierender Anzahl an Ressourcen bei den Testläufen unterschiedliche Werte annehmen. Dies kann damit begründet werden, dass der genetische Algorithmus sich optimalen Werten nähert, jedoch nicht unbedingt einen erreicht. Dieser Nachteil einer fehlenden optimalen Lösung der Zielfunktion kommt jedoch mit dem Vorteil besserer Rechenzeiten einher.

Tabelle 13: Werte der Zielfunktion bei Anwendung des doppelten genetischen Algorithmus

	Wert Zielfunktion											
	Anzahl durchzuführender Jobs											
Anzahl verfügbarer Ressour	4	5	6	7	8	9	10	12	14	16	18	20
3	5,4	5,8	4,7	4,9	4,9	5,8	7,5	9,6	9,7	12,0	13,7	18,1
4	4,5	5,0	4,1	5,2	5,0	5,3	6,3	7,1	9,0	8,6	11,6	15,4
5		4,7	4,0	4,6	4,4	5,1	6,4	6,0	7,3	9,5	11,4	13,4
6			4,1	5,1	4,9	5,2	6,2	6,5	7,7	8,4	12,3	11,8
7				4,9	4,6	5,3	6,1	6,6	6,1	8,3	10,5	11,0
8					5,0	5,6	6,0	6,3	7,9	8,0	8,4	10,4
9						5,6	6,3	6,1	7,1	7,7	8,6	10,2
10							6,6	6,4	7,4	7,7	8,2	10,0

Ein weiterer Vorzug des genetischen Algorithmus ist, dass eine Rechenzeit festgelegt werden kann *[MaxTime]*, innerhalb deren iteriert wird. Nach Ablauf dieser Zeit erhält man einen Wert in der Nähe eines Optimums. Diese Option besteht in der linearen Optimierung mit *intlinprog* nicht, sodass immer gewartet werden muss, bis der optimale Wert berechnet wird, um ein Ergebnis zu haben.

Anhand der durchgeführten Testläufe wird empfohlen, dass **ab einer Anzahl von 9 zu planenden Jobs** der genetische Algorithmus angewendet wird, um in einer zumutbaren Zeit (deutlich unter 20‘) ein Ergebnis ermitteln zu können.

6.3 Resultate in Bezug auf das Forschungsziel und die Forschungsfragen

In der Problemstellung dieser Arbeit wurde angestrebt, den Vergleich zwischen einem exakten und einem metaheuristischen Optimierungsalgorithmus durchzuführen. Der Schwerpunkt war darauf zu legen, ab welchem Komplexitätsgrad die jeweilige

Methode bei der taktzeitoptimierten Aufgabenzuordnung zu bevorzugen ist und weshalb dies der Fall ist.

Um diese Problemstellung zu untergliedern, wurden im Abschnitt 1.2 drei Forschungsfragen aufgestellt.

1. Bis zu welcher Ressourcen- und Aufgabenanzahl kann zur taktzeitoptimierten Planung noch ein exaktes Optimierungsverfahren angewendet werden, das ein globales Minimum ermittelt?

Als erstes wurde eine Zielfunktion zur taktzeitoptimierten Aufgabenzuordnung in einem Mensch-Roboter-Arbeitssystem gesucht.
Hierzu wurde zuerst der theoretische Fachbereich eingegrenzt (2) und im Rahmen einer systematischen Literaturrecherche der aktuelle Wissensstand eruiert (3). Aus diesem Wissen wurde eine passende Zielfunktion mit Nebenbedingungen entwickelt (4).

2. Was sind in diesem Zusammenhang Entscheidungskriterien, um ein lokales Minimum als beste Lösung zu akzeptieren?

Die zweite Frage bezog sich auf die Größe des ausgelegten Arbeitsplatzes, für den noch mithilfe eines exakten Optimierungsverfahrens ein globales Minimum der Zielfunktion ermittelt werden kann.
Zur Beantwortung dieser Fragestellung wurde, soweit möglich, die präsentierte Zielfunktion in einem ganzzahlig linear Optimierungsalgorithmus implementiert (5.1). Getestet wurde in 6.2.2 mit steigender Anzahl an Ressourcen und Jobs, welche Rechenzeit benötigt wird um eine optimale Lösung zu erhalten.
Wie in Tabelle 9 ersichtlich, wird bei einer Arbeitsumgebung mit 9 zu planenden Jobs und 7 verfügbaren Ressourcen bereits eine Rechenzeit von circa 15‘ benötigt. Dies stellt einen Schwellenwert dar, um die Implementierung für eine kurzfristige Entscheidung nutzen zu können.

3. Was sind in diesem Zusammenhang Entscheidungskriterien, um ein lokales Minimum als beste Lösung zu akzeptieren?

Die letzte Frage bezieht sich auf Entscheidungskriterien, um ein lokales Minimum als Ergebnis zu akzeptieren.
Zur Beantwortung dieser Frage ist maßgeblich, welches Ziel der Planer/die Planerin verfolgt. Wenn die Implementierung für Entscheidungen bei Ressourcenausfall im laufenden Betrieb genutzt wird, dann ist bei einem Arbeitsplatz, welcher mehr als 9 Jobs beinhaltet, das metaheuristische Verfahren mit deutlich besseren Rechenzeiten von Vorteil. In dieser Situation ist das Ziel meist die Aufrechterhaltung der Produktion.

Wenn andererseits ein Arbeitsplatz geplant wird, dann könnte die Verwendung des exakten Optimierungsverfahrens – trotz langer Rechenzeiten – von Interesse sein. Unter diesen Voraussetzungen ist meist die Umsetzung zeitlich versetzt von der Ermittlung und entsprechend kann die Rechenzeit in Kauf genommen werden.

Die in Kapitel 1.3. geplanten Ziele wurden schrittweise in den Abschnitten der theoretischen Grundlagen, der systematischen Literaturrecherche und auch dem Praxisteil dieser Arbeit abgearbeitet.

7 Ausblick

Im Rahmen dieser Arbeit wurde ein taktzeitoptimierendes Werkzeug für die Ressourcenzuweisung erstellt, welches dem Shopfloor sowohl zur Auslegung des Arbeitsumfelds als auch für den laufenden Betrieb zur schnellen Umplanung der Ressourcen bereitgestellt werden kann.

