FH Bielefeld
University of
Applied Sciences

AMMO
Angewandte Mathematische Modellierung & Optimierung

AMMO – Berichte aus Forschung und Technologietransfer

DISKRETE MODELLIERUNG UND OPTIMIERUNG PRAXISRELEVANTER PROZESSE MIT PETRI-NETZEN

Dr. rer. nat. Sabrina Proß

Heft Nr. 4

September 2014

Veröffentlichungsreihe (Onlinepublikation):
AMMO – Berichte aus Forschung und Technologietransfer

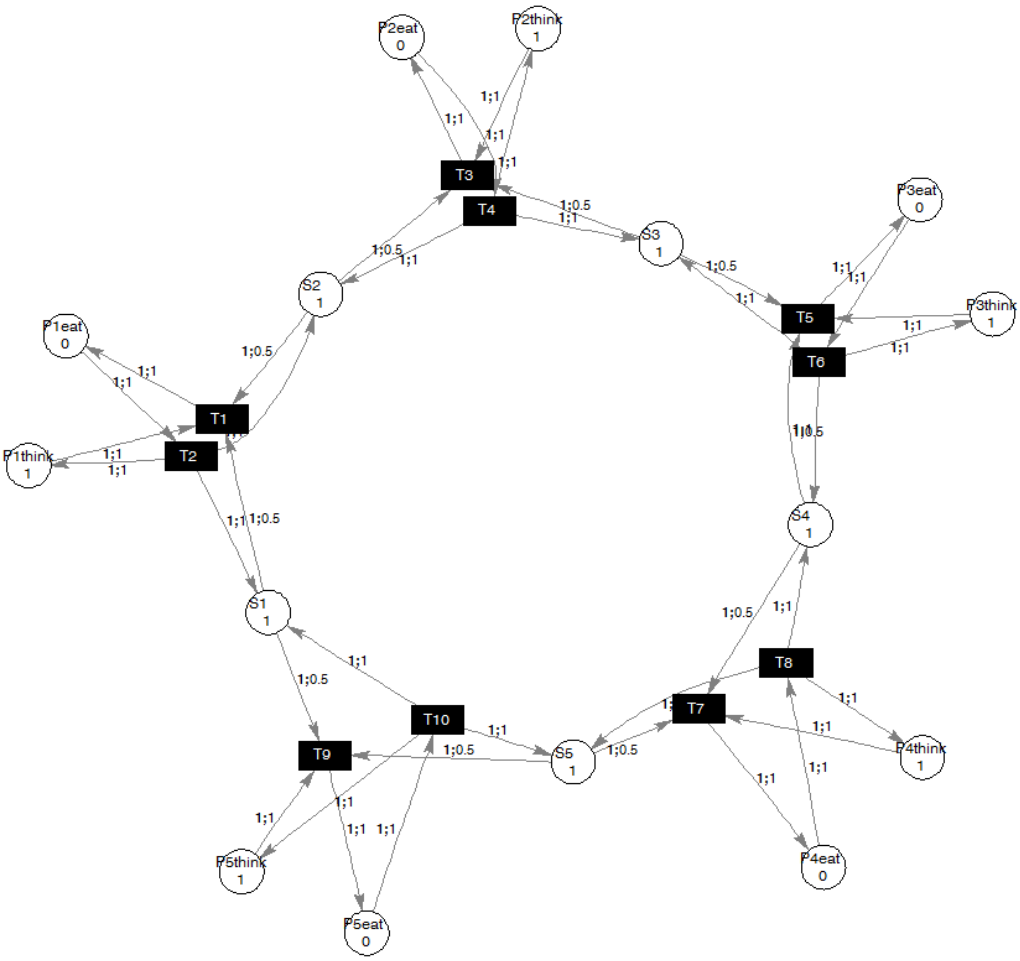
ISSN
2198-4824

Erscheinungsort
http://www.fh-bielefeld.de/ammo/veroeffentlichungen/ammo-berichte-aus-forschung-und-technologietransfer

Herausgeber
Sprecher FSP AMMO, Fachhochschule Bielefeld

Fachhochschule Bielefeld
Fachbereich Ingenieurwissenschaften und Mathematik
FSP Angewandte Mathematische Modellierung und Optimierung
Am Stadtholz 24
33609 Bielefeld

DISKRETE MODELLIERUNG UND OPTIMIERUNG PRAXISRELEVANTER PROZESSE MIT PETRI-NETZEN



Masterstudiengang Optimierung und Simulation

Masterarbeit zur Erlangung des akademischen Grades Master of Science (M.sc.)

Eingereicht an der Fachhochschule Bielefeld von

Dr. Sabrina Proß

Betreuer

Prof. Dr. Bernhard Bachmann
Prof. Dr. Hermann-Josef Kruse

Dr. Sabrina Proß

Diskrete Modellierung und Optimierung
praxisrelevanter Prozesse mit Petri-Netzen

Masterarbeit
Masterstudiengang Optimierung und Simulation

Fachbereich Ingenieurwissenschaften und Mathematik der Fachhochschule Bielefeld
Angewandte Mathematik, Prof. Dr. Bernhard Bachmann
Angewandte Mathematik, Prof. Dr. Hermann-Josef Kruse

Prüfer:
Prof. Dr. Bernhard Bachmann, Fachhochschule Bielefeld
Prof. Dr. Hermann-Josef Kruse, Fachhochschule Bielefeld

Druck: Bielefeld, Juli 2014
Gedruckt auf alterungsbeständigem Papier (ISO 9706)

Danksagung

Leider lässt sich eine wahrhafte Dankbarkeit mit Worten nicht ausdrücken.

Johann Wolfgang von Goethe

Ich möchte es dennoch an dieser Stelle versuchen und mich herzlich bei allen bedanken, die mich während der Erstellung dieser Arbeit unterstützt haben.

Zuerst gilt mein ganz besonderer Dank meinen Betreuern Herrn Prof. Dr. Kruse und Herrn Prof. Dr. Bachmann, die mich nicht nur während dieser Arbeit, sondern schon über Jahre hinweg unterstützt und beraten haben.

Für die anregenden Diskussionen und hilfreichen Ratschläge möchte ich mich bei den Mitgliedern der Arbeitsgruppe *gefärbte Petri-Netze* bedanken, zu der neben Herrn Prof. Dr. Kruse und Herrn Prof. Dr. Bachmann auch Herr Ochel und Herr Kleine-Döpke gehören.

Zudem gilt mein Dank meinen Kollegen aus Bielefeld und Gütersloh für die moralische Unterstützung und die motivierenden Gespräche.

Zutiefst zu Dank verpflichtet bin ich meinem Freund Michael und meinen Eltern Bärbel und Horst, die mich nicht nur während meiner Masterarbeit, sondern auch während meines gesamten Studiums und meiner Doktorarbeit immer tatkräftig unterstützt und ermutigt haben.

Inhaltsverzeichnis

1 EINLEITUNG

Aufgrund großer Speicherkapazitäten und moderner Computertechniken, werden unaufhörlich enorme Datenmengen in allen erdenklichen Bereichen des Lebens produziert. Diese **Daten** sind unverzichtbar, aber sie führen nicht direkt zu neuem Wissen über einen betrachteten Prozess oder ein betrachtetes System z.B. aus dem Bereich der Biologie, der Physik, der Logistik oder den Sozialwissenschaften. Es stellt sich die Frage: Wie gelangt man von „rohen“ Daten zu brauchbaren neuen Erkenntnissen über Systeme oder Prozesse?

Zur Beschreibung jeglicher Art von Prozessen oder Systemen, können **mathematische Modelle** verwendet werden. Dadurch ist eine Erforschung des Gegenstandsbereichs mithilfe von mathematischen Methoden möglich. Zahlreiche Formalismen zur Modellierung von Prozessen in unterschiedlichsten Anwendungsgebieten wurden bereits veröffentlicht. Eine Zusammenfassung bekannter Modellierungsansätze kann z.B. in (Haußer und Luchko 2011), (Imboden und Koch 2008) und (Ortlieb et al. 2009) gefunden werden.

Viele Modellformalismen haben aufgrund ihrer Struktur nur einen beschränkten Anwendungsbereich. Beispielweise können Differentialgleichungssysteme nur zur Abbildung von kontinuierlichen Prozessen herangezogen werden und zelluläre Automaten dienen ausschließlich zur Modellierung von diskreten Abläufen. Im Gegensatz dazu wurde in (Proß 2013) gezeigt, dass Petri-Netze mit ihren zahlreichen Erweiterungsmöglichkeiten das ideale Modellierungskonzept sind, um Systeme in nahezu jedem Abstraktionsgrad abbilden zu können. Sie ermöglichen eine qualitative sowie quantitative Modellierung. Ein **qualitatives Modell** stellt alle wesentlichen Elemente eines Systems und deren Beziehungen untereinander dar. Ein **quantitatives Modell** beschreibt zusätzlich die zeitlichen Veränderungen der Elemente des Systems. Daher ist ein qualitatives Modell die Basis für jedes quantitative Modell und aus einen qualitativen Modell kann durch sukzessive Erweiterung ein quantitatives entstehen. Darüber hinaus ermöglichen Petri-Netze sowohl die Abbildung kontinuierlicher als auch diskreter Prozesse. Zudem können beide Welten in einem Petri-Netz zusammengefasst werden, das **hybrides Petri-Netz** genannt wird (siehe z.B. David und Alla 2001).

In (Proß 2013) ist der erweiterte hybride Petri-Netz-Formalismus vorgestellt worden, der mit **xHPN** abgekürzt wird (e**x**tended **H**ybrid **P**etri **N**et). Dieser Formalismus ist zunächst entwickelt worden, um biologische Prozesse, wie beispielsweise Stoffwechselprozesse, Genregulation und Signaltransduktion, abbilden zu können. Im Laufe der Zeit hat sich jedoch herausgestellt, dass der *xHPN*-Formalismus so generisch und universell einsetzbar ist, dass er zur Modellierung von fast allen möglichen Prozessarten aus den unterschiedlichsten Anwendungsbereichen herangezogen werden kann.

Der *xHPN*-Formalismus ist mithilfe der objekt-orientierten Modellierungssprache **Modelica** umgesetzt worden, um eine graphische Modellierung und eine hybride Simulation zu ermöglichen (siehe auch Proß und Bachmann 2012). Alle Komponenten des Petri-Netzes sind in einer Modelica-Bibliothek zusammengefasst, die **PNlib** genannt wird.

Ziel dieser Arbeit ist den bestehenden *xHPN*-Formalismus im Bereich der qualitativen Modellbildung zu erweitern. Es sollen weitere Möglichkeiten zur Lösung von Konflikten sowie neue Strategien zur Auswahl der feuernden Transitionen entwickelt werden. Zusätzlich soll eine Optimierung von Petri-Netzen etabliert werden, d.h. es soll die Feuerungsreihenfolge ermittelt werden, die in Hinblick auf ein definiertes Ziel optimal ist. Jeder Transition könnte beispielsweise ein Nutzen oder Kosten für dessen Feuerung zugeordnet werden. Dann ist die Feuerungsreihenfolge gesucht, die nach einer bestimmten Anzahl Schritte den maximalen Nutzen bzw. minimale Kosten liefert. Dieser maximale Nutzen könnte beispielsweise die Anzahl Token in einem bestimmten Platz sein. Bei der Kostenminimierung kann zudem die Bedingung, dass eine gewisse Zielmarkierung erreicht wird, gestellt werden.

Diese Erweiterungen des Formalismus sollen in einem Tool implementiert werden, um eine **Simulation** zu ermöglichen. Dieses Ziel kann nicht durch Erweiterung der *PNlib* erreicht werden, da es mit Modelica zurzeit nicht möglich ist, ohne Zeitbetrachtung zu simulieren. Zudem bietet die *PNlib* in Kombination mit dem Modelica-Tool Dymola momentan keine zufriedenstellende graphische Oberfläche, mit der intuitiv, einfach und schnell Petri-Netze erstellt werden können.

Aus diesem Grund wurde ein neues Tool mit MATLAB entwickelt, das **PNmat** genannt wird. Die Matrizenschreibweise in MATLAB erfüllt alle Voraussetzungen, um das obengenannte Ziel zu erreichen. Zudem bietet MATLAB zahlreiche Möglichkeiten benutzerfreundliche Oberflächen zu erstellen.

Diese Arbeit ist in das **Projekt gefärbte Petri-Netze** des Forschungsschwerpunktes *Angewandte Mathematische Modellierung und Optimierung* der FH Bielefeld (*FSP AMMO*) integriert. Dieses Projekt hat das Ziel Petri-Netze mit allen zugehörigen Prozessen zu definieren und Lösungen für jegliche Art von Konflikten zu erarbeiten. Die Ergebnisse sollen in einem Buch zusammengefasst werden, das dem Leser einerseits einen Einstieg in die Welt der Petri-Netze bietet und in der Lehre eingesetzt werden soll (siehe Bachmann et al. 2014). Anderseits soll diese theoretische Arbeit als Spezifikation für die Entwicklung eines Tools zur Simulation von gefärbten Petri-Netzen dienen.

Die in dieser Arbeit entwickelte *PNmat* ist ein erster Schritt auf den Weg zur Simulation von gefärbten Petri-Netzen, mit der zunächst erweiterte Petri-Netze ohne Betrachtung der Zeit simuliert und optimiert werden können.

Die *PNmat* soll vor allem zur Unterstützung der Lehre eingesetzt werden. Die Studierenden sollen im Rahmen von Projektarbeiten den Umgang mit Petri-Netzen zur Modellierung praxisrelevanter Prozesse erlernen. Aber auch Projekte mit der Industrie sind denkbar.

Diese Arbeit ist neben der Einleitung und dem abschließenden Ausblick in drei wesentliche Kapitel untergliedert. In Kapitel 2 wird der in der *PNmat* umgesetzte Petri-Netz-Formalismus schrittweise hergeleitet. Zunächst wird das grundlegende Konzept und die Möglichkeiten zur Lösung von Konflikten vorgestellt. Anschließend wird der Prozess der Feuerung von Transitionen näher betrachtet und es werden unterschiedliche Strategien zur Auswahl der feuernden Transitionen präsentiert. Vereinfachungen und Erweiterungen werden im nächsten Abschnitt vorgestellt. Die Darstellung von Petri-Netzen mithilfe von Matrizen und Vektoren bildet den Abschluss dieses Kapitels.

Kapitel 3 widmet sich der Ermittlung der optimalen Feuerungsreihenfolge. Dazu werden zunächst zwei mögliche Optimierungsprobleme vorgestellt: Maximaler Nutzen und minimale Kosten. Anschließend werden Verfahren zur Lösung dieser Optimierungsprobleme präsentiert. Da es sich bei diesen binären Optimierungsproblemen um NP-schwere Probleme handelt werden neben einem exakten Verfahren auch Näherungsverfahren vorgestellt. Als exaktes Verfahren wird die Branch-and-Bound-Methode eingesetzt und bei den Näherungsverfahren werden verschiedene Metaheuristiken verwendet.

In Kapitel 4 wird die Umsetzung des Petri-Netz-Formalismus in MATLAB erläutert. Zunächst muss dazu die Mengenschreibweise der Petri-Netze in Vektoren und Matrizen

umgewandelt werden. Anschließend werden die Implementierung der einzelnen Prozesse zur Petri-Netz-Simulation und die Entwicklung einer graphischen Oberfläche näher betrachtet. Abschließend wird die Benutzung der *PNmat* anhand eines Beispiels erläutert.

2 PETRI-NETZE

Petri-Netze sind ein graphisches Modellierungskonzept zur Darstellung und Modellierung von Verhaltensweisen mit Konkurrenz, Nebenläufigkeit, Synchronisierung, Ressourcenteilung und Nichtdeterminismus. Das grundlegende Konzept wurde 1962 von Carl Adam Petri im Rahmen seiner Doktorarbeit vorgestellt (Petri 1962) und sukzessive erweitert, um den Anwendungsbereich zu vergrößern (siehe David und Alla 2010, Proß 2013).

Petri-Netze werden bereits in den unterschiedlichsten Anwendungsbereichen eingesetzt. Beispiele hierfür sind die Modellierung von Geschäftsprozessen, Produktionsprozessen, logistischen Prozessen, biologischen und chemischen Prozessen, Verkehrsflüssen, Datenflüssen, Arbeitsabläufen und Multiprozessorsystemen.

Dieses Kapitel dient als Spezifikation für die MATLAB-Implementierung des grundlegenden Petri-Netz-Konzeptes und dessen Vereinfachungen und Erweiterungen. **Vereinfachungen** des grundlegenden Petri-Netz-Konzeptes dienen der Übersichtlichkeit und der verbesserten graphischen Darstellung eines Petri-Netz-Modells. Zudem erleichtern sie den Modellierungsprozess. Vereinfachungen können aber immer in ein grundlegendes Petri-Netz umgewandelt werden und erweitern deshalb den Anwendungsbereich der Petri-Netze nicht. Im Gegensatz dazu können **Erweiterungen** nicht mit einem grundlegenden Petri-Netz modelliert werden und ermöglichen den Einsatz des Petri-Netz-Formalismus in einem größeren Anwendungsbereich.

2.1 GRUNDLEGENDES KONZEPT

Ein Petri-Netz ist ein gerichteter, 2-gefärbter Graph. Bei einem 2-gefärbten Graphen lassen sich die Knoten in zwei disjunkte Teilmengen aufteilen. Diese heißen im Falle eines Petri-Netzes **Transitionen** und **Plätze**. Plätze und Transitionen können durch **gerichtete Kanten** verbunden werden. Verbindungen zwischen Transitionen bzw. Plätzen untereinander sind hingegen nicht erlaubt. Plätze werden graphisch durch Kreise dargestellt und Transitionen durch Rechtecke (siehe Abbildung 2.1).

Definition 2.1

Ein **Netz** ist das Tupel (P, T, F, G) mit

- einer endlichen Menge an Plätzen $P = \{p_1, p_2, \dots, p_{n_p}\}$,
- einer endlichen Menge an Transitionen $T = \{t_1, t_2, \dots, t_{n_t}\}$, wobei $P \cap T = \emptyset$,
- einer Menge $F \subseteq (P \times T)$ an gerichteten Kanten (Pfeilen) von Plätzen zu Transitionen und
- einer Menge $G \subseteq (T \times P)$ an gerichteten Kanten (Pfeilen) von Transitionen zu Plätzen.

Abbildung 2.1: Graphische Darstellung von Plätzen und Transitionen

Werden die Kanten in einem Netz mit Gewichten versehen, handelt es sich um ein Petri-Netz. Diese Gewichte müssen positive ganze Zahlen sein, d.h. sie müssen mindestens den Wert eins haben.