In dieser Arbeit werden zwei Ressourcentypen berücksichtigt und dies wäre ein Aspekt, welcher weiter ausgebaut werden kann. Durch den Aufbau der Implementierung können mit einigen Anpassungen mehrere Ressourcenarten in Betracht gezogen werden, beispielsweise Mitarbeiter und Mitarbeiterinnen mit einer vollständigen Ausbildung und Lehrlinge und andererseits Roboter, welche Schweiß- oder Schraubarbeiten durchführen können. Wenn diese Unterteilung vorgenommen wird, können entsprechende Zeiten und durchführbare Tätigkeiten hinterlegt werden. Entsprechend können dann komplexe Arbeitssysteme modelliert und optimiert werden.

Für beide Ressourcen wird im doppelten genetischen Algorithmus dieselbe Rüstzeit angewandt. Dies könnte mit einigen Anpassungen am Code geändert werden, sodass jeder verfügbare Ressourcentyp eine andere Rüstzeit benötigt.
Weiters könnten Rüstzeiten auch ressourcenspezifisch konfiguriert werden, damit diese gleichzeitig mit anderen Jobs stattfinden können oder – wie derzeit implementiert – weiter einen pauschalen Zuschlag bei Einplanung der Ressource darstellen.

8 Anhang

8.1 Ergebnisse der systematischen Literaturrecherche

Tabelle 14: Übereinstimmung ausgewählter Arbeiten mit den Kriterien der systematischen Literaturrecherche

Schlüsselwörter									Eckdaten					semantische Auswahlkriterien			
genetic	heuristic & cycle time	task allocation	scheduling	human & robot	robot & multi-agent	Algorithmus	Titel	Autoren	Jahr	Journal	Institut	Land	Forschungsthema	taktzeitoptimierende Ermittlung	globales Ermittlung	lokales Ermittlung
●	◔	◔	◔	○	●	-	y and operation design of a robot logistics system for the hotel industry	Woo Jin Lee, Sung Il Kwag, Young Dae Ko	2019	Tourism Management	Department of Hotel and Tourism Management, College of Hospitality and Tourism, Sejong University, Seoul, Korea	Südkorea	○	●	●	○
●	●	●	●	○	●	Genetischer Algorithmus mit Tabu Suche	Multi-mobile robots and multi-trips feeding scheduling problem in smart manufacturing system: An d aenetic	Feng Yao, Yan-Jie Song, Zhong-Shan Zhang, Li-Ning Xing, Xin Ma, Xun-Jia Li	2019	International Journal of Advanced Robotic Systems	College of Systems Engineering, National University of Defense Technology, Changsha, Hunan, China	China	○	●	○	●
●	●	●	●	○	◔	Hybrider genetischer Algorithmus	Process sequencing for a pick-and-place robot in a real-life flexible robotic cell	Mazyar Ghadiri Nejad, Seyed Mahdi Shavarani, Hüseyin Güden, Reza Vatankhah Barenji	2019	International Journal of Manufacturing Technology	Department of Industrial Engineering, Eastern Mediterranean University, Famagusta, Turkey	Türkei	●	●	○	●
●	◔	●	●	○	◔	Nicht-dominant sortierender genetischer Algorithmus	Trade-off between process production cost in cyclic flexible robotic cells	Mazyar Ghadiri Nejad, Seyed Mahdi Shavarani, Béla Vizvári, Reza Vatankhah Barenji	2018	International Journal of Manufacturing Technology	Department of Industrial Engineering, Eastern Mediterranean University, Famagusta, Türkei	Türkei	●	●	○	●
●	◔	◔	◔	◔	◔	-	duling in human robot collaboration – a use embly of printed circuit boards	Karin Bogner, Ulrich Pferschy, Roland Unterberger, Herwig Zeiner	2018	International Journal of Production Research	Department of Statistics and Operation Research, University of Graz, Austria	Österreich	●	●	●	○

Eckdaten			Zielfunktion und Nebenbedingungen						
Titel	Autoren	Jahr	Thema der Arbeit	Zielfunktion	Erklärung der Zielfunktion	Nebenbedingungen	(2)	(3)	(4)	(5)
design of a robot logistics system for the hotel industry	Woo Jin Lee, Sung Il Kwag, Young Dae Ko	2019	Multimobile Roboter und Multi-Wege-Belieferung eines Hotels (Rommservice).	$\min \sum_{}^{K} \sum_{}^{N} \sum_{}^{M_n} j_{nm} + j'_{nm} - x_{nm}^{k} - y_{nm}^{k} - y'^{k}_{nm}$	Minimierung nicht erteilter Jobs im Rahmen eines Zyklus, indem von der Anzahl der abgegebenen Bestellungen zur Lieferung und Abholung die vergebenen Jobs des Bringens bzw. Abholens subtrahiert werden.	- Abschluss Durchführung Job während eines Zyklus - Es findet eine einzige Bestellung je Zimmer und Zyklus statt - Einschränkung, dass nur ein Roboter eine Bestellung bedient - Einschränkungen des Zusammenhanges zwischen Roboter und dual durchgeführten Jobs bzw. Abholungen (immer dual)	◔	○	○	●
Multi-mobile robots and multi-trips feeding scheduling problem in smart manufacturing system: An	Feng Yao, Yan-Jie Song, Zhong-Shan Zhang, Li-Ning Xing, Xin Ma, Xun-Jia Li	2019	Multimobile Roboter und Multi-Wege-Belieferung der Produktion mit Material.	$\min(\sum_{i=1}^{T} \sum_{=1}^{T} \sum_{k=1}^{M} \sum_{i=1}^{R} t_{ij} * x_{kl}^{ij} + \sum_{i=1}^{T} \sum_{=1}^{T} \sum_{k=1}^{M} \sum_{i=1}^{R} w_{ij} * x_{kl}^{ij})$	Minimierung der aufsumierten Kosten für Wegzeiten und Wartezeiten.	- Abschluss Durchführung des Jobs innerhalb des Zyklus - Roboter starten Zyklus aus Lager - Jede Lieferung wird nur ein Mal durchgeführt - Einschränkung der Transportkapazität der Roboter	●	○	○	●
Process sequencing for a pick-and-place robot in a real-life flexible robotic cell	Mazyar Ghadiri Nejad, Seyed Mahdi Shavarani, Hüseyin Güden, Reza Vatankhah Barenji	2019	Minimierung Zykluszeit Maschinen durch Optimierung Transportabfolgen des belieferndes Roboters.	$\min(\sum \sum d_{ab} * x_{ab} + \sum \sum w_{ab})$	Minimierung der Durchführungszeit eines Zyklus, inklusive der Wartezeiten des Roboters.	- Durchführung aller Jobs ein Mal - Ermittlung der Bearbeitungszeiten für die Jobs, inklusive der Wegzeiten, Be-/Entladezeiten bzw. der Wartezeiten - Keine Wartezeiten sind notwending, wenn es sich nicht um aufeinanderfolgende Tätigkeiten der Maschinen handelt - Bedingung, dass genug Bearbeitungsziet für die Maschine zwischen BE-/Entladen vorhanden ist	◕	◔	○	●
Trade-off between process in cyclic flexible robotic cells	Mazyar Ghadiri Nejad, Seyed Mahdi Shavarani, Béla Vizvári, Reza Vatankhah Barenji	2018	Minimierung Zykluszeit Zuführplanung von Maschinen durch Optimierung Transportabfolgen des belieferndes Roboters, als auch der Produktionskosten der Maschinen.	$\min[\sum_{i} \sum_{q} s_{iq} * p_q * c_q + c_0 \left(m \sum_{a \in A - L_1} t_{aL_1} - \sum_{i} \sum_{q} s_{iq} * p_q \right)]$ $\min(\sum_{a \in A - L_1} t_{aL_1})$	Die erste Zielfunktion minimiert die Produktionskosten der Maschinen unter Berücksichtigung der Bearbeitungskosten als auch der Stehzeiten. Die zweite Zielfunktion minimiert die Durchführungszeit eines Zyklus.	- Durchführung aller Jobs ein Mal - Keine Wartezeiten sind notwending, wenn es sich nicht um aufeinanderfolgende Tätigkeiten der Maschinen handelt - Die Wartezeit zur Entladung der Maschine darf nicht länger sein als die Bearbeitungszeit der Maschine - Mindestdauer Zwischen Startzeit beschränkt auf die Summe der Bearbeitungszeit der Maschine, des Roboters und der Wartezeit - Bedingung, dass genug Bearbeitungsziet für die Maschine zwischen Be-/Entladen vorhanden ist	●	◔	○	●
robot collaboration – a use circuit boards	Roland Unterberger, Herwig Zeiner	2018	Taktzeitoptimierte Aufgabenzuordnung in der Montage von Leiterplatten, in einer Mensch Roboter Kollaboration.	$\min \mathbf{C}_{max}$ $\mathbf{C}_{max} \geq$	Minimierung der maximalen Bearbeitungszeit.	- Durchführung aller Jobs ein Mal - Beschränkung der Ressourcen, einen Job gleichzeitig zu bearbeiten - Durchführung eines Jobs erst nach Abschluss all seiner Vorgänger - Abschluss der gegebenen Anzahl an "oder" Vorgängern vor Durchführung des Jobs, sodass ein Mindestabstand der Bearbeitungsplätze gewährleistet ist	●	◕	◕	●

8.2 Fallbeispiel

Tabelle 16: Eingaben zur Implementierung der MATLAB Algorithmen

no. tasks	7	max. 25
no. humans	3	
no. robots	3	
cost human	0,5906	€/min
cost robot	0,6422	€/min
set-up cost	2,5	
weight objective F1	0,33	
weight objective F2	0,33	
weight objective F3	0,33	

workdays	249	days/year
workhours/shift	8	hrs/shift
productivity humans	0,85	[1]
cost humans/year	60000	€/year
cost humans/minute	0,5906	€/min
productivity robots	0,95	[1]
aquisition price/robot	350000	€/year
depreciation years/robot	6	years
maintenance & service/robot	0,25	[1]
cost robot/minute	0,6422	€/min

Tabelle 17: Eingabe der Bearbeitungszeiten der Ressourcen für die jeweiligen Jobs

	Humans	Robots
Task 01	1	3
Task 02	5	1
Task 03	1	2
Task 04	3	3
Task 05	1	1
Task 06	3	3
Task 07	5	4
Σ	19	17

Tabelle 18: Eingabe der Vorgängermatrix für die jeweiligen Jobs

	Task 01	Task 02	Task 03	Task 04	Task 05	Task 06	Task 07
Task 01	0	0	0	0	0	0	0
Task 02	1	0	0	0	0	0	0
Task 03	1	1	0	0	0	0	0
Task 04	0	0	0	0	0	0	0
Task 05	1	1	1	1	0	0	0
Task 06	1	1	1	1	1	0	0
Task 07	1	1	1	1	1	1	0

8.3 Code MATLAB lineare Optimierung

Import

```
clear, rng default


% data import
dataFile = "Data_3.xlsx";


% import code is in a separate function
[preds, processingTimes, resourceNames, costsRaw, weightsObjectives] = …
import_data_from_excel_data3(dataFile);

% update min max for each objective part

minmax_updater(dataFile);
```

Problem Properties

```

% maximum possible time
maxPossibleTime = max(sum(processingTimes.Variables,1));
numberOfTasks = height(processingTimes);
numberOfResources = width(processingTimes);

% number of binary decision variables
nvars = numberOfTasks*maxPossibleTime*numberOfResources;
```

Index Table

```

% creating indexing table
ix = [];
index = 0;
for taskId = 1:numberOfTasks
    for resourceId = 1:numberOfResources
        for timeId = 1:maxPossibleTime
            index = index + 1;


            % assign properties
            ix(index).ix = index;
            ix(index).task = taskId;
            ix(index).resource = resourceId;
            ix(index).time = timeId;
        end
    end
end
ix = struct2table(ix);


% row numbers to variable ind for easier coding
ind = ix.ix;
```

Constraints

```
% empty row of zeros for the optimization problem definition matrices to ensure there are
enough (nvars) columns
zeroRow = zeros(0,nvars);
```

Constraint 1: Every task must start exactly once

```
% reset matrix
Aeq_every_task_once = zeroRow;
beq_every_task_once = [];


% summation of the decision variables for any taskId should be equal to one
for taskId = 1:numberOfTasks
    Aeq_every_task_once(taskId,ind(ix.task == taskId)) = 1;
    beq_every_task_once(taskId,1) = 1;
end
```