Definition 2.2

Das Tupel (P, T, F, G, f) ist ein **Petri-Netz**, wenn

- (P, T, F, G) ein Netz ist und
- $f: (F \cup G) \to \mathbb{N}$ eine Kantengewichtsfunktion, die jeder Kante eine positive ganze Zahl zuweist, wobei $(p_i \to t_j)$ die Kante von Platz $p_i \in P$ zur Transition $t_j \in T$ bezeichnet mit $f(p_i \to t_j)$ als zugehöriges Kantengewicht und $(t_j \to p_i)$ bezeichnet die Kante von t_j zu p_i mit dem Gewicht $f(t_j \to p_i)$.[1]

Jeder Platz kann eine nichtnegative Anzahl **Token** enthalten. Diese Token werden graphisch durch kleine, schwarze Punkte oder durch eine Zahl im inneren des Platzes gekennzeichnet (siehe Abbildung 2.1). Eine konkrete Festlegung der Tokenanzahl in einem Platz wird **Platzmarkierung** genannt und eine konkrete Festlegung der Tokenanzahlen aller Plätze wird **Petri-Netz-Markierung** genannt. Sind in einem Petri-Netz alle Plätze markiert - auch 0

[1] Für das Gewicht $f\left((p_i \to t_j)\right)$ der Kante $(p_i \to t_j)$ wird im Folgenden die vereinfachte Schreibweise $f(p_i \to t_j)$ verwendet.

Token ist eine Markierung - nennt man es **markiertes Petri-Netz** (siehe Abbildung 2.2). Allerdings wird im Folgenden auf diesen Zusatz verzichtet, da nur markierte Petri-Netze betrachtet werden.

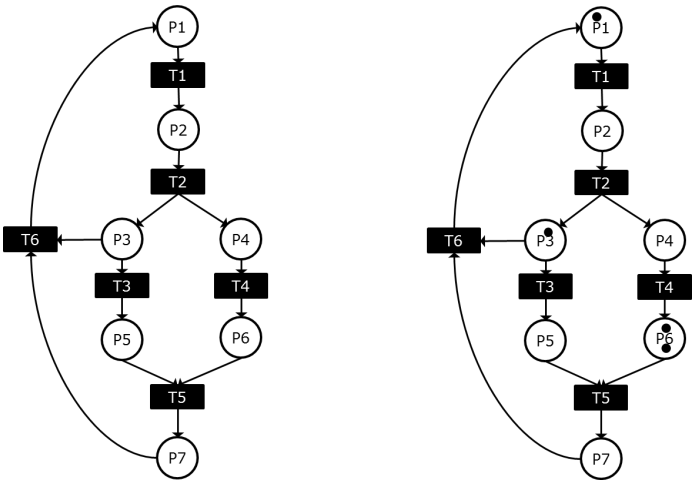


Abbildung 2.2: Nicht-markiertes (links) und markiertes Petri-Netz (rechts) (vgl. David und Alla 2010)[1]

Definition 2.3

Das Tupel (P, T, F, G, f) ist ein Petri-Netz. Eine Zuordnung $m: P \to \mathbb{N}_0$ wird **Petri-Netz-Markierung** genannt und ordnet jedem Platz $p_i \in P$ eine konkrete Tokenanzahl $m(p_i)$ zu, die **Platzmarkierung** genannt wird. Die Zuordnung $m_0: P \to \mathbb{N}_0$ ist die **Anfangsmarkierung** eines Petri-Netzes und das Tupel (P, T, F, G, f, m_0) wird **(anfangsmarkiertes) Petri-Netz** genannt.[2]

Definition 2.4

Die **Menge der Inputs** eines Petri-Netz-Elements $x \in (P \cup T)$ ist definiert als

$$Y_{in}(x) := \{y \in (P \cup T) | (y \to x) \in (F \cup G)\}$$

und die **Menge der Outputs** ist definiert als

$$Y_{out}(x) := \{y \in (P \cup T) | (x \to y) \in (F \cup G)\}.$$

Die Menge aller Inputplätze einer Transition $t_j \in T$ wird mit $P_{in}(t_j) \subseteq P$ bezeichnet und die Menge alle Outputplätze mit $P_{out}(t_j) \subseteq P$. Die Menge $T_{in}(p_i) \subseteq T$ umfasst alle Inputtransitionen eines Platzes $p_i \in P$ und $T_{out}(p_i) \subseteq T$ alle Outputtransitionen (siehe Abbildung 2.3). Die Anzahl an Inputs wird mit $n_{in}(x)$ bezeichnet und die Anzahl an Outputs mit $n_{out}(x)$.

[1] Wenn in den Abbildungen dieser Arbeit die Pfeile eines Petri-Netzes nicht mit einer Gewichtung versehen worden sind, dann ist diese stets eins.
[2] Da in dieser Arbeit nur anfangsmarkierte Petri-Netze vorkommen, werden diese im Folgenden kurz als Petri-Netze bezeichnet.

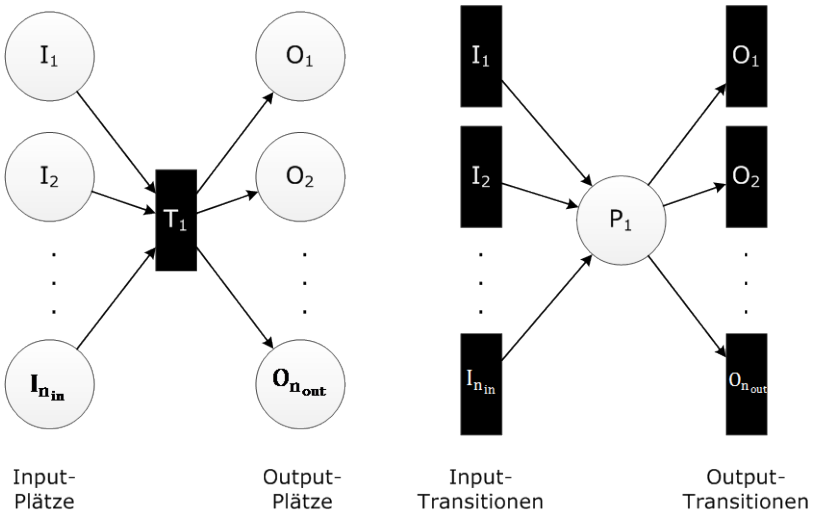


Abbildung 2.3: Input- und Outputplätze (links) und Input- und Outputtransitionen (rechts)

Eine Transition ist aktiv, wenn alle Inputplätze mindestens so viele Token haben wie das Kantengewicht.

Definition 2.5

Das Tupel (P, T, F, G, f, m_0) ist ein Petri-Netz. Eine Transition $t_j \in T$ ist **aktiv** genau dann, wenn

$$\forall p_i \in P_{in}(t_j) : m(p_i) \geq f(p_i \rightarrow t_j),$$

wobei $m(p_i)$ die Markierung von Platz p_i ist. Die Menge aller aktiven Inputtransitionen eines Platzes $p_i \in P$ wird mit $TA_{in}(p_i)$ bezeichnet und die Menge aller aktiven Outputtransitionen mit $TA_{out}(p_i)$.

Aktive Transitionen können durch ihre Inputplätze freigeschaltet werden. Wenn eine Transition von allen ihren Inputplätzen freigeschaltet wurde, ist sie **feuerbar**. Möglicherweise hat ein Platz nicht genügend Token, um alle aktiven Outputtransitionen gleichzeitig freizuschalten. In diesem Fall liegt ein genereller Konflikt vor, der gelöst werden muss, um eine Simulation zu ermöglichen.

Definition 2.6

Das Tupel (P, T, F, G, f, m_0) ist ein Petri-Netz. Ein Platz $p_i \in P$ mit der Markierung $m(p_i)$ hat einen **generellen Konflikt**, wenn

$$m(p_i) < \sum_{t_j \in TA_{out}(p_i)} f(p_i \rightarrow t_j)\,.^{1}$$

[1] Ein genereller Konflikt kann nur auftreten wenn die Menge $TA_{out}(p_i)$ zwei oder mehr Transitionen enthält. Dies wird durch diese Bedingung sichergestellt und muss nicht zusätzlich angegeben werden.

In Abbildung 2.4 haben die Plätze P_1 and P_2 einen generellen Konflikt. P_1 kann entweder T_1 oder T_2 freischalten und P_2 kann entweder T_2 oder T_3 freischalten. Die Transition T_4 ist keine Option, da sie durch die fehlenden Token in P_3 nicht aktiv ist.

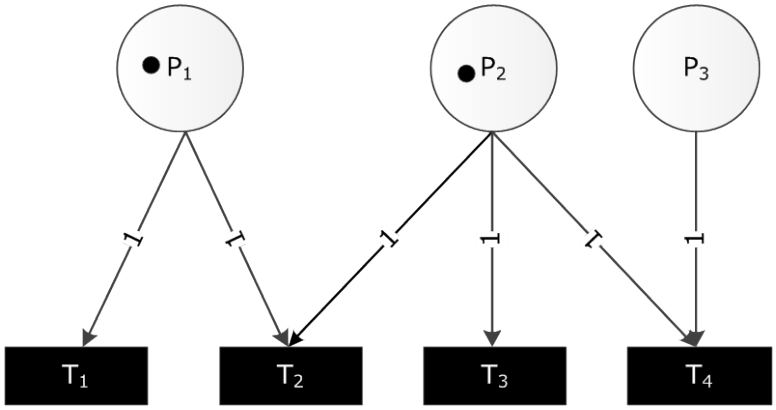


Abbildung 2.4: Genereller Konflikt von P_1 und P_2

2.2 KONFLIKTLÖSUNGEN

Unterschiedlichste Lösungen des generellen Konflikts in Definition 2.6 sind denkbar. Zunächst muss unterschieden werden, ob jeder Platz seinen Konflikt alleine löst, ohne Kenntnis darüber, welche Transitionen von den übrigen Plätzen freigeschaltet werden, oder ob die Plätze ihre Konflikte gemeinsam lösen.

In der Abbildung 2.5 sind beide Transitionen aktiv und die Plätze $P2$ und $P3$ haben einen generellen Konflikt. Beide müssen die Entscheidung treffen, ob sie $T1$ oder $T2$ freischalten. Diese Entscheidung kann jeder Platz alleine ohne Kenntnis über die Entscheidung des anderen Platzes treffen. Dann könnte es passieren, dass $P2$ beispielsweise $T1$ freischaltet und $P3$ die Transition $T2$. In dieser Situation könnte keine der beiden Transitionen feuern (siehe Definition 2.8). Das Vorgehen, dass jeder Platz seinen Konflikt alleine löst ohne Kenntnis der Entscheidungen der anderen Plätze, wird im Folgenden **lokale Konfliktlösung** genannt.

Anderseits können die Plätze ihre Konflikte auch gemeinsam lösen. Dann würden sich $P2$ und $P3$ entweder darauf einigen, dass sie entweder $T1$ oder $T2$ freischalten. Im Gegensatz zur lokalen Konfliktlösung, wird bei diesem Vorgehen, das im Folgenden **globale Konfliktlösung** genannt wird, auf jeden Fall eine der Transitionen feuerbar.

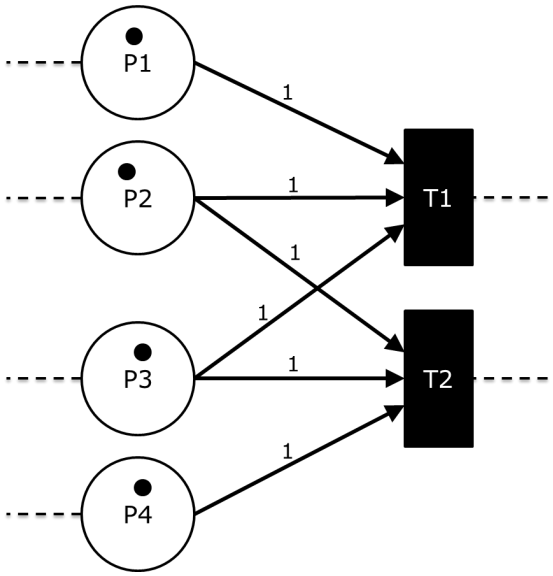


Abbildung 2.5: Lokale oder globale Konfliktlösung von P2 und P3

2.2.1 LOKALE KONFLIKTLÖSUNG

Bei der lokalen Konfliktlösung wird jeder Kante ausgehend von einem Platz zu einer Transition eine Bewertung zugewiesen. Basierend auf dieser Bewertung, schaltet der Platz so viele Transitionen wie möglich frei. Diese Bewertungen können Wahrscheinlichkeiten, Nutzen oder Nutzenquotienten (Nutzen pro Token) sein.

Definition 2.7

Das Tupel $PN = (P, T, F, G, f, e, \wp, m_0)$ ist ein **lokal kantenbewertetes Petri-Netz**, wenn

- (P, T, F, G, f, m_0) ein Petri-Netz ist,
- $e: P \rightarrow \{1,2,3\}$ eine Lösungsfunktion ist, die jedem Platz einen Lösungstyp zuordnet (1-Wahrscheinlichkeit, 2-Nutzen, 3-Nutzenquotient) und
- $\wp: F \rightarrow \{[0,1]: e(p_i) = 1, \mathbb{R}_{\geq 0}: e(p_i) = 2 \vee e(p_i) = 3\}$ eine Kantenbewertungsfunktion ist, die jeder Kante ausgehend von einem Platz $p_i \in P$ zu einer Transition $t_j \in T_{out}(p_i)$ eine Bewertung $\wp(p_i \rightarrow t_j)$ zuordnet.[1] [2]

[1] Für die Bewertung $\wp\left((p_i \rightarrow t_j)\right)$ der Kante $(p_i \rightarrow t_j)$ wird im Folgenden die vereinfachte Schreibweise $\wp(p_i \rightarrow t_j)$ verwendet.

[2] Bei der Konfliktlösung durch Wahrscheinlichkeiten, können den Kanten beliebig Werte aus dem Intervall [0,1] zugeordnet werden. Sinnvoll wäre natürlich, wenn gilt $\sum_{t_j \in T_{out}(p_i)} \wp(p_i \rightarrow t_j) = 1$. Die Wahrscheinlichkeiten werden aber bei der Konfliktlösung sowieso transformiert, so dass gilt $\sum_{t_j \in TA_{out}(p_i)} \wp(p_i \rightarrow t_j) = 1$.

In Abbildung 2.6 hat $P1$ einen generellen Konflikt, der mithilfe von Nutzenbewertungen gelöst werden soll. Die Transition $T3$ hat den höchsten Nutzen und wird deshalb als erstes freigeschaltet. Danach kann keine weitere Transition freigeschaltet werden, da $P1$ nur einen Token hat.

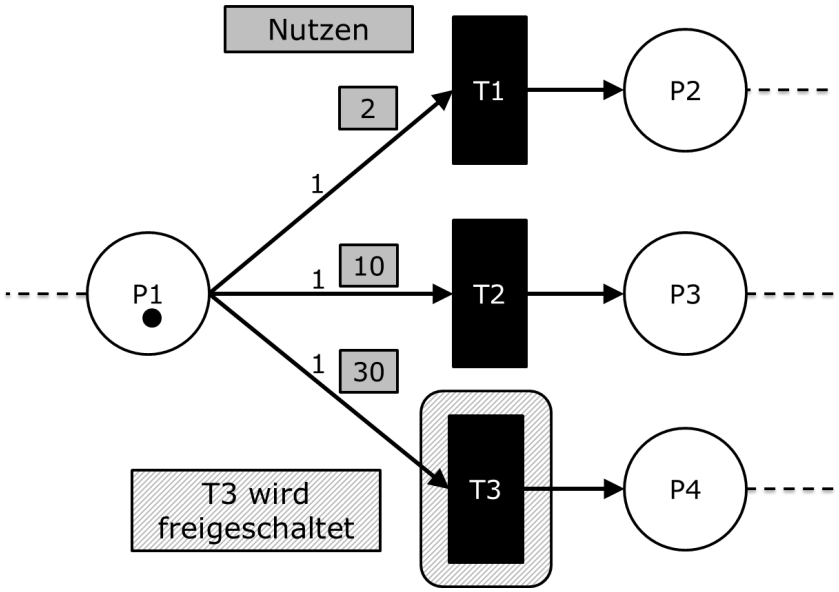


Abbildung 2.6: Lokale Konfliktlösung mithilfe von Nutzenbewertungen

In Abbildung 2.7 hat $P1$ einen generellen Konflikt, der mithilfe von Wahrscheinlichkeiten gelöst werden soll. Dazu werden die Wahrscheinlichkeiten, wie in der Abbildung dargestellt, in einem Kreisdiagramm aufgetragen. Anschließend wird eine über das Intervall $[0; 1]$ gleichverteilte Zufallszahl erzeugt, die die Transition festlegt, die als erstes freigeschaltet wird. In diesem Beispiel liegt die Zufallszahl 0.31 im Bereich von $T2$.

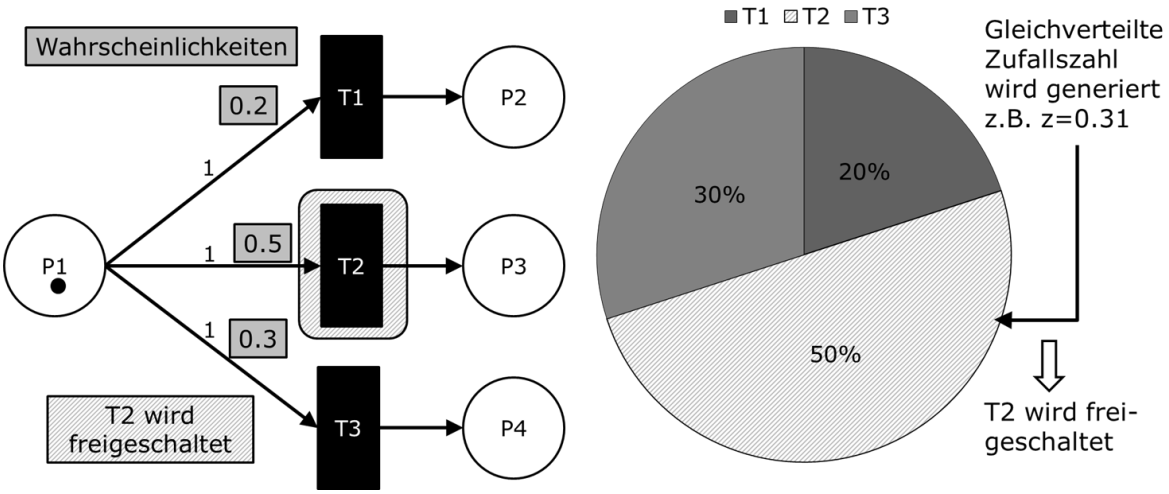


Abbildung 2.7: Lokale Konfliktlösung mithilfe von Wahrscheinlichkeiten

Eine aktive Transition ist genau dann feuerbar, wenn sie von allen Inputplätzen freigeschaltet wird.