Constraint 2: Every resource can work on up to one task at a time

```
% reset matrix
Aineq_every_resource_once =
false(numberOfTasks*numberOfResources*maxPossibleTime*(numberOfTasks-1),nvars);
bineq_every_resource_once = logical.empty;


count = 0;


% loop through each task and mark them as "happening"
for happeningTaskId = 1:numberOfTasks
    for resourceId = 1:numberOfResources

        % get how long the "happening" task takes
        durationForHappeningTask = processingTimes{happeningTaskId,resourceId};

        % "happening" tasks might start at any time
        for time = 1:maxPossibleTime

            % indice for the current "happening" task & resource & time (if one)
            indForHappeningTask = ind(ix.task == happeningTaskId & ix.resource ==…
resourceId & ix.time == time);

            % amount of blocked time units
            timeDelta = (durationForHappeningTask-1);


            for otherTaskId = 1:numberOfTasks
                if otherTaskId == happeningTaskId
                    continue
                end
```

```
                % index for the "other task" on the same resource & time that has to be
blocked
                blockedIndForOtherTask = ind(ix.task == otherTaskId & ix.resource ==…
resourceId & ix.time <= (time+timeDelta) & ix.time >= time);


                % if constraint is found add to equality matrix
                if ~isempty(indForHappeningTask) && ~isempty(blockedIndForOtherTask)
                    count = count + 1;
                    Aineq_every_resource_once(count,…
[indForHappeningTask; blockedIndForOtherTask]) = true;
                    bineq_every_resource_once(count,1) = true;
                end
            end
        end
    end
end


% deletes the final unneccesary rows
Aineq_every_resource_once((numel(bineq_every_resource_once)+1):end,:) = [];
```

Constraint 3: Predecessors

```
% reset matrix
Aineq_preds = zeroRow;
bineq_preds = [];


% loop through each task
for taskId = 1:numberOfTasks

    % get the predecessor task numbers
    predecessorsForThisTask = find(preds{taskId,:});


    % loop through the predecessor tasks
    for predCounter = 1:numel(predecessorsForThisTask)

        predId = predecessorsForThisTask(predCounter);


        % loop through the resources, as the predecessor tasks take varying time with
respect to the used resource
        for resourceId = 1:numberOfResources

            % get the duration on current resource
            durationForPred = processingTimes{predId,resourceId};

            for time = 1:maxPossibleTime

                % get the decision variable for the current task & time (for all resources)
                indTask = ind(ix.task == taskId & ix.time == time);
```

```
                % decision variables that would indicate the current predecessor is still
running or it will run later
                indPredStillRunningOrLater = ind(ix.task == predId & ix.resource ==…
 resourceId & ix.time > (time-durationForPred));


                % if constraint is found, add to inequality matrix
                if ~isempty(indTask) && ~isempty(indPredStillRunningOrLater)
                    Aineq_preds(end+1,[indTask; indPredStillRunningOrLater]) = 1;
                    bineq_preds(end+1,1) = 1;


                end
            end
        end
    end
end
```

Constraint 4: Binary Condition

```
% all the decision variables are integers
intcon = 1:nvars;


% lower bounds and upper bounds (modified later on)
lb = zeros(nvars,1);
ub = ones(nvars,1);
```

Combine Constraints

```
Aeq = [Aeq_every_task_once];
beq = [beq_every_task_once];


Aineq = [Aineq_every_resource_once; Aineq_preds];
bineq = [bineq_every_resource_once; bineq_preds];
```

Objective Vector

```
% reset
costsTime = zeros(nvars,1);
costsCost = zeros(nvars,1);
costsMakespan = zeros(nvars,1);


for taskId = 1:numberOfTasks
    for resourceId = 1:numberOfResources
        for time = 1:maxPossibleTime

            % minimize starting time
            costsTime(ix.task == taskId & ix.resource == resourceId & ix.time == time,1) =…
 time;

            % minimize cost
            costsCost(ix.task == taskId & ix.resource == resourceId & ix.time == time,1) =…
```

```
 processingTimes{taskId,resourceId}*costsRaw.Costs(resourceId);

            % minimize makespan
            if taskId == numberOfTasks
                costsMakespan(ix.task == taskId & ix.resource == …
resourceId & ix.time==time) = time +  processingTimes{taskId,resourceId}*…
costsRaw.Costs(resourceId) -1;
            end
        end
    end
end


% summation of all components of the objective function, normalized and weighted
costs = minmax_start_times(costsTime)*weightsObjectives(1,2) +…
minmax_work(costsCost)*weightsObjectives(1,3) + ….
minmax_makespan(costsMakespan)*weightsObjectives(1,1);
```

Solver

```
% intlinprog solver without custom parameters
tic
[resVec,fval,exitflag,output] = intlinprog(costs,intcon,Aineq,bineq,Aeq,beq,lb,ub);
intlinprogtoc = toc
```

Results

```
selected = result_vector_to_table(ix, resVec, processingTimes, costs);
objective_value_intlinprog=sum(selected.cost)
selected


 % visualization
 visualize_schedule(processingTimes, preds, resourceNames, ones(1,numberOfResources),…
selected)
 title("Solution Score intlinprog: "  + objective_value_intlinprog)
```

8.4 Code MATLAB lineare Optimierung mit for-Schleifen und genetischem Algorithmus

Geänderter Code-Abschnitt zur Generierung aller Ressourcen-Kombinationen mit for-Schleifen

```
tic
% loop through all the possible resource selections
for selectionId = height(resourceSelections):-1:1


    % get selection for the current iteration
    selection = resourceSelections(selectionId,:);

    % solve it
    [fval, schedule, resVec, ixWithDetails, maxPossibleTime, ix, costs] = ...
```

```
        linear_optimization_data3(selection, processingTimesOriginal, costsRawOriginal,
preds, weightsObjectives);


    processingTimesForPlot = processingTimesOriginal(:, selection);


    selected = result_vector_to_table(ix, resVec, processingTimesForPlot, costs);
    fig = gcf;
    fig.Visible = "on";
    visualize_schedule(processingTimesForPlot, preds, resourceNames(selection),
ones(1,width(processingTimesForPlot)), selected);

    saveas(fig,fullfile("graphs02",selectionId+".fig"));
    saveas(fig,fullfile("graphs02",selectionId+".png"));


    % save the objective score (cost)
    objective(selectionId,1) = fval;


    % save resources as text
    resources(selectionId,1) = string(strjoin((resourceNames(selection)),", "));
end
% get the runtime duration
loopsToc = toc
```

Geänderter Code-Abschnitt zur Generierung möglicher Ressourcen-Kombinationen mit dem genetischen Algorithmus

```
nvars = numberOfResources;


% binary constraints
lb = zeros(1,nvars);
ub = ones(1,nvars);
intcon = 1:nvars;


% make sure at least one resource is selected
Aineq = -ones(1,nvars);
bineq = -1;


memOpt.Enabled = true;


% modifying the population size
gaoptionsLoop = optimoptions("ga");
gaoptionsLoop.PopulationSize = numberOfResources;


tic
```

```
[solGaLoop,fvalGaLoop] =
ga(objForGaLoop,nvars,Aineq,bineq,[],[],lb,ub,[],intcon,gaoptionsLoop)
gaInsteadOfLoopsToc = toc
```

8.5 Code MATLAB doppelter genetischer Algorithmus

```
clear, rng default


% data import
dataFile = "Data_3.xlsx";


% import code is in a separate function
[preds, processingTimes, resourceNames, costsRaw, weightsObjectives, numHuman, numRobot,
startupCost] = ...
    import_data_from_excel_data3(dataFile);


% update min max
minmax_updater(dataFile);


numberOfTasks = height(preds);
numberOfResources = width(processingTimes);


processingTimes
% numTotal is equal to numberOfResources
numTotal = numHuman + numRobot;
```

Decision Variables

```
% will decide the resource for each task
nvars = numberOfTasks;
tasks = 1:numberOfTasks;
```

Constraints

```
% all decision variables are integers, lower bounds are 1 and upper bounds are numTotal

intcon_1 = 1:nvars;
lb_1 = ones(1, nvars);
ub_1 = ones(1, nvars)*numTotal;
```

Objective function for main genetic Algorithm

```
% inside there is a more complex problem from scratch, that will solve the remaining
constraints and will actually optimize and deliver the objective value

ga_1_wrap = @(x) ga_second_data3(x, processingTimes, numberOfTasks,… numberOfResources,
tasks, preds, costsRaw, startupCost, weightsObjectives);
```

Results

```
% solve ga optimization
tic
options = optimoptions("ga", ...
    "MaxStallGenerations",5, ...
    "UseParallel",true, ...
    "MaxTime",900, ... %in seconds
    "Display","iter", ...
    "PopulationSize",50, ...
    "MaxGenerations",40);

[resource,objectiveFromMainGa] =
ga(ga_1_wrap,nvars,[],[],[],[],lb_1,ub_1,[],intcon_1,options);
GAtoc=toc;


% get the timings
[objectiveFromSecondGa, time]= ga_1_wrap(resource);


% preview results
selected = table((1:numberOfTasks)', time', resource','VariableNames',…
["task", "time","resource"]);
selected = add_processing_times_to_table(selected, processingTimes);
selected.name = processingTimes.Properties.RowNames


figure
visualize_schedule(processingTimes, preds, resourceNames,true(1,numberOfResources),
selected)
title("Solution Score double GA: " + GAtoc  + " || "+ objectiveFromSecondGa)
```

8.6 Weitere erzeugte Funktionen, auf denen zurückgegriffen wird

Funktion zum Einlesen der Eingaben aus Excel

```
function [preds, processingTimes, resourceNames, costsRaw, weightsObjectives, numHuman,…
numRobot, startupCost] = import_data_from_excel_data3(dataFile)


preds = …
readtable(dataFile,"ReadRowNames",true,"Sheet","predecessor",'ReadVariableNames',false);
processingTimesRaw = readtable(dataFile,"ReadRowNames",true,"Sheet","processing times");


processingTimesRaw(string(processingTimesRaw.Var1) == "Σ",:) = [];
processingTimesRaw.Var1 = [];


numHuman = readmatrix(dataFile,"Sheet",1,"Range","C3:C3");
numRobot  = readmatrix(dataFile,"Sheet",1,"Range","C4:C4");
costHuman = readmatrix(dataFile,"Sheet",1,"Range","C5:C5");
costRobot = readmatrix(dataFile,"Sheet",1,"Range","C6:C6");
startupCost = readmatrix(dataFile,"Sheet",1,"Range","C7:C7");
```

```
weightObjective1 = readmatrix(dataFile,"Sheet",1,"Range","C9:C9");
weightObjective2 = readmatrix(dataFile,"Sheet",1,"Range","C10:C10");
weightObjective3 = readmatrix(dataFile,"Sheet",1,"Range","C11:C11");
weightObjective4 = readmatrix(dataFile,"Sheet",1,"Range","C12:C12");


weightsObjectives = [weightObjective1 weightObjective2 weightObjective3 weightObjective4]


Costs = repelem([costHuman costRobot],[numHuman numRobot])';
resourceNames = repelem(["Human " "Robot "],[numHuman numRobot]);
resourceIds = [1:numHuman 1:numRobot];
resourceNames = [resourceNames + resourceIds]';


costsRaw = table(Costs);
costsRaw.Properties.RowNames = resourceNames;


for humanId = 1:numHuman
    col = humanId;
    newProcessingTimes(:,col) = processingTimesRaw.Humans;
end


for robotId = 1:numRobot
    col = numHuman + robotId;
    newProcessingTimes(:,col) = processingTimesRaw.Robots;
end


processingTimes = array2table(newProcessingTimes);
processingTimes.Properties.RowNames = preds.Properties.RowNames;
processingTimes.Properties.VariableNames = resourceNames
end
```