Definition 2.8

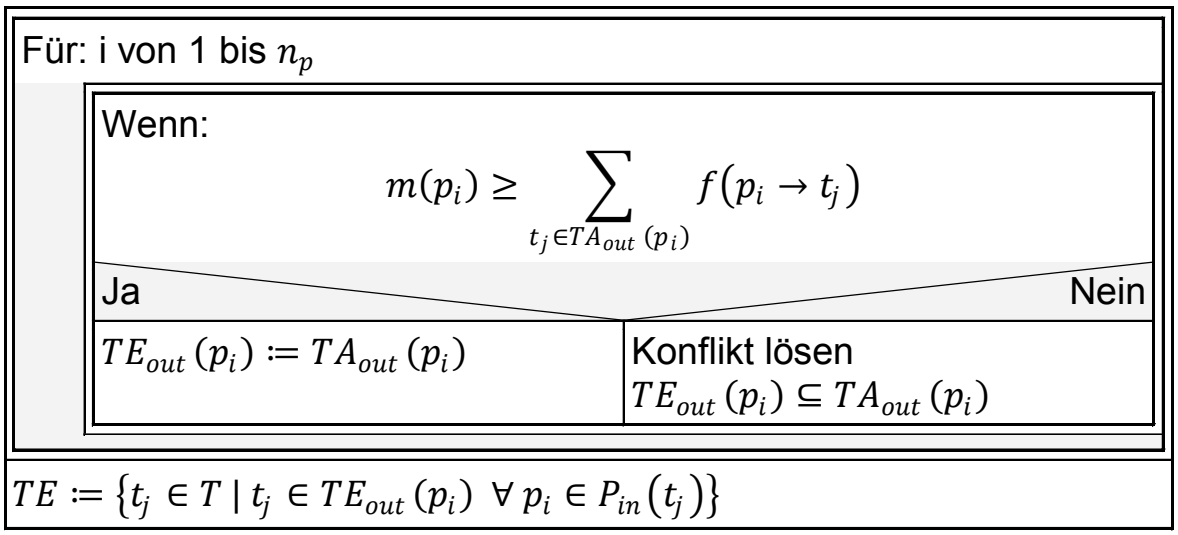
Sei PN ein lokal kantenbewertetes Petri-Netz. Eine aktive Transition $t_j \in T$ ist **feuerbar** genau dann, wenn

$$\forall\, p_i \in P_{in}(t_j):\ t_j \in TE_{out}(p_i),$$

wobei die Menge $TE_{out}(p_i)$ die freigeschalteten (enabled) Outputtransitionen von Platz p_i enthält.

In den nachfolgenden Algorithmen wird das Vorgehen bei der lokalen Konfliktlösung mithilfe von Struktogrammen nach Nassi-Shneiderman (DIN 66261, Nassi und Shneiderman 1973) beschrieben (siehe z.B. Pomberger und Dobler 2008). Die Konfliktlösungen mithilfe von Nutzen und Nutzenquotienten sind in Algorithmus 2.3 zusammengefasst. Der einzige Unterschied besteht darin, dass bei der Konfliktlösung durch Nutzenquotienten, der den Kanten zugewiesene Nutzen noch durch das Kantengewicht geteilt wird. Auf diese Weise erhält man einen Nutzen pro Token, der zur Lösung des Konflikts benötigt wird.

Algorithmus 2.1: Der lokale Freischaltungsprozess



Algorithmus 2.2: Lokale Konfliktlösung mithilfe von Wahrscheinlichkeiten

Für: alle $p_i \in P$ mit $e(p_i) = 1$

Wenn:

$$m(p_i) \geq \sum_{t_j \in TA_{out}(p_i)} f(p_i \to t_j)$$

Ja	Nein
$TE_{out}(p_i) \coloneqq TA_{out}(p_i)$	$m \coloneqq m(p_i)$ $TE_{out}(p_i) \coloneqq \{\ \}$ $TA \coloneqq TA_{out}(p_i)$ Für: i von 1 bis $\lvert TA_{out}(p_i)\rvert$ $sum \coloneqq \sum_{t_k \in TA} \wp(p_i \to t_k)$ $\wp(p_i \to t_k) \coloneqq \begin{cases} \frac{\wp(p_i \to t_k)}{sum}, & t_k \in TA \\ 0, & t_k \notin TA \end{cases}$, $\wp_{kum}(p_i \to t_k) \coloneqq \sum_{q=1}^{k} \wp(p_i \to t_q),\ k = 1, \dots, n_{out}(p_i)$ Generiere gleichverteilte Zufallszahl $z \in [0;1]$ Finde erstes Element $\wp_{kum}(p_i \to t) > z$ $TA \coloneqq TA\ /\{t\}$ Wenn: $m - f(p_i \to t) \geq 0$

Ja	Nein
$TE_{out}(p_i) \coloneqq TE_{out}(p_i) \cup \{t\}$ $m \coloneqq m - f(p_i \to t)$	

Algorithmus 2.3: Lokale Konfliktlösung mithilfe von Nutzen\Nutzenquotienten

Für: alle $p_i \in P$ mit $e(p_i) = 2 \vee e(p_i) = 3$

Wenn:

$$m(p_i) \geq \sum_{t_j \in TA_{out}(p_i)} f(p_i \rightarrow t_j)$$

Ja	Nein
$TE_{out}(p_i) \coloneqq TA_{out}(p_i)$	$m \coloneqq m(p_i)$ $TE_{out}(p_i) \coloneqq \{\ \}$ $TA \coloneqq TA_{out}(p_i)$ $\tilde{p}(p_i \rightarrow t_k) = \begin{cases} p(p_i \rightarrow t_k) & e(p_i) = 2 \\ \frac{p(p_i \rightarrow t_k)}{f(p_i \rightarrow t_k)} & e(p_i) = 3 \end{cases}, k = 1, \dots, n_{out}(p_i)$

Für: i von 1 bis $|TA_{out}(p_i)|$

$p_{max} \coloneqq \max_{t_k \in TA} \{\tilde{p}(p_i \rightarrow t_k)\}$

$T_{max} \coloneqq \{t \in TA \mid \tilde{p}(p_i \rightarrow t) = p_{max}\}$

Wähle ein $t \in T_{max}$

$TA \coloneqq TA \ / \{t\}$

Wenn: $m - f(p_i \rightarrow t) \geq 0$

Ja	Nein
$TE_{out}(p_i) \coloneqq TE_{out}(p_i) \cup t$ $m \coloneqq m - f(p_i \rightarrow t)$	

Das Problem der Konfliktlösung mithilfe von Nutzen kann auch als das aus der kombinatorischen Optimierung bekannte **Rucksackproblem** aufgefasst werden:

Aus einer Menge von n Gegenständen soll eine Teilmenge ausgewählt werden. Jeder dieser Gegenstände hat ein Gewicht f_j und einen Nutzen p_j. Nun soll der Gesamtnutzen der Gegenstände maximiert werden, wobei deren Gesamtgewicht eine vorgegebene Gewichtsschranke m nicht überschreiten darf:

$$z = \max\left(\sum_{j=1}^{n} p_j \cdot x_j\right)$$

unter den Nebenbedingungen

$$\sum_{j=1}^{n} f_j \cdot x_j \leq m$$
$$x_j \in \{0,1\} \quad \forall j \in \{1, \dots, n\}$$

Hierbei bedeutet $x_j = 1$, dass der Gegenstand im Rucksack mitgenommen wird und $x_j = 0$, dass er nicht mitgenommen wird.

Das Konfliktproblem eines Platzes in einem Petri-Netz lässt sich als Rucksackproblem formulieren:

Aus einer Menge von aktiven Transitionen soll eine Teilmenge freigeschaltet werden. Jede dieser Transitionen hat ein Kantengewicht $f(p_i \rightarrow t)$ und einen Nutzen $\wp(p_i \rightarrow t)$. Nun soll der Gesamtnutzen der freigeschalteten Transitionen maximiert werden, wobei deren Kantengewichtsumme die aktuelle Markierung $m(p_i)$ nicht überschreiten darf.

Dazu wird das folgende Maximierungsproblem aufgestellt:

$$z = \max\left(\sum_{j \in T_{out}(p_i)} \wp(p_i \rightarrow t_j) \cdot x_j\right)$$

unter den Nebenbedingungen

$$\sum_{j \in T_{out}(p_i)} f(p_i \rightarrow t_j) \cdot x_j \leq m(p_i)$$
$$x_j \in \{0,1\}$$
$$t_j \notin TA_{out}(p_i) \Rightarrow x_j = 0$$

Als geeignete Lösungsmethode für Rucksackprobleme haben sich **Branch-and-Bound-Verfahren** erwiesen. In Kapitel 3.3.1 wird das Branch-and-Bound-Verfahren ausführlich erläutert.

2.2.2 Globale Konfliktlösung

Konflikte werden global auch mithilfe von Bewertungen gelöst. Hierbei gibt es, im Gegensatz zur lokalen Lösung, zwei Möglichkeiten. Entweder werden die Kanten bewertet oder die

Transitionen. Außerdem muss für das ganze Petri-Netz ein einheitlicher Lösungstyp (Wahrscheinlichkeit, Nutzen oder Nutzenquotient) gelten.

Definition 2.9

Das Tupel $PN = (P, T, F, G, f, e, \wp, m_0)$ ist ein **global kantenbewertetes Petri-Netz**, wenn

- (P, T, F, G, f, m_0) ein Petri-Netz ist,
- $e \rightarrow \{1,2,3\}$ eine globale Lösungsfunktion ist, die dem Petri-Netz einen globalen Lösungstyp zuordnet (1-Wahrscheinlichkeit, 2-Nutzen, 3-Nutzenquotient) und
- $\wp: F \rightarrow \{[0,1]: e = 1,\ \mathbb{R}_{\geq 0}:\ e = 2 \vee e = 3\}$ eine Kantenbewertungsfunktion ist, die jeder Kante ausgehend von einem Platz $p_i \in P$ zu einer Transition $t_j \in T_{out}(p_i)$ eine Bewertung $\wp(p_i \rightarrow t_j)$ zuordnet.

Definition 2.10

Das Tupel $PN = (P, T, F, G, f, e, w, m_0)$ ist ein **global transitionenbewertetes Petri-Netz**, wenn

- (P, T, F, G, f, m_0) ein Petri-Netz ist,
- $e \rightarrow \{1,2\}$ eine globale Lösungsfunktion ist, die dem Petri-Netz einen globalen Lösungstyp zuordnet (1-Wahrscheinlichkeit, 2-Nutzen) und
- $w: T \rightarrow \{[0,1]: e = 1,\ \mathbb{R}_{\geq 0}:\ e = 2\}$ eine Transitionenbewertungsfunktion ist, die jeder Transition $t_j \in T$ des Petri-Netzes eine Bewertung $w(t_j)$ zuordnet.

Das globale kantenbewertete Petri-Netz wird für die Konfliktlösung in ein transitionenbewertetes umgewandelt. Dazu können bei den Lösungstypen Nutzen und Nutzenquotienten entweder alle Kantenbewertungen der Input-Plätze aufsummiert werden oder es kann das arithmetische Mittel aus ihnen gebildet werden. Beim Lösungstyp Wahrscheinlichkeiten werden alle Input-Kantenbewertungen multipliziert und anschließend wieder in das Intervall $[0; 1]$ transformiert. Es ergeben sich für diese vier Lösungstypen folgende **Transformationsregeln**[1]:

Lösungstyp „Wahrscheinlichkeiten“ $e = 1$:

$$(TR1) \qquad w(t_j) = \prod_{p_i \in P_{in}(t_j)} \wp(p_i \rightarrow t_j), \quad j = 1, \dots, n_t$$

[1] Es sind auch andere Transformationsregeln möglich. Diese Arbeit beschränkt sich aber auf diese Regeln.

Lösungstyp „Nutzen“ $e = 2$:

$$(TR2a) \qquad \mathcal{w}(t_j) = \frac{1}{n_{in}(t_j)} \cdot \sum_{p_i \in P_{in}(t_j)} \mathcal{p}(p_i \to t_j), \quad j = 1, \dots, n_t$$

$$(TR2b) \qquad \mathcal{w}(t_j) = \sum_{p_i \in P_{in}(t_j)} \mathcal{p}(p_i \to t_j), \quad j = 1, \dots, n_t$$

Lösungstyp „Nutzenquotienten“ $e = 3$:

$$(TR3a) \qquad \mathcal{w}(t_j) = \frac{1}{n_{in}(t_j)} \cdot \sum_{p_i \in P_{in}(t_j)} \frac{\mathcal{p}(p_i \to t_j)}{f(p_i \to t_j)}, \quad j = 1, \dots, n_t$$

$$(TR3b) \qquad \mathcal{w}(t_j) = \sum_{p_i \in P_{in}(t_j)} \frac{\mathcal{p}(p_i \to t_j)}{f(p_i \to t_j)}, \quad j = 1, \dots, n_t$$

In einem globalen Petri-Netz sind alle freigeschalteten Transitionen stets auch feuerbar, da sie immer von allen Plätzen gleichzeitig freigeschaltet werden. Dies wird im Folgenden **globale Freischaltung** genannt.

Definition 2.11

Sei PN ein global kantenbewertetes oder transitionenbewertetes Petri-Netz. Eine aktive Transition $t_j \in T$ ist **feuerbar** genau dann, wenn

$$t_j \in TE,$$

wobei TE die Menge der global freigeschalteten (enabled) Transitionen des Petri-Netzes ist.

Abbildung 2.8 zeigt auf der linken Seite ein global kantenbewertetes Petri-Netz, in dem die Konflikte von $P1$ und $P2$ mithilfe von Wahrscheinlichkeiten gelöst werden. Dazu werden im ersten Schritt die Wahrscheinlichkeiten an den Kanten durch die Transformationsregel $(TR1)$ in Transionenbewertungen umgewandelt. Für die Transitionenbewertungen von $T1$ und $T2$ ergeben sich

$$\mathcal{w}(T1) = \prod_{p_i \in \{P1, P2, P3\}} \mathcal{p}(p_i \to T1) = 1 \cdot 0.5 \cdot 0.25 = 0.125$$

$$\mathcal{w}(T2) = \prod_{p_i \in \{P2, P3, P4\}} \mathcal{p}(p_i \to T2) = 0.5 \cdot 0.75 \cdot 1 = 0.375.$$

Anschließend werden diese Transitionenbewertungen für die weiteren Berechnungen wieder in das Intervall $[0;1]$ transformiert. Somit wird $T1$ mit einer Wahrscheinlichkeit von 0.25 global freigeschaltet und feuerbar und $T2$ mit einer Wahrscheinlichkeit von 0.75. Ähnlich wie bei der lokalen Freischaltung mit Wahrscheinlichkeiten (siehe Abbildung 2.7 und Algorithmus 2.2), werden die Transitionen wieder nach dem Zufallsprinzip ausgewählt und auf ihre Freischaltbarkeit mit den folgenden zwei Kriterien

$$t \in TA$$

$$m_i - f(p_i \to t) \geq 0 \quad \forall p_i \in P_{in}(t)$$

überprüft, wobei m_i die Markierung von p_i ist nach Abzug der Kantengewichte der Transitionen, die bereits in die Menge der freigeschalteten Transitionen aufgenommen wurden.

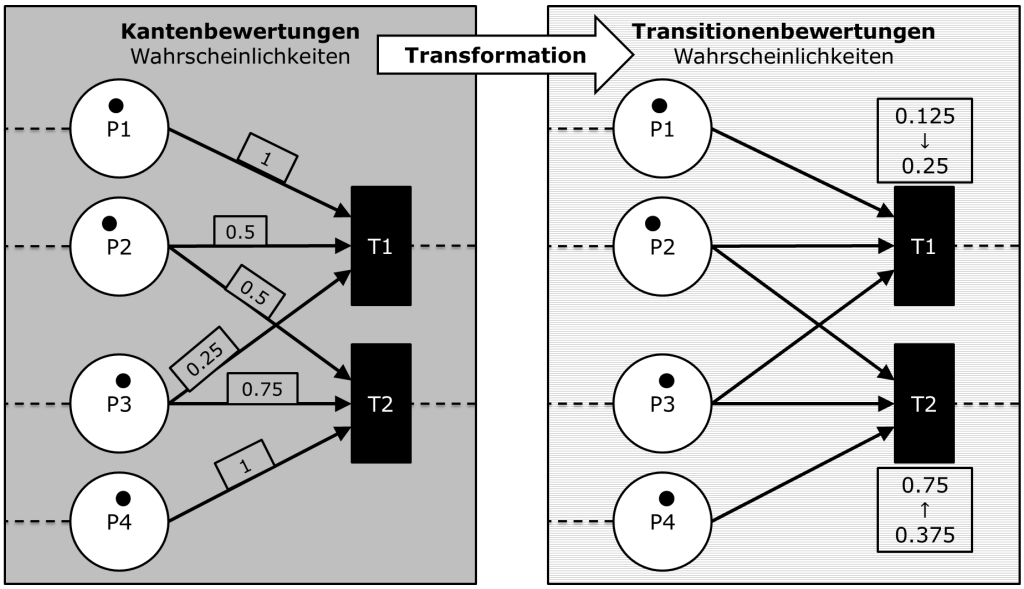


Abbildung 2.8: Global kantenbewertetes Petri-Netz mit Wahrscheinlichkeiten (links) und Konfliktlösung durch Transitionenbewertungen (rechts)

Abbildung 2.9 zeigt auf der linken Seite ein global kantenbewertetes Petri-Netz, in dem die Konflikte von $P1$ und $P2$ mithilfe von Nutzenbewertungen gelöst werden. Dazu werden im ersten Schritt die Nutzenbewertungen an den Kanten durch die Transformationsregel $(TR2a)$ in Transionenbewertungen umgewandelt. Für die Transitionenbewertungen von $T1$ und $T2$ ergeben sich

$$w(T1) = \frac{1}{3} \cdot \sum_{p_i \in \{P1,P2,P3\}} p(p_i \to T1) = \frac{1}{3} \cdot (10 + 20 + 30) = 20$$

$$w(T2) = \frac{1}{4} \cdot \sum_{p_i \in \{P2,P3,P4,P5\}} p(p_i \to T2) = \frac{1}{4} \cdot (30 + 10 + 10 + 10) = 15.$$

Somit wird die Transition $T1$ global freigeschaltet und ist feuerbar, da sie den größeren Nutzen hat.