Funktion zur Ermittlung der Minima und Maxima

```
function [] = minmax_updater(dataFile)
% function updates the min and max values used by minmax_makespan, minmax_start_times and
minmax_work functions


% if no file name is defined, use Data_3.xlsx as the filename:
if nargin == 0
    dataFile = "Data_3.xlsx";
end


% get the processing times matrix and costs vector
[preds, processingTimes, ~, costsRaw, ~, numHuman, numRobot, startupCost] = ...
    import_data_from_excel_data3(dataFile);
```

```
numberOfTasks = height(processingTimes);
numberOfResources = width(processingTimes);


% get maximum possible time
[~, ~, ~, ~, ~, ~, ~, ~, maxPossibleTime] = ...
    build_problem_data3(numberOfTasks, numberOfResources, processingTimes, preds);


% calculating min and max


% min makespan is 1
min_makespan = 1
minmax.obj_makespan.range(1) = min_makespan / numberOfTasks;


% max makespan is maxPossibleTime
max_makespan = maxPossibleTime
minmax.obj_makespan.range(2) = max_makespan / numberOfTasks;


% min start times summation is numberOfTasks
minmax.obj_start_times.range(1) = numberOfTasks / numberOfTasks;


% max start times summation is numberOfTasks*maxPossibleTime
minmax.obj_start_times.range(2) = numberOfTasks*maxPossibleTime / numberOfTasks;


% get the minimum and maximum processing time for each task and sum them up to get minimum
and maximum working hours
minWorkHours = sum(min(processingTimes.Variables,[],2))
maxWorkHours = sum(max(processingTimes.Variables,[],2))


% calculate the cheapest and most expensive work costs by multiplying cheapest and most
expensive resource cost with the smallest and biggest possible work hours
cheapestCost = min(costsRaw.Costs)*minWorkHours
expensiveCost = max(costsRaw.Costs)*maxWorkHours


minmax.obj_work.range(1) = cheapestCost / numberOfTasks;
minmax.obj_work.range(2) = expensiveCost / numberOfTasks;


% delete the minmax values in the memory
clear minmax_makespan minmax_start_times minmax_work


% save the new ones
save minmax minmax
end
```

Berechnung der Werte für Minima und Maxima für die Zielfunktion

```
function [output] = minmax_start_times(input)
persistent min max
if isempty(min)
    load minmax minmax
    min = minmax.obj_start_times.range(1);
    max = minmax.obj_start_times.range(2);
end

output = (input - min)/(max-min);
output(input == 0) = 0;

end


function [output] = minmax_work(input)
persistent min max
if isempty(min)
    load minmax minmax
    min = minmax.obj_work.range(1);
    max = minmax.obj_work.range(2);
end

output = (input - min)/(max-min);
output(input == 0) = 0;

end


function [output] = minmax_makespan(input)
persistent min max
if isempty(min)
    load minmax minmax
    min = minmax.obj_makespan.range(1);
    max = minmax.obj_makespan.range(2);
end

output = (input - min)/(max-min);
output(input == 0) = 0;


end
```

Tabellarische Darstellung der Ergebnisse

```
function selected = result_vector_to_table(ix, resVec, processingTimes, costs)


ixWithDetails = ix;


if exist("costs","var")
 ixWithDetails.cost = costs;
end


% show selected decision variables as a schedule
 selected = ixWithDetails(resVec > 0.5,:);


% add processing times
selected = add_processing_times_to_table(selected, processingTimes);
```

```
% add names
selected.name = processingTimes.Properties.RowNames;
end
```

Graphische Darstellung der Ergebnisse

```
function visualize_schedule(processingTimes, preds, resourceNames, selection, schedule)


    numberOfResources = sum(selection);
    numberOfTasks = height(processingTimes);

    for taskId = 1:(numberOfTasks-1)
        % loop through each task


        % get its chosen resource and time, and get the duration on that resource
        taskResource = schedule.resource(schedule.task == taskId);
        taskDuration = processingTimes{taskId,taskResource};
        taskTime = schedule.time(schedule.task == taskId);


        % create the time indices for plotting
        taskTimeSeries = taskTime:1:(taskTime+taskDuration);


        % repeat the resource array to plot it
        taskResourceSeries = repmat(taskResource,[1 numel(taskTimeSeries)]);


        p = plot(taskTimeSeries, taskResourceSeries,'LineWidth',5,'LineStyle','-
','Marker','o');
        hold on


    end


    hold off
    xlabel("Time")
    ylabel("Resource")
    grid on
    title("Solution")


    taskNames = "Task" + (1:numberOfTasks);
    legend(taskNames(1:(end-
1)),"Location","eastoutside","Orientation","horizontal","NumColumns",2)


    ylim([0 numberOfResources+1])

    ax = gca;
    ax.YTick = 1:numberOfResources;
```

```
    ax.YTickLabel = resourceNames(logical(selection));
end
```

Verschachtelter genetischer Algorithmus, der die Vorgängerbeziehungen berücksichtigt (wobei die Ressourcenzuweisung bereits aus dem äußeren genetischen Algorithmus feststeht) und den genetischen Algorithmus-Solver aufruft

```
function [objectiveValue2, solution_2] = ga_second_data3(solution_1, … processingTimes,
numberOfTasks, numberOfResources, tasks, preds, costsRaw, … startupCost, weightsObjectives)

% lower bounds and upper bounds for the second ga
nvars = numberOfTasks;
maxPossibleTime = max(sum(processingTimes.Variables,1)) + 1;
lb_2 = ones(1,nvars);
ub_2 = ones(1,nvars)*maxPossibleTime;


intcon_2 = 1:nvars;


% build the predecessor constraints as linear inequality constraints
Aineq_2 = [];
bineq_2 = [];

% for each resource
for resourceId = 1:numberOfResources

    % get the list of tasks happening on that resource
    tasksOnThisResource = tasks(round(solution_1) == resourceId);


    % for each task happening on that resource
    for taskOrder = 1:numel(tasksOnThisResource)

        taskId = tasksOnThisResource(taskOrder);


        % find its predecessors
        predecessorsForThisTask = find(preds{taskId,:});


        % for each predecessor
        for predCounter = 1:numel(predecessorsForThisTask)

            predId = predecessorsForThisTask(predCounter);


            % find the resource of the predecessor
            selectedResourceForPred = round(solution_1(predId));


            % get the corresponding task duration
```

```
            exactDurationForPred = (processingTimes{predId,selectedResourceForPred});


            % add a new row to the inequality matrix and vector
            Aineq_2(end+1, [taskId predId]) = [-1 1];

            % the difference between the task and the predecessor must be at least
predecessors duration
            bineq_2(end+1) = -exactDurationForPred;
        end
    end
end


% make sure inequality matrix has the correct number of columns and rows
if height(Aineq_2) == 0
    Aineq_2(1,:) = 0;
    bineq_2(1) = 0;
end
if width(Aineq_2)<nvars
    Aineq_2(end,nvars) = 0;
End


% nonlinear constraints for the second ga [resource is used once per time unit (max)] are
in the noncon_second file
noncon2Wrap = @(dec) noncon_second(dec, solution_1, numberOfResources, … processingTimes);


% codes for speed
noncon2Wrap_mem = memoize(noncon2Wrap);
noncon2Wrap_mem.clearCache();
noncon2Wrap_mem.CacheSize = 2^nvars;
noncon2Wrap_mem_final = @(x) noncon2Wrap_mem(x);


% objective function in the file objective_ga_second_data3 is the objective of this one and
therefore the main GA
objWrap = @(x) objective_ga_second_data3(x, solution_1, numberOfResources, processingTimes,
costsRaw, startupCost, weightsObjectives);


% no text output is needed from the second GA as it will be run a lot of times by the main
GA
options = optimoptions("ga","Display","off","PlotFcn",[],"PopulationSize",15);


% termination criteria for the second GA
options.MaxStallGenerations = 10;


% the objective score from the second ga (objectiveValue2) is directly the output of this
function
```

```
[solution_2,objectiveValue2] =
ga(objWrap,nvars,Aineq_2,bineq_2,[],[],lb_2,ub_2,noncon2Wrap,intcon_2,options);


end
```

Nichtlineare Nebenbedingung, welche gewährleistet, dass jede Ressource maximal ein Mal je Zeiteinheit eingesetzt wird

```
function [cineq, ceq] = noncon_second(decisionVariables, solution_1, … numberOfResources,
processingTimes)
% decision variables from the first GA = resource assignments
solution_1 = round(solution_1);

% decision variables from the second GA = starting times
decisionVariables = round(decisionVariables);

cineq = [];
% for each resource, find the tasks on that resource
for resourceId = 1:numberOfResources
    tasksOnThisResource = find(solution_1 == resourceId);

    % create a variable called busy ranges. At the end of this operation,
    this variable will hold values that show when exactly the resource is in use
    busyRanges = [];

    % this variable will hold the total processing time on that resource

    % it is used to make sure tasks are not intersecting on the same resource
    processingTimeTotal = 0;
    for taskOrder = 1:numel(tasksOnThisResource)

        % for each task on this resource
        taskId = tasksOnThisResource(taskOrder);

        % get when it starts
        startTime = decisionVariables(taskId);

        % get its duration
        pt = processingTimes{taskId, resourceId};

        % calculate the end time
        stopTime = startTime + pt - 1;

        % add this duration to the total duration on that resource
        processingTimeTotal = processingTimeTotal + pt;

        % create an array from start time to stop time, including middle times
        busyRange = startTime:stopTime;

        % add that array to the "busyRanges" variable
        busyRanges = [busyRanges busyRange];
    end
%if there is no intersection on a resource, the number of unique values in the busyRanges
variable should be equal to total processing time

% if there are less unique values, that means there are repeating times and therefore one
resource is assigned to multiple tasks at the same time
    cineq(end+1) = processingTimeTotal - numel(unique(busyRanges));

end

% there are no nonlinear equality constraints
ceq = [];
end
```

Zielfunktion des Verschachtelten genetischen Algorithmus

```
function obj_total = objective_ga_second_data3(decisionVariables, solution_1,
numberOfResources,processingTimes, costsRaw, startupCost, weightsObjectives)
% this function calculates the objective

solution_1 = round(solution_1);
decisionVariables = round(decisionVariables);

% objective of minimizing the start times
obj_start_times = sum(decisionVariables);

% objective to detect and add the standby as work time
obj_standby = 0;
obj_work = 0;
numberOfTasks = numel(solution_1);

for resourceId = 1:numberOfResources

    % for each resouce, get tasks happening on that resource
    tasksOnThisResource = find(solution_1 == resourceId);
    tasksOnThisResource(tasksOnThisResource==numberOfTasks)=[];

    % variable will hold the busy times for schedule
    busyRanges = [];

    for taskOrder = 1:numel(tasksOnThisResource)
        taskId = tasksOnThisResource(taskOrder);
        startTime = decisionVariables(taskId);
        pt = processingTimes{taskId, resourceId};
        stopTime = startTime + pt - 1;
        busyRange = startTime:stopTime;
        busyRanges = [busyRanges busyRange];
    end

    % sort the busyRanges array from min to max
    busyRanges = sort(unique(busyRanges));


    % allRange starts with first task and ends when the last task on that resource is
    finished
    allRange = min(busyRanges):max(busyRanges);

% the numbers that exist in allRange but not in busyRange are standby times
    standbyRange = setdiff(allRange, busyRanges);

    % count the number of elements in the standbyRange variable for this resource and add
    it to total work time
    obj_standby = (obj_standby + numel(standbyRange))*costsRaw.Costs(resourceId);
    obj_work = (obj_work + numel(busyRanges))*costsRaw.Costs(resourceId);
end



% objective to minimize makespan and startup cost
obj_makespan = decisionVariables(end) + processingTimes{end,solution_1(end)} - 1 +
numberOfResources*startupCost;

% final objective with weighted normalized (minmax) summation
obj_total = minmax_start_times(obj_start_times)*weightsObjectives(1,2) ...
    + minmax_makespan(obj_makespan)*weightsObjectives(1,1)...
    + minmax_work(obj_standby+obj_work)*weightsObjectives(1,3);


end
```

9 Literaturverzeichnis

Behrens, R., 2018. *Biomechanische Grenzwerte für die sichere Mensch-Roboter-Kollaboration.* Wiesbaden: Springer Vieweg.

Blazewicz, J., Ecker, K., Schmidt, G., Sterna, M., Weglarz, J., 2019. *Handbook On Scheduling.* s.l.:Springer.

Bogner, K., Pferschy, U., Unterberger, R. & Zeiner, H., 2018. Optimised scheduling in human-robot collaboration – a use case in the assembly of printed circuit boards. *International Journal of Production Research,* Issue August.

Brereton, P., Kitchenham, B., Budgen, D., Turner, M., Khalil, M., 2006. Lessons from applying the systematic literature review process within the software engineering domain. *The Journal of Systems and Software,* Issue 80, pp. 571-583.

Brucker, P. & Knust, S., 2012. *Complex Scheduling.* 2. Hrsg. Berlin Heidelberg: Springer.