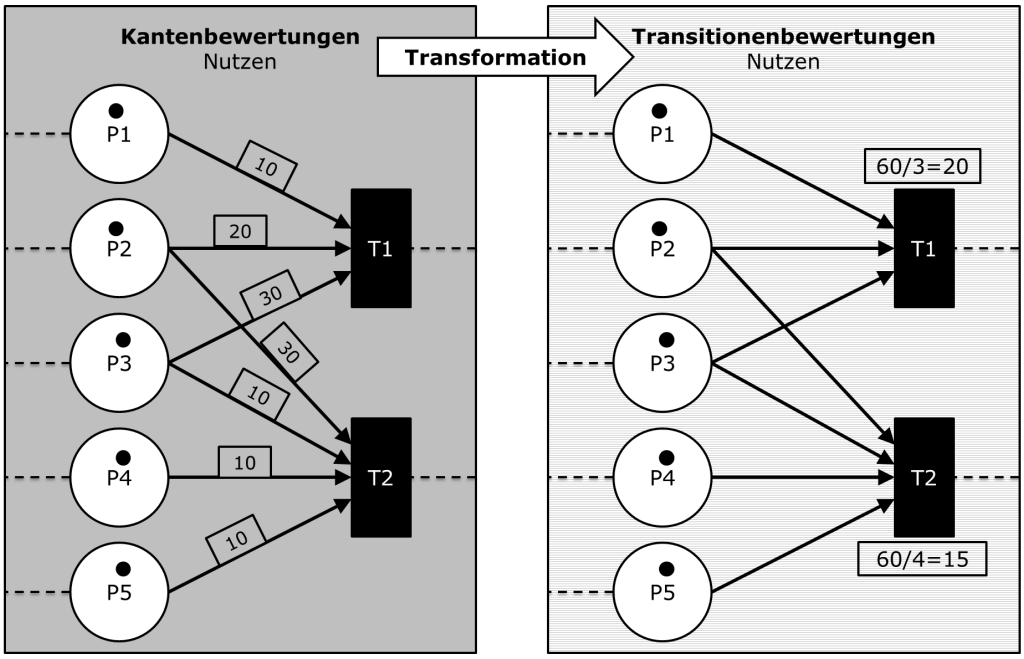


Abbildung 2.9: Global kantenbewertetes Petri-Netz mit Nutzen (links) und Konfliktlösung durch Transitionenbewertungen (rechts)

Liegen Bewertungen in Form von Nutzen vor, können globale Konflikte auch als Rucksackproblem formuliert werden, da sich das Rucksackproblem auf mehrere Dimensionen verallgemeinern lässt. In der Literatur ist dieses Optimierungsproblem unter dem Namen **mehrdimensionales Rucksackproblem** (englisch. **multidimensional knapsack problem** (MKP)) bekannt:

$$z = \max\left(\sum_{j=1}^{n} \wp_j \cdot x_j\right)$$

unter den Nebenbedingungen

$$\sum_{j=1}^{n} f_{i,j} \cdot x_j \leq m_i \quad \forall i \in \{1, \dots, l\}$$

$$x_j \in \{0,1\} \quad \forall j \in \{1, \dots, n\}$$

Ökonomische Anwendung findet das MKP beispielsweise in der Planung von Investitionen über l Perioden. Hierbei sollen aus n möglichen Projekten mit gegebenem Kapitalwert $\wp_j$ einige ausgewählt werden, so dass die Summe der Kapitalwerte maximiert wird. Um die Projekte zu finanzieren, steht in jeder Periode ein Budget m_i zur Verfügung. Die

Nettozahlung für Projekt j in Periode i beträgt $f_{i,j}$ und $x_j = 1$ bedeutet, Projekt j wird durchgeführt und $x_j = 0$ bedeutet, dass es nicht durchgeführt wird (vgl. Domschke und Drexl 2005).

MKP können mithilfe einer **Branch-and-Bound-Methode** gelöst werden. Dazu wird das zugehörige Optimierungsproblem zur Konfliktlösung aufgestellt. Im Fall von Kantenbewertungen müssen diese zunächst mithilfe der Transformationsregeln in Transitionenbewertungen umgewandelt werden. Das Maximierungsproblem lautet:

$$z = \max\left(\sum_{j=1}^{n_t} w(t_j) \cdot x_j\right)$$

unter den Nebenbedingungen

$$\sum_{j \in T_{out}(p_i)} f(p_i \rightarrow t_j) \cdot x_j \leq m(p_i) \quad \forall p_i \in P$$

$$x_j \in \{0,1\}$$

$$t_j \notin TA \Rightarrow x_j = 0.$$

In Kapitel 3.3.1 wird das Branch-and-Bound-Verfahren ausführlich erläutert. In den folgenden Algorithmen wird das Vorgehen bei der globalen Konfliktlösung mithilfe von Struktogrammen nach Nassi-Shneiderman (DIN 66261, Nassi und Shneiderman 1973) beschrieben.[1]

[1] Es ist zu beachten, dass Nutzenquotienten zur globalen Konfliktlösung nur herangezogen werden können, wenn ein kantenbewertetes Petri-Netz vorliegt.

Algorithmus 2.4: Globale Konfliktlösung mithilfe von Wahrscheinlichkeiten

Wenn:

$$m(p_i) \geq \sum_{t_j \in TA_{out}(p_i)} f(p_i \rightarrow t_j) \quad \forall p_i \in P$$

Ja	Nein
$TE \coloneqq TA$	$TE := \{\ \}$ $m_i \coloneqq m(p_i),\ i = 1, \dots, n_p$ Wenn: kantenbewertetes Petri-Netz?

Ja	Nein
Berechnung der Transitionenbewertungen $w(t_j),\ j = 1, \dots, n_t$ mit (TR1)	

Für: j von 1 bis $|TA|$

$$sum \coloneqq \sum_{t_k \in TA} w(t_k)$$

$$w(t_k) \coloneqq \begin{cases} \frac{w(t_k)}{sum}, & t_k \in TA \\ 0, & t_k \notin TA \end{cases}$$

$$w_{kum}(t_k) \coloneqq \sum_{q=1}^{k} w(t_k),\ k = 1, \dots, n_t$$

Generiere gleichverteilte Zufallszahl $z \in [0; 1]$

Finde erstes Element $w_{kum}(t) > z$

$TA \coloneqq TA\ /\{t\}$

Wenn: $(m_i - f(p_i \rightarrow t) \geq 0)\ \ \forall p_i \in P_{in}(t)$

Ja	Nein
$TE := TE \cup \{t\}$ $m_i \coloneqq m_i - f(p_i \rightarrow t)\ \ \forall p_i \in P_{in}(t)$	

Algorithmus 2.5: Globale Konfliktlösung mithilfe von Nutzen\Nutzenquotienten

Wenn:

$$m(p_i) \geq \sum_{t_j \in TA_{out}(p_i)} f(p_i \rightarrow t_j) \quad \forall p_i \in P$$

Ja	Nein
$TE \coloneqq TA$	$TE := \{\ \}$ $m_i \coloneqq m(p_i),\ i = 1, \dots, n_p$ Wenn: kantenbewertetes Petri-Netz?

Ja	Nein
Berechnung der Transitionenbewertungen $w(t_j),\ j = 1, \dots, n_t$ mit (TR2a), (TR2b), (TR3a) oder (TR3b)	

Für: j von 1 bis $|TA|$

$w_{max} \coloneqq \max_{t_k \in TA}\{w(t_k)\}$

$T_{max} \coloneqq \{t \in TA \mid w(t) = w_{max}\}$

Wähle ein $t \in T_{max}$

$TA \coloneqq TA\ /\{t\}$

Wenn: $(m_i - f(p_i \rightarrow t) \geq 0)\ \ \forall p_i \in P_{in}(t)$

Ja	Nein
$TE := TE \cup \{t\}$ $m_i \coloneqq m_i - f(p_i \rightarrow t)\ \ \forall p_i \in P_{in}(t)$	

2.3 FEUERUNGSPROZESS

Für die Auswahl, welche der feuerbaren Transitionen auch wirklich gefeuert werden, sind unterschiedliche **Feuerungsstrategien** möglich, wobei TF die Menge der feuernden Transitionen bezeichnet:

$(F1)$ Alle freigeschalteten Transitionen feuern $(TF \equiv TE)$.

$(F2)$ Eine der freigeschalteten Transitionen feuert, wobei die Auswahl zufällig getroffen wird $(TF = \{t\},\ t \in TE)$.

($F3$) Es gibt sogenannte heiße (hot) ($TE_h \subseteq TE$) und kalte (cold) ($TE_c \subseteq TE$) freigeschaltete Transitionen. Die heißen freigeschalteten Transitionen feuern auf jeden Fall ($TF_h = TE_h$) und bei den kalten Transitionen kann der Benutzer entscheiden, ob sie feuern oder nicht ($TF_c \subseteq TE_c, TF = TF_h \cup TF_c$).[1]

Für die dritte Feuerungsstrategie muss die Menge der Transitionen in heiße und kalte Transitionen aufgeteilt werden.

Definition 2.12

Das Tupel $bPN = (PN, T_h, T_c)$ ist ein **benutzerdefiniertes Petri-Netz**, wenn

- PN ein Petri-Netz nach Definition 2.7, Definition 2.9 oder Definition 2.10 ist,
- $T_h \subseteq T$ die Menge der heißen Transitionen ist, die, wenn sie freigeschaltet werden, auch feuern,
- $T_c \subseteq T$ die Menge der kalten Transitionen ist, die, wenn sie freigeschaltet werden, durch die Entscheidung des Benutzers feuern, wobei T in T_h und T_c zerlegt wird.

Wenn eine oder mehrere Transitionen feuern, müssen die Markierungen der Plätze neu berechnet werden. Hierbei wird die Kantengewichtssumme der feuernden Outputtransitionen von der derzeitigen Markierung abgezogen und die Kantengewichtssumme der feuernden Inputtransitionen wird hinzuaddiert.

Definition 2.13

Sei PN ein Petri-Netz. Eine feuerbare Transition $t \in T$ **feuert,** indem das Kantengewicht an Token von allen Inputplätzen abgezogen wird

$$m'(p_i) = m(p_i) - f(p_i \to t) \quad \forall\, p_i \in P_{in}(t)$$

und das Kantengewicht an Token bei allen Outputplätzen hinzuaddiert wird

$$m'(p_i) = m(p_i) + f(\mathrm{t} \to p_i) \quad \forall\, p_i \in P_{out}(t).$$

Die neue Markierung $m'(p_i)$ des Platzes $p_i \in P$ wird durch die folgende Gleichung berechnet

$$m'(p_i) = m(p_i) - \sum_{t_j \in TF_{out}(p_i)} f(p_i \to t_j) + \sum_{t_j \in TF_{in}(p_i)} f(t_j \to p_i)\,,$$

wobei $TF_{out}(p_i) \subseteq T$ die Menge aller feuernden Outputtransitionen ist und $TF_{in}(p_i) \subseteq T$ die Menge aller feuernden Inputtransitionen. Die Summen

$$\sum_{t_j \in TF_{out}(p_i)} f(p_i \to t_j)$$

[1] Es sind auch andere Feuerungsstrategien möglich. Diese Arbeit beschränkt sich aber auf diese Strategien.

und

$$\sum_{t_j \in TF_{in}(p_i)} f(t_j \to p_i)$$

werden als **Output-Feuersumme** bzw. **Input-Feuersumme** des jeweiligen Platzes p_i bezeichnet.

In Abbildung 2.10 sind die Transitionen $T1$, $T2$ und $T3$ feuerbar und werden gefeuert. Die neuen Markierungen sind im unteren Teil der Abbildung dargestellt. Die Transition $T4$ ist nicht aktiv, da der Platz $P41$ leer ist.

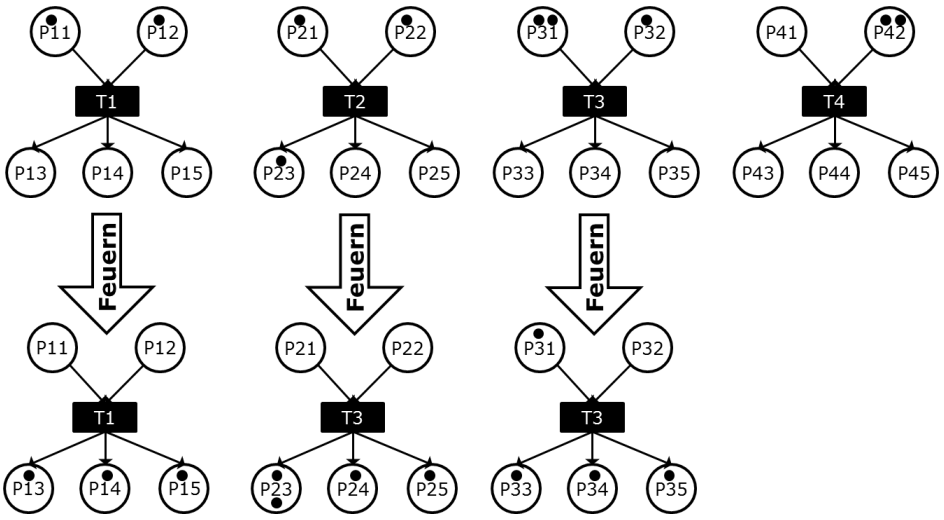


Abbildung 2.10: Beispiele von Transitionenfeuerungen (vgl. David und Alla 2010)

Das Petri-Netz in Abbildung 2.11 hat die Anfangsmarkierung $m_0 = (2,5,0,1,0,0,2)$. Auf der Grundlage von Definition 2.5 sind die Transitionen $T1$ und $T2$ aktiv, da

$T1$: $m(P1) = 2 \geq f(P1 \to T1) = 2 \ \wedge \ m(P2) = 5 \geq f(P2 \to T1) = 1$

$T2$: $m(P2) = 5 \geq f(P2 \to T2) = 1.$

Es besteht nach Definition 2.6 kein genereller Konflikt, da

$P1$: $m(P1) = 2 \geq f(P1 \to T1) = 2$

$P2$: $m(P2) = 5 \geq f(P2 \to T1) + f(P2 \to T2) = 2.$

Somit können $T1$ und $T2$ freigeschaltet werden und sind feuerbar. Sie feuern nebenläufig[1] indem

- ⇨ 2 Token von $P1$ und 2 Token von $P2$ entfernt werden **und**
- ⇨ 3 Token zu $P3$, 2 Token zu $P4$ und 2 Token zu $P5$ hinzuaddiert werden.

[1] Nebenläufiges Feuern bedeutet, dass es egal ist, ob erst $T1$ und dann $T2$ oder erst $T2$ und dann $T1$ oder beide gleichzeitig gefeuert werden. Man erhält immer das gleiche Ergebnis, d.h. die gleiche Markierung des Petri-Netzes.

Die neue Markierung des Petri-Netzes ist im unteren Teil der Abbildung dargestellt.

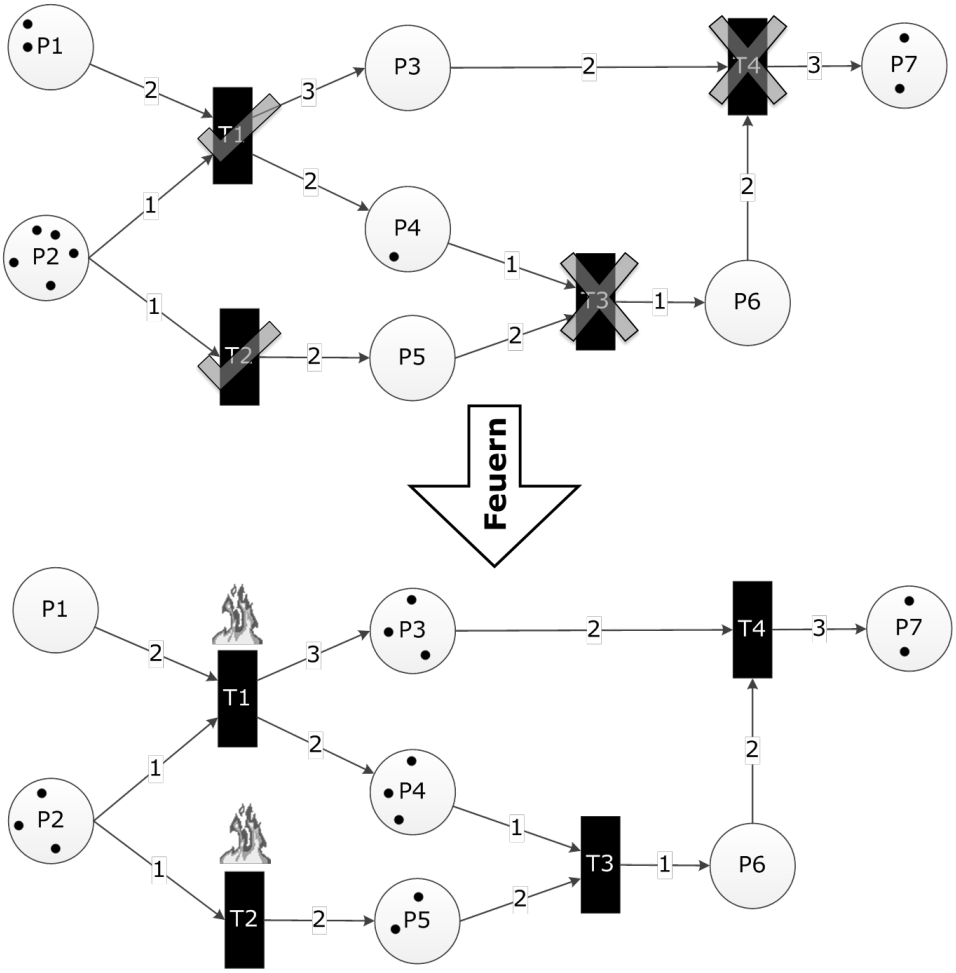


Abbildung 2.11: Beispiel eines Feuerungsprozesses: T1 und T2 sind feuerbar (oben) und werden gefeuert (unten)

2.4 Vereinfachungen und Erweiterungen

In diesem Abschnitt werden Vereinfachungen und Erweiterungen des grundlegenden Petri-Netz-Konzeptes aus Kapitel 2.1 vorgestellt.

2.4.1 Kapazitäten

Zunächst kann ein Petri-Netz mit **Kapazitäten** versehen werden, d.h. jeder Platz erhält eine untere und obere Schranke an Token, die er enthalten darf. Die Einführung von Kapazitäten

ist eine Vereinfachung des grundlegenden Konzeptes. Jedes Petri-Netz mit Kapazitäten kann wieder in ein Petri-Netz ohne Kapazitäten umgewandelt werden (siehe Abbildung 2.12).