Corsten, H., Corsten, H. & Sartor, C., 2015. *Operations Research: eine problemorientierte Einfünhrung.* s.l.:Verlag Franz Vahlen.

Dangelmaier, W., 2021. *Produktionstheorie 4.* Berlin: Springer Vieweg.

Faber, M., Bützler, J. & Schlick, C., 2015. Human-robot cooperation in future production systems: Analysis of requirements for designing an ergonomic work system. *Procedia Manufacturing,* Issue 3, pp. 510-517.

Garey, M. R., Johnson, D. S. & Johnson, D. E., 1979. *Computers & Intractability: A Guide to the Theory of Np-Completeness (Series of Books in the Mathematical Sciences).* San Francisco: WH Freeman and Co.

Gerdts, M. & Lempio, F., 2011. *Mathematische Optimierungsverfahren des Operations Research.* s.l.:De Gruyter.

Glück, M., 2022. *Mensch-Roboter-Kooperation erfolgreich einführen - Grundlagen, Leitfaden, Applikationen.* Wiesbaden: Springer Vieweg.

Göllmann, L., Hübl, R., Pulham, S., Ritter, S., Schon, H., Schüffler, K., Voß, U., Vossen, G., 2017. *Mathematik für Ingenieure: Verstehen – Rechnen – Anwenden (Band 2).* s.l.:Springer Berlin Heidelberg.

Grimme, C. & Bossek, J., 2018. *Einführung in die Optimierung.* Wiesbaden: Springer Vieweg.

Hammer, P., Johnson, E. & Korte, B., 1979. *Discrete optimization Volume 11.* s.l.:North-Holland Pub. Co..

Jarboui, B., Siarry, P. & Teghem, J., 2013. *Metaheuristics for Production Scheduling.* London: ISTE Ltd.

Kalvelagen, E., 2016. *Yet Another Math Programming Consultant.* [Online] Available at: http://yetanothermathprogrammingconsultant.blogspot.com/2016/02/xor-as-linear-inequalities.html
[Zugriff am 15 August 2022].

Kandiller, L., 2007. *Principles of Mathematics in Operations Research.* New York: Springer.

Kathöfer, U. & Müller-Funk, U., 2017. *Operations Research.* 3. Hrsg. Konstanz: UVK Verlagsgesellschaft.

Kitchenham, B., 2004. *Procedures for Performing Systematic Reviews,* Australia: Keele University Technical Report.

Klaerner, J., 2021. *igus blogs.* [Online] Available at: https://blog.igus.de/was-kostet-ein-roboter-wirklich/
[Zugriff am 05 November 2022].

Koop, A. & Moock, H., 2018. *Lineare Optimierung – eine anwendungsorientierte Einführung in Operations Research.* s.l.:Springer Berlin Heidelberg.

Lee, W. J., Kwag, S. I. & Ko, Y. D., 2019. Optimal capacity and operation design of a robot logistics system for the hotel industry. *Tourism Management,* Issue Februar.

Lopez, P. & Roubellat, F., 2008. *Production scheduling.* s.l.:Wiley.

März, L., Krug, W., Rose, O. & Weigert, G., 2010. *Simulation und Optimierung in Produktion und Logistik.* s.l.:Springer.

MATLAB, 2022. *MathWorks.* [Online] Available at: https://de.mathworks.com/help/gads/ga.html?s_tid=doc_ta
[Zugriff am 15 OKtober 2022].

Monguzzi, A., Badawi, M., Zanchettin, A. M. & Rocco, P., 2022. A mixed capability-based and optimization methodology for human-robot task allocation and scheduling. *IEEE International Conference on Robot and Human Interactive Communication,* Issue 31.

Müller, R., Franke, J., Heinrich, D., Kuhlenkötter, B., Raatz, A., Verl, A., 2019. *Handbuch Mensch-Roboter-Kollaboration.* München: Carl Hanser Verlag.

Nahrstedt, H., 2018. *Algorithmen für Ingenieure.* 3. Hrsg. Wiesbaden: Springer Vieweg.

Nejad, M. G., Güden, H., Vizvári, B. & Barenji, R. V., 2018. Trade-off between process scheduling and production cost in cyclic flexible robotic cells. *The International Journal of Advanced Manufacturing Technology,* Issue February.

Nejad, M. G., Shavarani, S. M., Güden, H. & Barenji, R. V., 2019. Process sequencing for a pick-and-place robot in a real-life flexible robotic cell. *The International Journal of Advanced Manufacturing Technology,* Issue May.

Pinedo, M., 2016. *Scheduling.* 4, Hrsg. s.l.:Springer.

Ren, W., Yang, X., Yan, Y. & Hu, Y., 2022. The decision-making framework for assembly tasks planning in human-robot collaborated manufacturing system. *International Journal of Computer Integrated Manufacturing.*

Singla, V., 2021. *Operations Research Using Excel: A Case Study Approach.* s.l.:CRC Press.

Tsarouchi, P., Michalos, G., Makris, S., Thanasis, A., Dimoulas, K., Chryssolouris, G., 2017. On a human-robot worplace design and task allocation system. *International Journal of Computer Integrated Manufacturing,* Issue 30.

von Rimscha, M., 2008. *Algorithmen kompakt und verständlich.* 1. Hrsg. Wiesbaden: Vieweg+Teubner.

Wirtschaftskammer Österreich, 2022. *wko.at.* [Online] Available at: https://www.wko.at/service/kollektivvertrag/kv-eisen-metallverarbeitende-gewerbe-arbeiter-2022.html#heading_mindestgrundloehne
[Zugriff am 01 Dezember 2022].

Yao, F., Song, Y.-J., Zhang, Z.-S., Xing, L.-N., Ma, X., Li, X.-J., 2019. Multi-mobile robots and multi-trips feeding scheduling problem in smart manufacturing system: An improved hybrid genetic algorithm. *International Journal of Advanced Robotic Systems,* Issue August.

Zhang, F., Nguyen, S. M. Y. & Zhang, M., 2021. *Genetic programming for production scheduling.* s.l.:Springer.

Zimmermann, H.-J., 2008. *Operations Research.* 2. Hrsg. Wiesbaden: vieweg.

10 Abbildungsverzeichnis

11 Tabellenverzeichnis