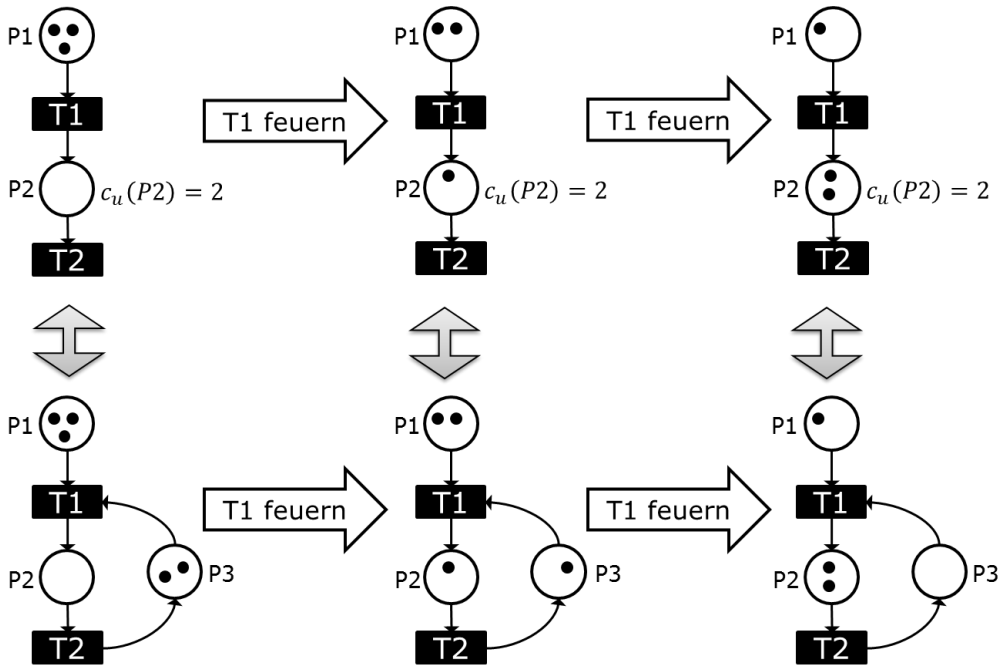


Abbildung 2.12: Petri-Netze mit Kapazitäten (oben) zur Vereinfachung der Darstellung im Vergleich zum grundlegenden Konzept (unten) (vgl. David und Alla 2010)

Definition 2.14

Das Tupel $cPN = (PN, c_l, c_u)$ ist ein **Petri-Netz mit Kapazitäten**, wenn

- PN ein Petri-Netz nach Definition 2.7, Definition 2.9, Definition 2.10 oder Definition 2.12 ist, wobei im Falle von Kantenbewertungen die Kantenbewertungsfunktion um die Kanten von den Transitionen zu den Plätzen erweitert werden muss:

$$\wp: (F \cup G) \rightarrow \{[0,1]: e(p_i) = 1, \mathbb{R}_{\geq 0}: \ e(p_i) = 2 \ \vee \ e(p_i) = 3\}$$

- $c_l: P \rightarrow \mathbb{N}_0$ eine Zuordnung ist, die jedem Platz $p_i \in P$ eine **minimale Kapazität** (lower capacity) $c_l(p_i)$ zuordnet und
- $c_u: P \rightarrow \mathbb{N}_0$ eine Zuordnung ist, die jedem Platz $p_i \in P$ eine **maximale Kapazität** (upper capacity) $c_u(p_i)$ zuordnet,

Eine zulässige Anfangsmarkierung m_0 muss die folgenden Bedingungen erfüllen

$$c_l(p_i) \leq m_0(p_i) \leq c_u(p_i) \quad \forall \, p_i \in P.$$

Eine Transition in einem Petri-Netz mit Kapazitäten darf nur feuern, wenn dadurch die unteren $(c_l(p_i))$ und oberen Tokenschranken $(c_u(p_i))$ der Input- und Outputplätze nicht verletzt werden. In Abbildung 2.12 ist ein Feuern von $T1$ möglich, da nach Abzug des

Kantengewichts die untere Tokengrenze von $P1$ nicht unterschritten werden würde. Im Gegensatz dazu ist ein Feuern von $T2$ nicht möglich, da $P3$ schon an der unteren Tokengrenze ist und somit kein Token mehr entfernt werden darf.

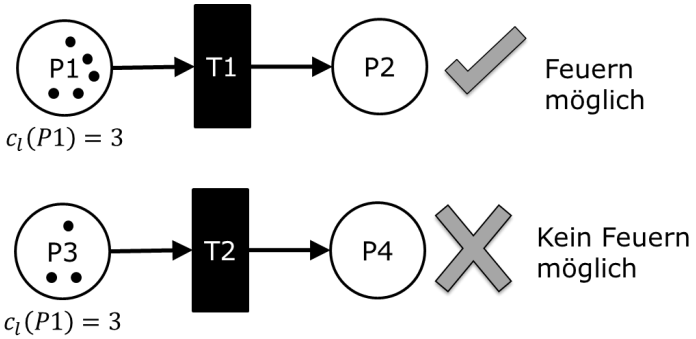


Abbildung 2.13: Beispiele für Petri-Netze mit Kapazitäten: T1 kann feuern; T2 ist nicht aktiv

Definition 2.15

Sei cPN ein Petri-Netz mit Kapazitäten. Eine Transition $t_j \in T$ ist **aktiv** genau dann, wenn

$$\forall\, p_i \in P_{in}(t_j)\colon\ m(p_i) - f(p_i \to t_j) \geq c_l(p_i)$$

und

$$\forall\, p_i \in P_{out}(t_j)\colon\ m(p_i) + f(t_j \to p_i) \leq c_u(p_i),$$

wobei $m(p_i)$ die Markierung von Platz p_i ist.

Ein Platz in einem Petri-Netz mit Kapazitäten kann zwei unterschiedliche Arten von Konflikten haben:

- **Genereller Outputkonflikt**: Ein Platz hat aufgrund seiner minimalen Kapazität nicht genug Token, um alle aktiven Outputtransitionen freizuschalten.
- **Genereller Inputkonflikt**: Ein Platz kann aufgrund seiner maximalen Kapazität nicht alle Token der aktiven Inputtransitionen aufnehmen.

Definition 2.16

Sei cPN ein Petri-Netz mit Kapazitäten. Ein Platz $p_i \in P$ hat einen **generellen Outputkonflikt**, wenn

$$m(p_i) - \sum_{t_j \in TA_{out}(p_i)} f(p_i \to t_j) < c_l(p_i).$$

Ein Platz $p_i \in P$ hat einen **generellen Inputkonflikt**, wenn

$$m(p_i) + \sum_{t_j \in TA_{in}(p_i)} f(t_j \to p_i) > c_u(p_i),$$

wobei $m(p_i)$ die Markierung von Platz p_i ist.

Beide Konflikte können lokal oder global mit den in Kapitel 2.2 vorgestellten Methoden gelöst werden. Bei der **lokalen Konfliktlösung** werden nicht nur den Kanten von den Plätzen zu den Transitionen Bewertungen zugeordnet, sondern auch den Kanten von den Transitionen zu den Plätzen, um einen Inputkonflikt lösen zu können. Zunächst schalten die Plätze ihre Outputtransitionen frei, d.h. die Outputkonflikte werden gelöst. Anschließend werden die Inputtransitionen freigeschaltet. Hierbei stehen aber nur noch diejenigen zur Verfügung, die bereits von allen ihren Inputplätzen freigeschaltet worden sind

$$TAe_{in}(p_i) = \left\{ t_j \in T \mid t_j \in TA_{in}(p_i) \ \wedge \left(t_j \in TE_{out}(p_k) \ \ \forall\, p_k \in P_{in}(t_j) \right) \right\}$$

(siehe Algorithmus 2.6). Demnach ist eine Transition feuerbar, wenn sie von allen Outputplätzen freigeschaltet wurde, da diese die Freischaltung der Inputplätze voraussetzt.

Definition 2.17

Sei cPN ein Petri-Netz mit Kapazitäten. Eine aktive Transition $t_j \in T$ ist **feuerbar** genau dann, wenn

$$\forall\, p_i \in P_{out}(t_j):\ t_j \in TE_{in}(p_i),$$

wobei die Menge $TE_{in}(p_i)$ die freigeschalteten (enabled) Inputtransitionen von Platz p_i enthält.

Zur Lösung der Outputkonflikte können Algorithmus 2.2 und Algorithmus 2.3 verwendet werden. Es muss lediglich die Bedingung

$$t \in TA_{out}(p_i) \wedge (m - f(p_i \to t) \geq 0)$$

durch

$$t \in TA_{out}(p_i) \wedge \left(m - f(p_i \to t) \geq c_l(p_i)\right)$$

ersetzt werden. Für die Inputkonfliktlösung muss u.a. diese Bedingung durch

$$t \in TAe_{in}(p_i) \wedge \left(m + f(t \to p_i) \leq c_u(p_i)\right)$$

ersetzt werden. Zusätzlich müssen weitere Anpassungen gemacht werden, die am Beispiel der Konfliktlösung durch Nutzen und Nutzenquotienten in Algorithmus 2.7 dargestellt werden und für die Konfliktlösung mithilfe von Wahrscheinlichkeiten entsprechend übernommen werden können.

Algorithmus 2.6: Lokaler Freischaltungsprozess für Petri-Netze mit Kapazitäten

Für: i von 1 bis n_p

Wenn:

$$m(p_i) - \sum_{t_j \in TA_{out}(p_i)} f(p_i \rightarrow t_j) \geq c_l(p_i)$$

Ja	Nein
$TE_{out}(p_i) := TA_{out}(p_i)$	Outputkonflikt lösen $TE_{out}(p_i) \subseteq TA_{out}(p_i)$

$$TAe_{in}(p_i) := \left\{ t_j \in T \mid t_j \in TA_{in}(p_i) \wedge \left(t_j \in TE_{out}(p_k) \quad \forall\, p_k \in P_{in}(t_j) \right) \right\}, \quad i = 1, \ldots, n_p$$

Für: i von 1 bis n_p

Wenn:

$$m(p_i) + \sum_{t_j \in TAe_{in}(p_i)} f(t_j \rightarrow p_i) \leq c_u(p_i)$$

Ja	Nein
$TE_{in}(p_i) := TAe_{in}(p_i)$	Inputkonflikt lösen $TE_{in}(p_i) \subseteq TAe_{in}(p_i)$

$$TE := \{ t_j \in T \mid t_j \in TE_{in}(p_i) \quad \forall\, p_i \in P_{out}(t_j) \}$$

Algorithmus 2.7: Lokale Inputkonfliktlösung mithilfe von Nutzen\-quotienten

Für: alle $p_i \in P$ mit $e(p_i) = 2 \vee e(p_i) = 3$

Wenn:

$$m(p_i) + \sum_{t_j \in TAe_{in}(p_i)} f(t_j \to p_i) \leq c_u(p_i)$$

Ja	Nein
$TE_{in}(p_i) \coloneqq TAe_{in}(p_i)$	$m \coloneqq m(p_i)$ $TE_{in}(p_i) \coloneqq \{\ \}$ $TA \coloneqq TAe_{in}(p_i)$ $\tilde{\mathcal{p}}(t_k \to p_i) = \begin{cases} \mathcal{p}(t_k \to p_i) & e(p_i) = 2 \\ \dfrac{\mathcal{p}(t_k \to p_i)}{f(t_k \to p_i)} & e(p_i) = 3 \end{cases}, k = 1, \dots, n_{out}(p_i)$ Für: i von 1 bis $\lvert TAe_{in}(p_i) \rvert$ $\mathcal{p}_{max} \coloneqq \max_{t_k \in TA} \{\tilde{\mathcal{p}}(t_k \to p_i)\}$ $T_{max} \coloneqq \{t \in TA \mid \tilde{\mathcal{p}}(t \to p_i) = \mathcal{p}_{max}\}$ Wähle ein $t \in T_{max}$ $TA \coloneqq TA\ / \{t\}$ Wenn: $m + f(t \to p_i) \leq c_u(p_i)$

Ja	Nein
$TE_{in}(p_i) \coloneqq TE_{in}(p_i) \cup \{t\}$ $m \coloneqq m + f(t \to p_i)$	

In der Abbildung 2.14 sind alle Transitionen aktiv, aber $P2$ hat einen Outputkonflikt und $P3$ hat einen Inputkonflikt. Zunächst schalten $P1$ und $P2$ ihre Outputtransitionen frei. Hierbei wählt $P1$ die Transition $T1$ und $P2$ kann sich für $T2$ oder $T3$ entscheiden. Anschließend kann sich $P3$ für eine der von den Inputplätzen freigeschalteten Transitionen entscheiden.

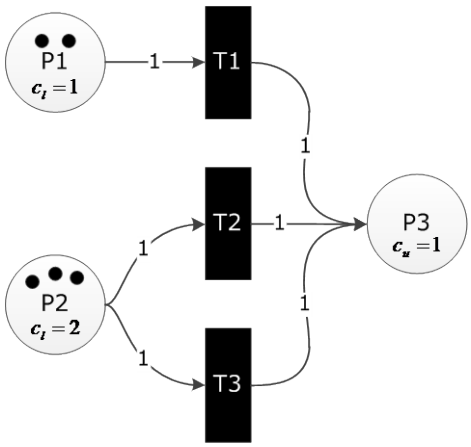


Abbildung 2.14: Petri-Netz mit Konflikten: P1 hat einen Outputkonflikt und P3 hat einen Inputkonflikt

2.4.2 TEST- UND HEMMKANTEN

Das grundlegende Petri-Netz-Konzept kann, wie z.B. in (Matsuno et al. 2003) und (David und Alla 2010) vorgestellt, um **Test-** und **Hemmkanten** erweitert werden. Im Gegensatz zu den Kapazitäten ist dies eine „echte" Erweiterung mit der Eigenschaften eines Systems modelliert werden können, die zuvor nicht abbildbar waren. Diese speziellen Kanten dürfen nur von Plätzen ausgehen und in Transitionen münden. Testkanten werden durch gestrichelte Linien dargestellt und Hemmkanten durch durchgezogene Linien mit Kreisendungen (siehe Abbildung 2.15). Wenn ein Platz über eine Test- oder Hemmkante mit einer Transition verbunden ist, wird die Markierung dieses Platzes während des Feuerprozesses nicht verändert. Die Markierung beeinflusst nur den Aktivierungsprozess. Eine Transition in einem erweiterten Petri-Netz ist **aktiv**, wenn

- alle Inputplätze, die über eine „normale" Kante mit der Transition verbunden sind, mindestens so viele Token haben wie das Kantengewicht,
- alle Inputplätze, die über eine Testkante mit der Transition verbunden sind, mehr Token haben als das Kantengewicht und
- alle Inputplätze, die über eine Hemmkante mit der Transition verbunden sind, weniger Token haben als das Kantengewicht.

In Abbildung 2.15 sind die Plätze $P2$ und $P5$ über eine Testkante mit der Transition $T1$ bzw. $T2$ verbunden. Hierbei ist $T1$ aktiv, da $m(P2) = 3 > f(P2 \rightarrow T1) = 2$. Die Transition $T2$ ist hingegen nicht aktiv, da $m(P5) = 1 \not> f(P5 \rightarrow T2) = 2$.

Die Plätze $P8$ und $P11$ sind über eine Hemmkante mit der Transition $T3$ bzw. $T4$ verbunden. Hierbei ist $T4$ aktiv, da $m(P11) = 1 < f(P11 \rightarrow T4) = 2$ und $T3$ ist nicht aktiv, da $m(P8) = 3 \not< f(P8 \rightarrow T3) = 2$.

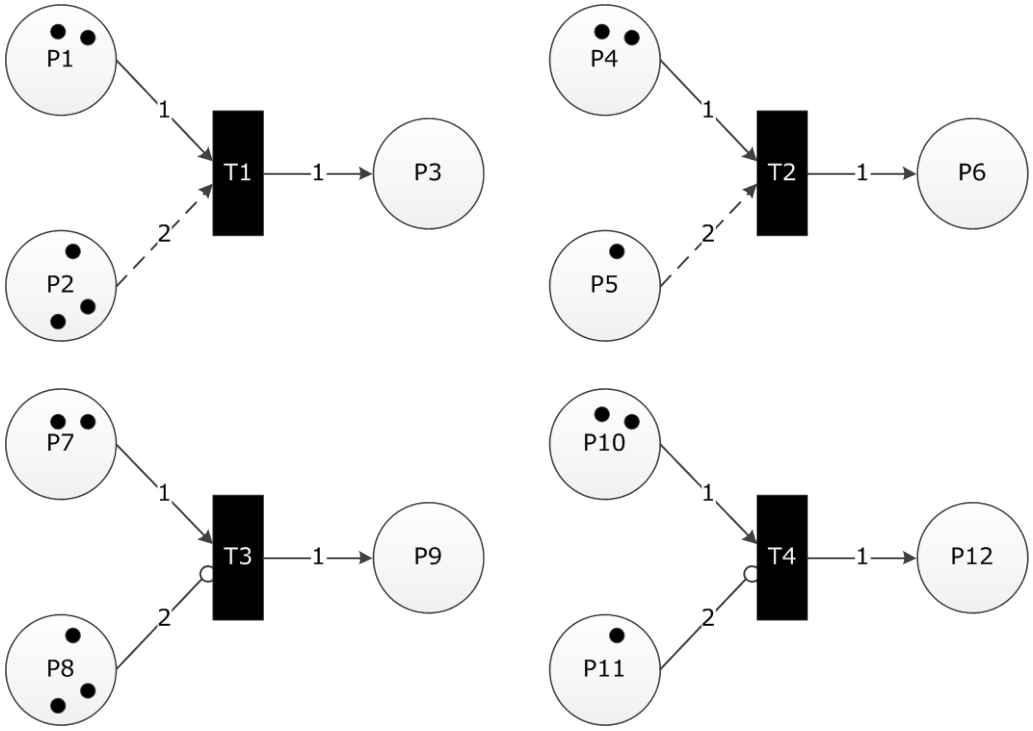


Abbildung 2.15: Erweiterte Petri-Netze mit Testkante (oben) bzw. Hemmkante (unten)

Definition 2.18

Das Tupel $ePN = (PN, \mathcal{T}, \mathcal{I})$ ist ein **erweitertes Petri-Netz**, wenn

- PN ein Petri-Netz nach Definition 2.7, Definition 2.9, Definition 2.10 oder Definition 2.12 ist,
- $\mathcal{T} \subseteq (P \times T)$ eine Menge von Testkanten ist,
- $\mathcal{I} \subseteq (P \times T)$ eine Menge von Hemmkanten ist, wobei $F, \mathcal{T}$ und $\mathcal{I}$ paarweise disjunkt sind. Die Kantengewichtsfunktion f muss modifiziert werden zu $f: (F \cup G \cup \mathcal{T} \cup \mathcal{I}) \rightarrow \{\mathbb{N}: (x \rightarrow y) \in (F \cup G \cup \mathcal{I}),\ \mathbb{N}_0: (x \rightarrow y) \in \mathcal{T}\}$.[1]

Definition 2.19

Sei ePN ein erweitertes Petri-Netz. Eine Transition $t_j \in T$ ist **aktiv**, genau dann wenn

$$\forall\, p_i \in P_{in}(t_j): \begin{cases} m(p_i) \geq f(p_i \rightarrow t_j) & \text{wenn } (p_i \rightarrow t_j) \in F \\ m(p_i) > f(p_i \rightarrow t_j) & \text{wenn } (p_i \rightarrow t_j) \in \mathcal{T} \\ m(p_i) < f(p_i \rightarrow t_j) & \text{wenn } (p_i \rightarrow t_j) \in \mathcal{I}. \end{cases}$$

[1] Für die Testkante muss auch das Kantengewicht null zugelassen werden. Dies bedeutet, dass ein Platz nicht leer sein darf, damit die Transition aktiv werden kann (siehe auch Definition 2.19).

Die Konflikte in einem erweiterten Petri-Netz können mit den in Kapitel 2.2 vorgestellten Methoden gelöst werden. Der Feuerungsprozess wird wie in Kapitel 2.3 beschrieben durchgeführt.

Auch Petri-Netze mit Kapazitäten können um Test- und Hemmkanten erweitert werden. Dann ändert sich der Aktivierungsprozess aus Definition 2.19 leicht.

Definition 2.20

Das Tupel $xPN = (cPN, \mathcal{T}, \mathcal{I})$ ist ein **erweitertes Petri-Netz mit Kapazitäten**, wenn

- cPN ein Petri-Netz mit Kapazitäten nach Definition 2.14 ist,
- $\mathcal{T} \subseteq (P \times T)$ eine Menge von Testkanten ist,
- $\mathcal{I} \subseteq (P \times T)$ eine Menge von Hemmkanten ist, wobei $F, \mathcal{T}$ und $\mathcal{I}$ paarweise disjunkt sind. Die Kantengewichtsfunktion f muss modifiziert werden zu $f: (F \cup G \cup \mathcal{T} \cup \mathcal{I}) \to \{\mathbb{N}: (x \to y) \in (F \cup G \cup \mathcal{I}),\ \mathbb{N}_0: (x \to y) \in \mathcal{T}\}$.

Definition 2.21

Sei xPN ein erweitertes Petri-Netz mit Kapazitäten. Eine Transition $t \in T$ ist **aktiv**, genau dann wenn

$$\forall\, p_i \in P_{in}(t_j): \begin{cases} m(p_i) - f(p_i \to t_j) \geq c_l(p_i) & \text{wenn } (p_i \to t_j) \in F \\ m(p_i) > f(p_i \to t_j) & \text{wenn } (p_i \to t_j) \in \mathcal{T} \\ m(p_i) < f(p_i \to t_j) & \text{wenn } (p_i \to t_j) \in \mathcal{I} \end{cases}$$

und

$$\forall\, p_i \in P_{out}(t_j) :\ m(p_i) + f(p_i \to t_j) \leq c_u(p_i).$$ [1]

Die Input- und Outputkonflikte in einem erweiterten Petri-Netz mit Kapazitäten können mit den in Kapitel 2.4.1 beschriebenen Methoden gelöst werden. Der Feuerungsprozess wird wie in Kapitel 2.3 beschrieben durchgeführt.

Zur Erläuterung des erweiterten Anwendungsbereiches werden nachfolgend zwei Beispiele angeführt. In Abbildung 2.16 sind zwei biologische Reaktionen dargestellt, die mithilfe von erweiterten Petri-Netzen modelliert worden sind. Bei der oberen Reaktion kann der Ausgangsstoff, auch Substrat genannt, nur in das Produkt umgewandelt werden, wenn der Hemmstoff dieser Reaktion eine gewisse Grenze nicht übersteigt. Im Gegensatz dazu muss

[1] Es sei bemerkt, dass die Kapazitätsgrenzen nur bei den „normalen“ Kanten wirken, da durch Test- und Hemmkanten keine Token entfernt werden.

bei der unteren Reaktion von dem aktivierenden Stoff eine bestimmte Menge vorhanden sein, damit die Reaktion überhaupt ablaufen kann.

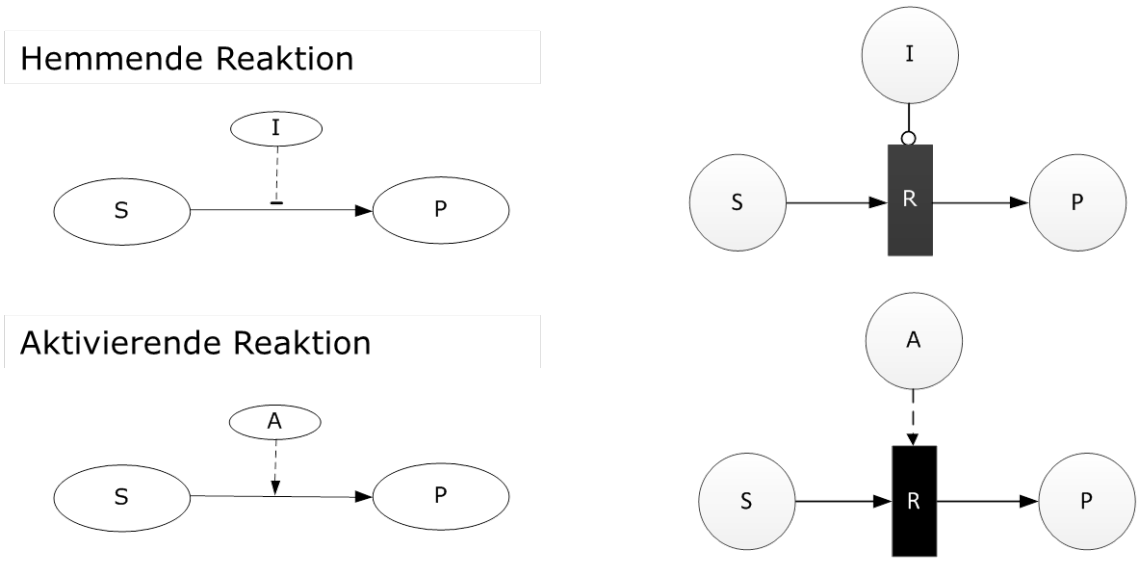


Abbildung 2.16: Modellierung von biologischen Reaktionen mithilfe von erweiterten Petri-Netzen

In Abbildung 2.17 wird der folgende Sachverhalt mithilfe von einem erweiterten Petri-Netz dargestellt: Ein Verkäufer lässt Kunden in das Geschäft und schließt dann die Eingangstür bevor er mit seiner Arbeit beginnt. Wenn die Kunden bedient wurden, verlassen Sie das Geschäft durch eine andere Tür. Die Eingangstür wird erst wieder geöffnet, wenn alle Kunden, die reingekommen sind, das Geschäft verlassen haben.

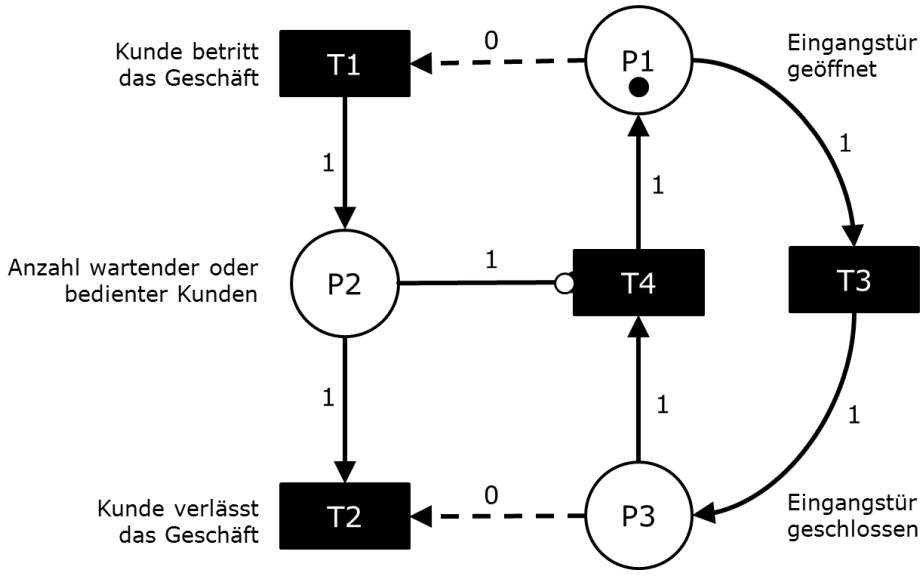


Abbildung 2.17: Modellierung eines Geschäftseingangs mithilfe von erweiterten Petri-Netzen

2.4.3 FUNKTIONEN

Bei funktionalen Petri-Netzen sind die Kantengewichte Funktionen, die von den Markierungen der Plätze abhängen. Dieses Konzept wurde 1998 von Hofestädt und Thelen vorgestellt (Hofestädt und Thelen 1998). Abbildung 2.18 zeigt das Petri-Netz einer biochemischen Reaktion, bei der Substrate (S) mithilfe von Enzymen (E) in Produkte (P) umgewandelt werden. Wie viele Substrate pro Schritt in Produkte umgewandelt werden, hängt von der vorhandenen Anzahl Enzyme ab. Bei jeder Feuerung der Transition R werden $n_1 \cdot m(E)$ Token von Platz S entfernt und $m(E)$ von Platz E. Außerdem werden $n_2 \cdot m(E)$ Token zu Platz P hinzugefügt und $m(E)$ zu Platz E. Die Transition R ist solange aktiv wie E nicht leer ist und S mindestens $n_1 \cdot m(E)$ Token hat.

Definition 2.22

Das Tupel $fPN = (PN)$ ist ein funktionales Petri-Netz, wenn PN ein Petri-Netz nach Definition 2.7, Definition 2.9, Definition 2.10 oder Definition 2.12 ist und die Kantengewichtsfunktion modifiziert wird zu einer dynamischen Kantengewichtsfunktion $f: (F \cup G, m) \rightarrow \mathbb{N}_0$, die jeder Kante ein Gewicht zuweist, das von den Markierungen der Plätze abhängen kann.[1]

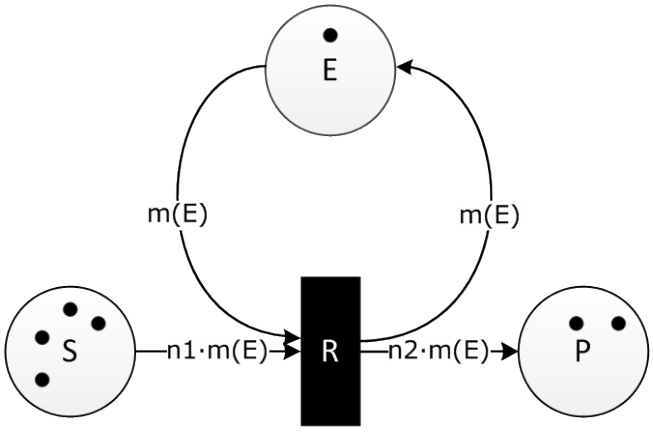


Abbildung 2.18: Funktionales Petri-Netz, bei dem die Kantengewichte Funktionen der Markierungen sind

Der Aktivierungsprozess aus Kapitel 2.1 kann übernommen werden sowie die Konfliktlösungen aus Kapitel 2.2 und der Feuerungsprozess aus Kapitel 2.3. Funktionale

[1] Für funktionale Petri-Netze muss auch das Kantengewicht null zugelassen werden, da eine Markierung auch null werden kann und die Kantengewichte funktional von den Markierungen der Plätze abhängen. Werden Konflikte mithilfe von Nutzenquotienten gelöst, müssen entsprechende Vorkehrungen getroffen werden, damit es nicht zu einer Division durch null kommt.

Petri-Netze können auch um Kapazitäten sowie Test- und Hemmkanten erweitert werden. Dann behalten die entsprechenden Definitionen aus Kapitel 2.4.1 bzw. Kapitel 2.4.2 ihre Gültigkeit.

2.5 MATRIZENDARSTELLUNG VON PETRI-NETZEN

Jedes Petri-Netz kann auch mithilfe von Matrizen und Vektoren spezifiziert werden. Dadurch wird die Berechnung der neuen Markierung des Petri-Netzes nach Feuerung von Transitionen auf eine Matrix-Vektor-Multiplikation zurückgeführt. Hierbei werden die Kanten und deren Gewichte in einer Matrix zusammengefasst. Die Zeilen stehen bei dieser Matrix für die einzelnen Plätze und die Spalten für die Transitionen, d.h. es handelt sich um eine Matrix mit der Dimension $(n_p \times n_t)$:

$$\mathcal{F} = \begin{pmatrix} f_{1,1} & \cdots & f_{1,n_t} \\ \vdots & \ddots & \vdots \\ f_{n_p,1} & \cdots & f_{n_p,n_t} \end{pmatrix} \quad \text{mit} \quad f_{i,j} = \begin{cases} f(p_i \to t_j), & (p_i \to t_j) \in F \\ 0, & (p_i \to t_j) \notin F \end{cases}$$

$$\mathcal{G} = \begin{pmatrix} g_{1,1} & \cdots & g_{1,n_t} \\ \vdots & \ddots & \vdots \\ g_{n_p,1} & \cdots & g_{n_p,n_t} \end{pmatrix} \quad \text{mit} \quad g_{i,j} = \begin{cases} f(t_j \to p_i), & (t_j \to p_i) \in G \\ 0, & (t_j \to p_i) \notin G \end{cases}$$

$$\mathcal{H} = \mathcal{G} - \mathcal{F}$$

Die Matrix $\mathcal{F}$ beinhaltet die Gewichte der Kanten von den Plätzen zu den Transitionen. Besteht zwischen einen Platz p und einer Transition t keine Verbindung $\left((p \to t) \notin F\right)$, dann wird das zugehörige Matrixelement auf null gesetzt. Bei der Matrix $\mathcal{G}$, die die Kantengewichte von den Transitionen zu den Plätzen enthält, wird genauso verfahren. Zudem gibt es noch eine Matrix $\mathcal{H}$, die sich aus der Differenz von $\mathcal{G}$ und $\mathcal{F}$ ergibt. In gleicher Weise können die Kantenbewertungen zur lokalen Konfliktlösung in einem kantenbewerteten Petri-Netz sowie Test- und Hemmkanten in einem erweiterten Petri-Netz durch eine Matrix dargestellt werden.

Die Markierungen der Plätze eines Petri-Netzes können in einem Spaltenvektor zusammengefasst werden:[1]

$$\boldsymbol{m} = \left(m(p_1), m(p_2), \ldots, m\left(p_{n_p}\right)\right)^{\mathsf{T}} = \left(m_1, m_2, \ldots, m_{n_p}\right)^{\mathsf{T}}.$$

[1] Bezeichnungen für Vektoren werden im Folgenden fett dargestellt.

Ebenso können die minimalen und maximalen Kapazitäten und die Bewertungen der Transitionen bei der globalen Konfliktlösung jeweils in einem Vektor zusammengefasst werden.

Ob eine Transition feuert oder nicht, kann mit einem Spaltenvektor bestehend aus Nullen und Einsen dargestellt werden:

$$\boldsymbol{f} = \left(x_1, x_2, \dots, x_{n_t}\right)^\top \quad \text{mit} \quad x_j = \begin{cases} 0, & t_j \notin TF \\ 1, & t_j \in TF \end{cases}$$

Hierbei bedeutet $x_j = 1$, dass die Transition t_j feuert und $x_j = 0$, dass sie nicht feuert. Analog können die Vektoren für die Menge der aktiven Transitionen und die Menge der freigeschalteten Transitionen aufgestellt werden.

Die Aktivierung einer Transition $t_j \in T$ in einem Petri-Netz mit Kapazitäten (siehe Definition 2.15) könnte mit folgenden Bedingungen, die komponentenweise zu verstehen sind, geprüft werden

$$\boldsymbol{m} - \mathcal{F} \cdot \boldsymbol{e_j} \geq \boldsymbol{c_l} \quad \text{und} \quad \boldsymbol{m} + \mathcal{G} \cdot \boldsymbol{e_j} \leq \boldsymbol{c_u},$$

hierbei sind $\boldsymbol{c_l}$ und $\boldsymbol{c_u}$ die Vektoren mit den minimalen bzw. maximalen Kapazitäten und $\boldsymbol{e_j}$ ist der $j$te-Einheitsvektor.

Mit den Bedingungen

$$\boldsymbol{m} - \mathcal{F} \cdot \boldsymbol{a} \geq \boldsymbol{c_l} \quad \text{bzw.} \quad \boldsymbol{m} + \mathcal{G} \cdot \boldsymbol{a} \leq \boldsymbol{c_u}$$

kann ein Output- bzw. Inputkonflikt festgestellt werden, wobei $\boldsymbol{a}$ ein Vektor ist, der angibt, ob eine Transition aktiv ist oder nicht.

Die neue Markierung kann mit folgender Matrix-Vektor-Multiplikation berechnet werden

$$\boldsymbol{m_{neu}} = \boldsymbol{m} + \mathcal{H} \cdot \boldsymbol{f}.$$

Die Matrixdarstellung spielt eine wesentliche Rolle bei der Umsetzung des Petri-Netz-Formalismus mithilfe von MATLAB (siehe Kapitel 4). Zudem erleichtert sie die Formulierung der Optimierungsalgorithmen in Kapitel 3.

Das Beispiel in Abbildung 2.19 soll die Darstellung von Petri-Netzen mithilfe von Matrizen und Vektoren verdeutlichen. Es ergeben sich folgende Matrizen für die Kantengewichte und die Anfangsmarkierung:

$$\mathcal{F} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \quad \mathcal{G} = \begin{pmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 5 \\ 5 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathcal{H} = \begin{pmatrix} -2 & 0 & 3 & 0 \\ -3 & -1 & 0 & 5 \\ 5 & 0 & -2 & 0 \\ 1 & 2 & 0 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\boldsymbol{m}_0 = (7 \quad 10 \quad 4 \quad 1 \quad 0)^{\mathsf{T}}.$$

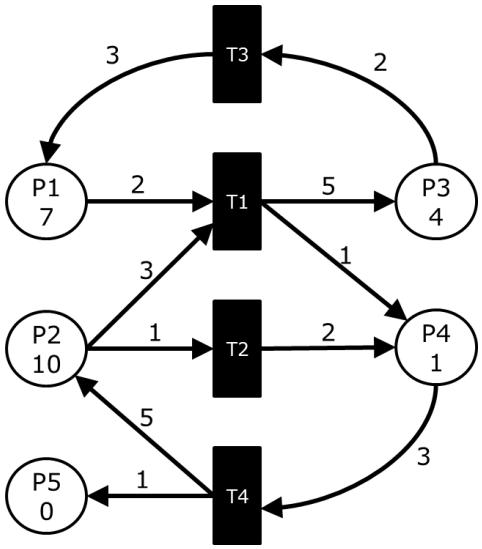


Abbildung 2.19: Beispiel zur Darstellung eines Petri-Netzes mithilfe von Matrizen

Mit der folgenden Bedingung kann überprüft werden, ob die Transition $T1$ aktiv ist

$$\boldsymbol{m} - F \cdot \boldsymbol{e_1} \geq \boldsymbol{0}$$

$$\begin{pmatrix} 7 \\ 10 \\ 4 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} 7 \\ 10 \\ 4 \\ 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 \\ 3 \\ 0 \\ 0 \\ 0 \end{pmatrix} \geq \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Da $T1$ aktiv ist, kann sie gefeuert werden und die neue Markierung des Petri-Netzes kann mithilfe der folgenden Gleichung berechnet werden

$$\boldsymbol{m_{neu}} = \boldsymbol{m} + \mathcal{H} \cdot \boldsymbol{f}$$

3 OPTIMALE FEUERUNGSREIHENFOLGE (OFR)

Bei der Optimierung eines Petri-Netzes soll eine zu einer vorgegeben Zielsetzung optimale Feuerungsreihenfolge (OFR) der Transitionen gefunden werden. Hierbei sind verschiedene Zielsetzungen denkbar. Beispielsweise könnte man daran interessiert sein, die Feuerungsreihenfolge zu finden, die die Anzahl Token in einem oder in mehreren Plätzen nach einer vorgegebenen Anzahl Schritten maximiert. Eine andere Zielsetzung könnte sein, die Feuerungsreihenfolge mit minimalen Kosten zu ermitteln, die gleichzeitig nach einer vorgegeben Anzahl Schritten eine bestimmte Zielmarkierung erreicht. Die Grundlage für die Petri-Netz-Optimierung bilden Petri-Netze mit Kapazitäten (Definition 2.14).

Bei allen Zielsetzungen der Petri-Netz-Optimierung muss über Nebenbedingungen gewährleistet werden, dass die Kapazitäten nicht verletzt werden. Die Nebenbedingungen

$$m_k(p_i) - \sum_{t_j \in T_{out}(p_i)} f(p_i \to t_j) \cdot x_{k,j} \geq c_l(p_i) \quad \forall p_i \in P \quad \forall k \in \{1, \dots, n_s\} \qquad (NBl)$$

verhindern, dass die minimalen Kapazitäten der Plätze unterschritten werden und die Nebenbedingungen

$$m_k(p_i) + \sum_{t_j \in T_{in}(p_i)} f(t_j \to p_i) \cdot x_{k,j} \leq c_u(p_i) \quad \forall p_i \in P \quad \forall k \in \{1, \dots, n_s\} \qquad (NBu)$$

verhindern die Überschreitung der maximalen Kapazitäten. Hierbei ist n_s die Anzahl Schritte, $m_k(p_i)$ ist die Markierung von Platz $p_i \in P$ im Schritt k und $c_l(p_i)$ und $c_u(p_i)$ ist die untere bzw. obere Kapazität von Platz p_i.

Zusätzlich sind für die **Feuerungsvariablen** $x_{k,j},\ k = 1, \dots, n_s,\ j = 1, \dots, n_t$ nur binäre Werte zulässig

$$x_{k,j} \in \{0,1\} \quad k = 1, \dots, n_s, \qquad j = 1, \dots, n_t \qquad (NBb)$$

Wenn $x_{k,j} = 1$, feuert die Transition t_j in Schritt k, und wenn $x_{k,j} = 0$, feuert sie nicht.

Zudem kann die Feuerungsstrategie über Nebenbedingungen gesteuert werden. Hierbei wird zwischen parallelem und seriellem Feuern unterschieden. Bei der **parallelen Feuerung** können in einem Schritt mehrere Transitionen parallel, also gleichzeitig, feuern. Hierfür sind keine weiteren Nebenbedingungen notwendig. Beim **seriellen Feuern** hingegen darf pro

$$= \begin{pmatrix} 7 \\ 10 \\ 4 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -2 & 0 & 3 & 0 \\ -3 & -1 & 0 & 5 \\ 5 & 0 & -2 & 0 \\ 1 & 2 & 0 & -3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 7 \\ 10 \\ 4 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -2 \\ -3 \\ 5 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 7 \\ 9 \\ 2 \\ 0 \end{pmatrix}.$$

Schritt nur eine Transition feuern. Dieses Verhalten kann über die folgenden Nebenbedingungen geregelt werden

$$\sum_{j=1}^{n_t} x_{k,j} = 1 \quad k = 1, \dots, n_s \tag{NBs}$$

Darüber hinaus kann über weitere Nebenbedingungen erreicht werden, dass eine bestimmte **Zielmarkierung** $\boldsymbol{m_z} = \left(m_z(p_1), \dots, m_z\left(p_{n_p}\right)\right)$ nach n_s Schritten mindestens erreicht wird

$$m_{n_s}(p_i) \geq m_z(p_i), \qquad i = 1, \dots, n_p$$

$$m_{n_s}(p_i) = m_0(p_i) + \sum_{k=1}^{n_s} \left(\sum_{t_j \in T_{in}(p_i)} f(t_j \to p_i) \cdot x_{k,j} - \sum_{t_j \in T_{out}(p_i)} f(p_i \to t_j) \cdot x_{k,j} \right) \tag{NBz}$$

Nachfolgend werden zwei mögliche Zielsetzungen der Petri-Netz-Optimierung vorgestellt. Hierbei handelt es sich immer um **binäre Optimierungsprobleme**, da für die Feuerungsvariablen x_{kj} nur die Werte 0 und 1 zugelassen sind.

3.1 Maximaler Nutzen

Beim diesem Optimierungsproblem ist die Zielsetzung, dass diejenige Feuerungsreihenfolge gefunden werden soll, die nach einer vorgegebenen Anzahl Schritte den **maximalen Nutzen** liefert. Die Grundlage hierzu bildet ein **global transitionenbewertetes Petri-Netz mit Kapazitäten** (siehe Definition 2.10 und Definition 2.14), bei dem jeder Transition ein Nutzen für dessen Feuerung zugeordnet wird. Hieraus ergibt sich die folgende Zielfunktion für die Optimierung

$$z = \max\left(\sum_{k=1}^{n_s} \sum_{j=1}^{n_t} w(t_j) \cdot x_{k,j} \right).$$

Hierbei ist n_s die Anzahl Schritte, n_t die Anzahl der Transitionen des Petri-Netzes, $w(t_j) \in \mathbb{R}_{\geq 0}$ der Nutzen der Transition $t_j \in T$ und $x_{k,j},\ k = 1, \dots, n_s,\ j = 1, \dots, n_t$ die Feuerungsvariablen.

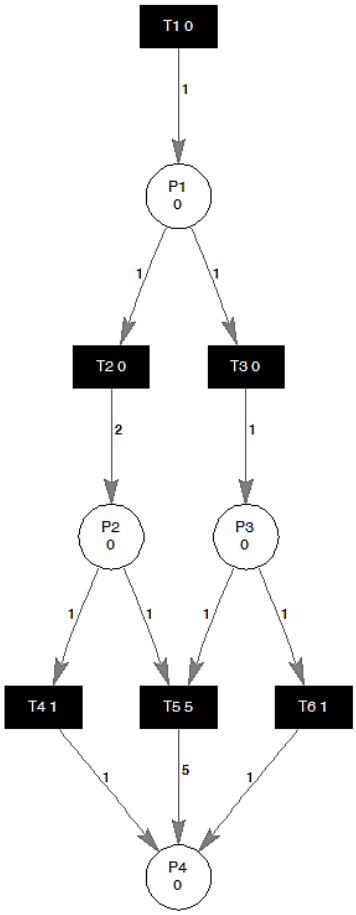


Abbildung 3.1: Optimale Feuerungsreihenfolge: Maximierung der Token in P4

Dieses Optimierungsproblem kann auch dazu genutzt werden die Tokenanzahl in einem oder in mehreren Plätzen zu maximieren. Dies soll beispielhaft am Petri-Netz in Abbildung 3.1 gezeigt werden. In diesem Petri-Netz soll die Anzahl Token in $P4$ maximiert werden, wobei $P2$ und $P3$ je nach Markierung einen Konflikt haben können. Es soll nun diejenige Feuerungsreihenfolge ermittelt werden, die nach 10 Schritten die maximale Anzahl Token in $P4$ liefert. Hierbei erhalten die Transitionen $T1$, $T2$ und $T3$ keinen Nutzen ($\omega_1 = \omega_2 = \omega_3 = 0$), da sie nicht direkt zur Vermehrung der Token in $P4$ beitragen. Die Transitionen $T4$, $T5$ und $T6$ erhalten ihr Kantengewicht zu $P4$ als Nutzen ($\omega_4 = 1,\ \omega_5 = 5,\ \omega_6 = 1$), da diese Token pro Feuerungsschritt zu $P4$ hinzugefügt werden. Zusätzlich können pro Schritt mehrere Transitionen parallel feuern. Hieraus ergibt sich die folgende Zielfunktion:

$$z = \max\left(\sum_{k=1}^{10}\left(x_{k,4} + 5x_{k,5} + x_{k,6}\right)\right)$$

Da es keine Kapazitäten für die Plätze gibt, reicht es aus über die Nebenbedingung (NBl) mit $c_l(p_i) = 0,\ i = 1,..,4$ sicherzustellen, dass in keinem Schritt durch Feuerung der Transitionen negative Token auftreten

$$m_k(p_1) - x_{k,2} - x_{k,3} \geq 0$$
$$m_k(p_2) - x_{k,4} - x_{k,5} \geq 0$$
$$m_k(p_3) - x_{k,5} - x_{k,6} \geq 0$$
$$\forall k \in \{1, \dots, 10\}$$

Dieses Optimierungsproblem stellt eine globale Konfliktlösung über mehrere Schritte dar. Im Gegensatz zur globalen Konfliktlösung aus Kapitel 2.2.2, wird hier nicht der Nutzen pro Schritt maximiert, sondern der Gesamtnutzen nach einer vorgegebenen Anzahl Schritte. Auch dieses Optimierungsproblem kann als MKP formuliert werden:

Aus einer Menge von aktiven Transitionen soll in jedem Schritt eine Teilmenge freigeschaltet werden. Jede dieser Transitionen hat Input- und Output-Kantengewichte sowie einen Nutzen für dessen Feuerung. Nun soll der Gesamtnutzen der freigeschalteten Transitionen in n_s Schritten maximiert werden, wobei deren Kantengewichtsummen die Kapazitäten der Plätze in keinem Schritt verletzen dürfen.

3.2 Minimale Kosten

Bei diesem Optimierungsproblem soll eine vorgegebene Zielmarkierung $\boldsymbol{m_z}$ mindestens erreicht werden bei gleichzeitiger Kostenminimierung. Die Grundlage dazu bildet wieder ein **global transitionenbewertetes Petri-Netz mit Kapazitäten** (siehe Definition 2.10 und Definition 2.14), bei dem jeder Transition im Gegensatz zum vorherigen Optimierungsproblem kein Nutzen sondern Kosten für dessen Feuerung zugeordnet wird. Hieraus ergibt sich die folgende Zielfunktion für die Optimierung

$$z = \min\left(\sum_{k=1}^{n_s}\sum_{j=1}^{n_t} w(t_j) \cdot x_{k,j}\right),$$

wobei $\omega(t_j) \in \mathbb{R}_{\geq 0}$ die Kosten der Feuerung der Transition $t_j \in T$ darstellen. Das gleichzeitig die vorgegebene Zielmarkierung $\boldsymbol{m_z}$ mindestens erreicht wird, wird über die Nebenbedingung (NBz) in das Optimierungsproblem integriert.

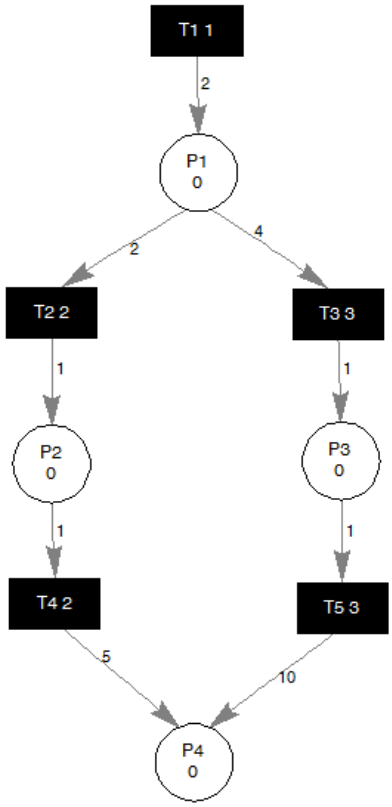


Abbildung 3.2: Optimale Feuerungsreihenfolge: Minimierung der Kosten bei gleichzeitiger Erreichung der Zielmarkierung

In dem Petri-Netz in Abbildung 3.2 soll nach 15 Schritten mindestens die Zielmarkierung $m_z = (0,0,0,11)$ erreicht werden, d.h. in $P4$ sollen sich nach 15 Schritten mindestens 11 Token befinden. Gleichzeitig sollen die Kosten der Feuerungen minimiert werden. Die Kosten der Transitionen betragen $\omega(T1) = 1, \omega(T2) = 2, \omega(T3) = 3, \omega(T4) = 2, \omega(T5) = 3$. Hieraus ergibt sich die Zielfunktion

$$z = \min\left(\sum_{k=1}^{15}\left(x_{k,1} + 2x_{k,2} + 3x_{k,3} + 2x_{k,4} + 3x_{k,5}\right)\right).$$

Die Erreichung von der Mindestmarkierung von 11 Token in $P4$ wird durch die Nebenbedingung (NBz)

$$m_{15}(P4) = \sum_{k=1}^{15}\left(5x_{k,4} + 10x_{k.5}\right) \geq m_z(P4) = 11$$

erreicht. Da es keine Kapazitäten für die Plätze gibt, reicht es aus über die Nebenbedingung (NBl) mit $c_l(p_i) = 0, i = 1,..,4$ sicherzustellen, dass in keinem Schritt durch die Feuerung der Transitionen negative Token auftreten

$$m_k(p_1) - 2x_{k,2} - 4x_{k,3} \geq 0$$
$$m_k(p_2) - x_{k,4} \geq 0$$
$$m_k(p_3) - x_{k,5} \geq 0$$
$$\forall k \in \{1, \dots, 15\}$$

Zusätzlich sollen die Transitionen seriell gefeuert werden. Dies wird über die Nebenbedingungen (NBs)

$$\sum_{j=1}^{5} x_{k,j} = 1 \quad k = 1, \dots, 15$$

sichergestellt.

Auch dieses Optimierungsproblem stellt eine globale Konfliktlösung über mehrere Schritte dar, dass als MKP formuliert werden kann:

Aus einer Menge von aktiven Transitionen soll in jedem Schritt eine Teilmenge freigeschaltet werden. Jede dieser Transitionen hat Input- und Output-Kantengewichte sowie Kosten für dessen Feuerung. Nun sollen die Gesamtkosten der freigeschalteten Transitionen in n_s Schritten minimiert werden bei gleichzeitiger Erfüllung einer Zielmarkierung. Zusätzlich dürfen die Kantengewichtsummen die Kapazitäten der Plätze in keinem Schritt verletzen.

3.3 VERFAHREN ZU ERMITTLUNG DER OFR

Um die optimale Feuerungsreihenfolge (OFR) zu ermitteln, muss ein binäres Optimierungsproblem gelöst werden. Dieses Optimierungsproblem lässt sich, wie zuvor gezeigt, als MKP formulieren (vgl. Kapitel 2.2). MKP sind bekanntermaßen **NP-schwere** Probleme, da die Anzahl möglicher Lösungen exponentiell mit der Anzahl Variablen $l = n_s \cdot n_t$ steigt. Für die Lösung des binären Optimierungsproblems gibt es 2^l mögliche Kombinationen der Variablen.

Zur Lösung dieses Optimierungsproblems stehen in der Literatur zwei Klassen von Verfahren zur Verfügung: Exakte Verfahren und Näherungsverfahren. **Exakte Verfahren** garantieren, dass die optimale Lösung gefunden wird. Dennoch wird dafür bei NP-schweren Problemen im schlimmsten Fall exponentielle Rechenzeit benötigt. Wenn die optimale Lösung nicht

effizient ermittelt werden kann, besteht die einzige Möglichkeit darin Optimalität durch Effizienz einzutauschen, d.h. man begnügt sich mit einer sehr guten Lösung, die in polynomieller Rechenzeit erreicht werden kann. **Näherungsverfahren**, oft auch **Heuristiken** genannt, liefern eine gute Lösung bei relativ geringem Rechenaufwand. Dennoch sind sie nicht in der Lage die Optimalität der Lösung zu garantieren.

In diesem Kapitel sollen zunächst ganz allgemein Verfahren vorgestellt werden, die prinzipiell für die Ermittlung der OFR eingesetzt werden können. Die notwendigen spezifischen Anpassungen, die aus der Struktur des Optimierungsproblems herrühren, werden im anschließenden Kapitel diskutiert.

3.3.1 Exakte Verfahren

Zur exakten Auffindung der OFR kann ein Branch-and-Bound-Ansatz verwendet werden. Die Grundidee von Branch-and-Bound-Verfahren besteht darin, den gesamten zulässigen Lösungsbereich X_0 des Ausgangsproblems P_0 systematisch in Teilprobleme P_l mit disjunkten Teillösungsbereichen X_l aufzuspalten (**Branching**). Durch rekursives Aufspalten in Teilbereiche entsteht eine Baumstruktur (siehe Abbildung 3.3), wobei jedes Teilproblem durch einen Knoten dargestellt wird (vgl. Domschke und Drexl 2005).[1]

Jeder Teilbereich wird mithilfe von Zielfunktionsschranken (**Bounds**) auf dessen Relevanz für weitere Aufspaltungen untersucht. Auf eine weitere Betrachtung eines Teilbereiches kann verzichtet werden, wenn der theoretisch bestmögliche Zielfunktionswert schlechter ist als der beste Zielfunktionswert zu einer bereits bekannten zulässigen Lösung. Es lässt sich stets eine **untere Schranke** $\underline{z}$ für den Zielfunktionswert angeben. Dies ist die aktuell beste zulässige Lösung. Zu Beginn des Verfahrens kann $\underline{z}$ beispielsweise durch eine Heuristik ermittelt werden. Die obere Schranke $\overline{z_l}$ lässt sich für jeden Teilbereich X_l berechnen, indem man eine Relaxation $\tilde{P}_l$ des Teilproblems P_l löst. Ein Problem P_l wird nicht weiterbetrachtet, d.h. es wird nicht aufgespaltet, wenn

- **Fall 1**: $\overline{z_l} \leq \underline{z}$: Die Lösung des Teilproblems kann nicht besser sein, als die aktuell beste zulässige Lösung.

[1] Bei der folgenden Beschreibung des Verfahrens wird von einem Maximierungsproblem ausgegangen.

- **Fall 2**: $\overline{z_l} > \underline{z}$ und die optimale Lösung von $\tilde{P}_l$ ist eine zulässige ganzzahlige Lösung: Eine neue beste Lösung wurde gefunden und die untere Grenze wird entsprechend neu gesetzt $\underline{z} := \overline{z_l}$.
- **Fall 3**: $\tilde{X}_l = \emptyset$: $\tilde{P}_l$ besitzt keine zulässige Lösung und deshalb besitzt auch P_l keine zulässige Lösung (vgl. Domschke und Drexl 2005).

Algorithmus 3.1: Allgemeiner Ablauf des Branch-and-Bound-Verfahrens (vgl. Domschke und Drexl 2005)

<table>
<tr><td colspan="3">Wenn: Zulässige Lösung vorhanden?</td></tr>
<tr><td colspan="2">Ja</td><td>Nein</td></tr>
<tr><td colspan="2"></td><td>(1) zulässige Lösung und $\underline{z}$ z.B. durch Heuristik bestimmen</td></tr>
<tr><td colspan="3">$P_l := P_0$</td></tr>
<tr><td colspan="3">(2) Löse LP-Relaxation $\tilde{P}_l$</td></tr>
<tr><td colspan="3">Wenn: $\overline{z_l} \leq \underline{z}$ oder $\tilde{X}_l = \emptyset$?</td></tr>
<tr><td colspan="2">Ja</td><td>Nein</td></tr>
<tr><td colspan="2"></td><td>Wenn: $\overline{z_l} > \underline{z}$ und optimale Lösung ist ganzzahlig und zulässig</td></tr>
<tr><td colspan="2"></td><td>Ja | Nein</td></tr>
<tr><td colspan="2"></td><td>(3) $\underline{z} := \overline{z_l}$ ggf. Kandidatenliste bereinigen | (4) P_l zum Aufspalten in Kandidatenliste ablegen</td></tr>
<tr><td colspan="3">Wenn: Kandidatenliste leer?</td></tr>
<tr><td colspan="2">Ja</td><td>Nein</td></tr>
<tr><td colspan="2">STOPP</td><td>(5) Auswahl eines Problems P_k aus der Kandidatenliste
(6) Bilde Teilproblem P_l von P_k; falls P_k vollständig verzweigt ist, P_k aus der Kandidatenliste entfernen;
Gehe zu (2)</td></tr>
</table>

In Algorithmus 3.1 ist der allgemeine Ablauf des Branch-and-Bound-Verfahrens dargestellt. Im Folgenden werden die einzelnen Komponenten des Branch-and-Bound-Verfahrens ausführlicher erläutert. Diese Erläuterungen sind angelehnt an (Mathworks 2012b und Domschke und Drexl 2005).

Relaxation

Die drei wichtigsten Relaxationen sind die LP-Relaxation, die Lagrange-Relaxation und die Surrogate-Relaxation. Zur weiteren Erläuterung wird das Ausgangsproblem, wie folgt, mithilfe von Vektoren und Matrizen definiert

$$P: \quad z = \max(\boldsymbol{cx} \mid A\boldsymbol{x} \leq \boldsymbol{b},\ \ \boldsymbol{x} \in \{0,1\}^l),$$

wobei der Vektor $\boldsymbol{c} \in R_{\geq 0}^{l}$ die Koeffizienten der Zielfunktion enthält, $A \in \mathbb{R}^{n_s \cdot n_p \times l}$ ist die Koeffizientenmatrix der Nebenbedingungen, der Vektor $\boldsymbol{b} \in \mathbb{R}^{n_s \cdot n_p}$ enthält die rechten Seiten der Nebenbedingungen und $\boldsymbol{x} \in \{0,1\}^l$ ist der binäre Vektor (vgl. Kapitel 3.4).

Bei der **LP-Relaxation** wird die Forderung der Ganzzahligkeit aufgegeben. Die Nebenbedingungen (NBb) $\boldsymbol{x} \in \{0,1\}^l$ werden zu $\boldsymbol{x} \in [0;1]^l$ relaxiert. Dadurch lässt sich das Problem mit Methoden der linearen Optimierung (z.B. Simplexalgorithmus) lösen. Es ergibt sich folgendes relaxiertes Problem

$$\tilde{P}_{LP}: \quad z_{LP} = \max(\boldsymbol{cx} \mid A\boldsymbol{x} \leq b,\ \ \boldsymbol{x} \in [0;1]^l).$$

Eine **Lagrange-Relaxation** erhält man, indem alle Nebenbedingungen bis auf eine aktive Nebenbedingung (Nebenbedingung q) weggelassen werden. Dadurch ist das Problem leicht lösbar. Die Verletzungen der restlichen Nebenbedingungen werden gewichtet durch Lagrange-Multiplikatoren $\lambda \geq 0$ in die Zielfunktion aufgenommen

$$\tilde{P}_L: \quad z_L = \max\left(\boldsymbol{cx} + \lambda\left(\boldsymbol{b}_{\sim \boldsymbol{q}} - A_{\sim q}\boldsymbol{x}\right) \mid A_q\boldsymbol{x} \leq \boldsymbol{b_q},\ \ \boldsymbol{x} \in \{0,1\}^l\right),$$

wobei $\boldsymbol{b}_{\sim \boldsymbol{q}}$ der Vektor $\boldsymbol{b}$ ist ohne die q-te Zeile und $A_{\sim q}$ ist die Matrix A ebenfalls ohne die q-te Zeile. Dementsprechend sind A_q und $\boldsymbol{b_q}$ jeweils die q-te Zeile von A bzw. $\boldsymbol{b}$.

Bei einer **Surrogate-Relaxation** werden alle Nebenbedingungen zu einer Nebenbedingung zusammengefasst. Dazu werden die einzelnen Bedingungen mit den Faktoren $\boldsymbol{\mu}$ gewichtet und aufsummiert

$$\tilde{P}_S: \quad z_S = \max(\boldsymbol{cx} \mid \boldsymbol{\mu}(A\boldsymbol{x} - \boldsymbol{b}) \leq 0,\ \ \boldsymbol{x} \in \{0,1\}^l).$$

Branching

Der Algorithmus erzeugt einen Suchbaum durch wiederholte Hinzunahme von Bedingungen. In einem Branching-Schritt wählt der Algorithmus eine Variable aus, deren Wert nicht ganzzahlig ist, und fügt die Bedingungen $x_{k,j} = 0$ und $x_{k,j} = 1$ hinzu. Hieraus entstehen zwei Äste im Suchbaum. Dieser Prozess kann durch einen Binärbaum dargestellt werden, wobei

die Knoten die hinzugefügten Bedingungen darstellen (siehe Abbildung 3.3). Hierbei muss die Reihenfolge der Variablen im Baum nicht mit deren Indizes übereinstimmen.

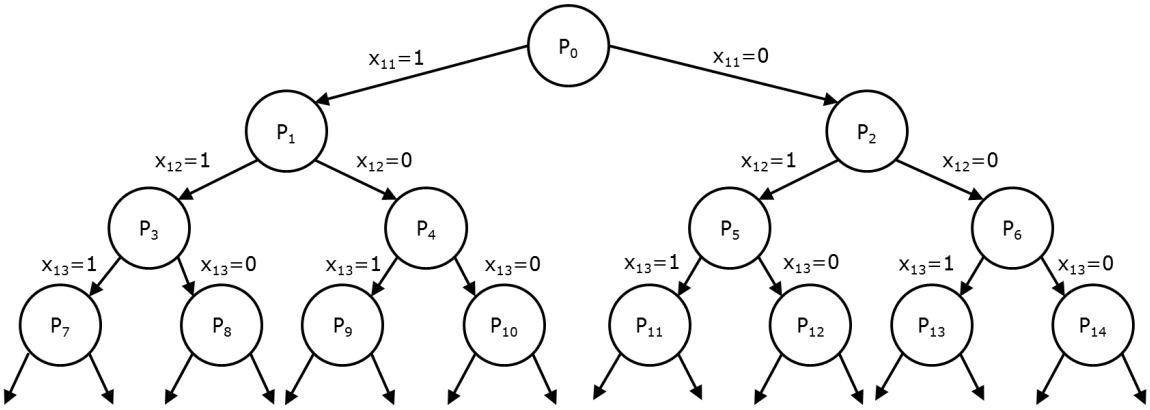


Abbildung 3.3: Binärbaum zur Darstellung des Branchings der Branch-and-Bound-Methode

Branch-Entscheidung

Die Branch-Entscheidung wird nachfolgend anhand der LP-Relaxation erläutert, da diese auch zur Ermittlung der OFR genutzt wurde. An jedem Knoten löst der Algorithmus ein LP-Relaxation-Problem mit den Bedingungen des Knotens und entscheidet abhängig vom Ergebnis, ob an diesem Knoten weitere Äste erzeugt werden oder ob zu einem anderen Knoten gesprungen wird. Bei den zuvor genannten Fällen 1-3 wird keine weitere Verzweigung vorgenommen:

- **Fall 1**: $\overline{z_l} \leq \underline{z}$: Der Knoten wird aus dem Baum entfernt und keine weiteren Äste unter diesem Knoten werden betrachtet. Anschließend geht der Algorithmus zu einem neuen Knoten über.
- **Fall 2**: $\overline{z_l} > \underline{z}$ und die optimale Lösung von $\tilde{P}_l$ ist eine zulässige binäre ganzzahlige Lösung: Die beste ganzzahlige Lösung wird aktualisiert und der Algorithmus geht zu einem neuen Knoten über.
- **Fall 3**: $\tilde{X}_l = \emptyset$: Der Knoten wird aus dem Baum entfernt und keine weiteren Äste unter diesem Knoten werden betrachtet. Anschließend geht der Algorithmus zu einem neuen Knoten über.

Für die Auswahl des neuen Knotens sind unterschiedliche Strategien möglich z.B.

- **Tiefensuche**: Bei jedem Knoten im Suchbaum wird geprüft, ob es einen Kind-Knoten eine Ebene tiefer gibt, der noch nicht untersucht wurde. Der Algorithmus wählt dann einen dieser Knoten. Gibt es diesen nicht, geht der Algorithmus zu dem Knoten, der sich eine Ebene höher im Suchbaum befindet und wählt einen Kind-Knoten eine Ebene unter dem Knoten.

– **Bester Knoten**: Der Knoten mit der größten oberen Schranke $\overline{z_l}$ des Zielfunktionswertes wird gewählt.

Der Algorithmus führt ein Branching durch, d.h. eine weitere Verzweigung an dem Knoten, wenn die Lösung der LP-Relaxation noch nicht ganzzahlig ist und der optimale Funktionswert größer ist als der Funktionswert der bisher besten ganzzahligen Lösung. Hierfür sind unterschiedliche Strategien möglich z.B.

– Die Variable mit der minimalen Ganzzahlunzulässigkeit wird gewählt (die Variable, die am nächsten an 0 oder 1 liegt, aber nicht gleich 0 oder 1 ist).
– Die Variable mit der maximalen Ganzzahlunzulässigkeit wird gewählt (die Variable, die am nächsten an 0.5 liegt).

Bounds

Der optimale Wert des LP-Relaxation-Problems stellt eine obere Grenze $\overline{z_l}$ für das ganzzahlige Programmierungsproblem dar. Wenn die Lösung des LP-Relaxation-Problems schon ganzzahlig ist, ist der optimale Wert eine untere Schranke $\underline{z}$ für das ganzzahlige Optimierungsproblem. Wenn der Suchbaum weiter wächst, benutzt der Algorithmus die Grenzen, die im Bounding-Schritt ermittelt wurden, zur Aktualisierung der unteren und oberen Grenzen der Zielfunktionswerte. Die Zielfunktionswertgrenze dient als Schranke, um unnötige Äste abzuschneiden.

Beispiel

Zur Verdeutlichung des Branch-and-Bound-Verfahrens bei binären Optimierungsproblemen soll es anhand des folgenden Rucksackproblems erläutert werden (vgl. Suhl und Mellouli 2009)

$$z = \max(135x_1 + 140x_2 + 300x_3 + 330x_4 + 400x_5 + 700x_6)$$

u.d.N.

$$15x_1 + 20x_2 + 25x_3 + 30x_4 + 50x_5 + 70x_6 \leq 115$$

$$x_1, x_2, x_3 x_4, x_5, x_6 \in \{0,1\}$$

Zunächst wird die untere Grenze mithilfe eines Greedy-Verfahrens ermittelt. Ein **Greedy-Verfahren** ist eine Heuristik, die sich dadurch auszeichnet, dass sie in jedem Schritt die Auswahl mit der größtmöglichen Verbesserung der Zielfunktion trifft. Dazu wird im Falle eines Rucksackproblems für jeden Gegenstand der Quotient Nutzen/Gewicht gebildet. Diese Quotienten werden absteigend sortiert und es werden der Reihenfolge nach so viele

Gegenstände hinzugefügt bis der Rucksack voll ist. Wenn ein Gegenstand nicht mehr in den Rucksack passt, wird der nächste Gegenstand geprüft bis alle Gegenstände geprüft wurden. Hierbei ergibt sich ein Nutzen von 905, der für das Branch-and-Bound-Verfahren als untere Grenze herangezogen wird $\underline{z} = 905$.

Anschließend werden zur Berechnung der oberen Grenze alle Variablen des Ausgangsproblems P_0 LP-relaxiert, d.h. die Ganzzahligkeitsbedingungen $x_i \in \{0,1\}$, $i = 1, \dots, 6$ werden fallengelassen und durch die Bedingungen $0 \leq x_i \leq 1$ ersetzt. Dann kann das relaxierte Problem $\tilde{P}_0$ mit Methoden der linearen Optimierung (z.B. Simplexverfahren oder Innere-Punkte-Methode) gelöst werden. Der optimale Funktionswert von $\tilde{P}_0$ ist die obere Grenze $\overline{z_0} = 1230$ des binären Optimierungsproblems. Der optimale Funktionswert des binären Problems kann nicht größer sein als diese obere Schranke.

Im nächsten Schritt muss eine noch nicht ganzzahlige Variable zur weiteren Verzweigung ausgewählt werden. Dazu gibt es, wie oben erwähnt, unterschiedliche Strategien. In diesem Beispiel wird die Variable mit der maximalen Ganzzahlunzulässigkeit ausgewählt, d.h. die Variable dessen Wert am nächsten an 0.5 liegt. Das ist die Variable $x_6$und es werden die beiden Äste für $x_6 = 1$ und $x_6 = 0$ hinzugefügt. Der gesamte Binärbaum ist in Abbildung 3.4 dargestellt.

Nun muss ein Teilproblem zur weiteren Verzweigung ausgewählt werden. Auch hier stehen verschiedene Strategien zur Verfügung. In diesem Beispiel wird jeweils das Teilproblem mit der größten oberen Schranke ausgewählt, da man sich erhofft, dass dieses auch zur besten Lösung des binären Optimierungsproblems führt. Aus diesem Grund wird P_1 ausgewählt, d.h. x_6 wird auf 1 fixiert. Anschließend werden ebenfalls für die übrigen Variablen die Ganzzahligkeitsbedingungen fallengelassen und das relaxierte Teilproblem wird gelöst.

Dieses Verfahren wird in dieser Weise weitergeführt. Ein Teilproblem wird nicht weiterbetrachtet, wenn einer der drei oben genannten Fälle eintritt. Fall 1, die obere Grenze des Teilproblems ist kleiner als die aktuell beste ganzzahlige Lösung, tritt bei den Teilproblemen P_2 und P_9 ein. Die Teilprobleme P_5 und P_{10} werden hingegen nicht weiterverzweigt, da sie bereits ganzzahlig sind und P_5 liefert auch die beste ganzzahlige Lösung des Optimierungsproblems. Hierbei wird eine Variable mit einer Toleranz von 10^{-8} als ganzzahlig betrachtet. Die restlichen Teilprobleme werden nicht weiterbetrachtet, da die Gewichtsbeschränkung von 115 überschritten wird.

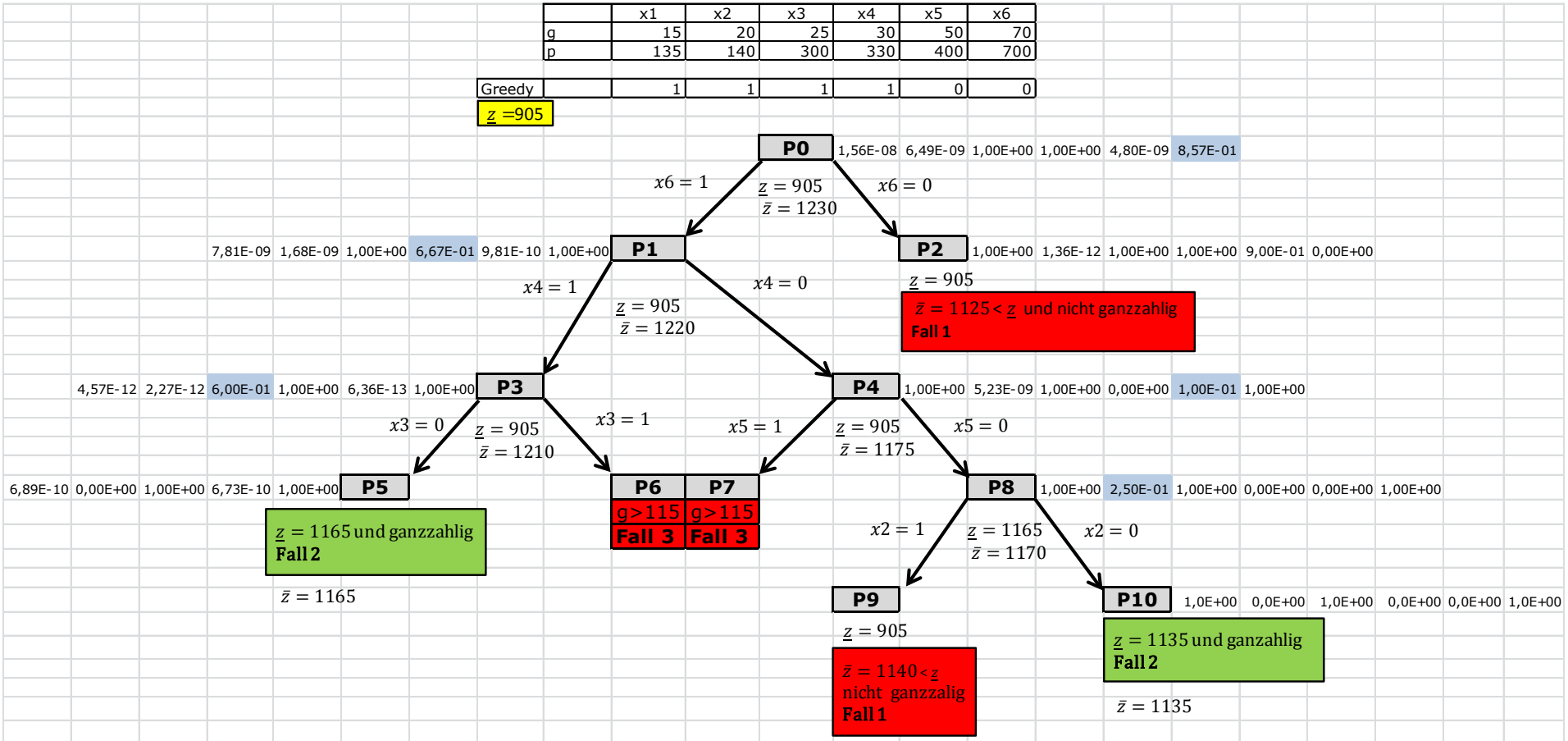


Abbildung 3.4: Binärbaum zur Darstellung der Branch-and-Bound-Methode am Beispiel

3.3.2 NÄHERUNGSVERFAHREN

Aufgrund der hohen Rechenzeiten bei den exakten Verfahren, geht man häufig dazu über „nur“ eine gute zulässige Lösung mithilfe von Näherungsverfahren (auch Heuristiken genannt) zu bestimmen. Diese Verfahren bieten im Gegensatz zu den exakten Verfahren keine Garantie auf Optimalität der gefundenen Lösung. Sie sind lediglich Vorgehensregeln, die für die jeweilige Problemstruktur sinnvoll und erfolgsversprechend erscheinen (vgl. Domschke und Drexl 2005).

Näherungsverfahren lassen sich anhand ihrer zugrundeliegenden Technik unterteilen in:

- Konstruktive Methoden (auch Eröffnungsverfahren genannt)
- Lokale Suchmethoden (auch Verbesserungsverfahren oder iterative Verfahren genannt)
- Unvollständige exakte Verfahren, z.B. vorzeitig abgebrochenes Branch-and-Bound-Verfahren.

Bei den **konstruktiven Methoden** wird eine Lösung „aus dem nichts“ generiert durch schrittweises Hinzufügen von Lösungskomponenten bis die Lösung komplett ist. Die Reihenfolge, in der die Komponenten hinzugefügt werden, kann zufällig sein oder durch eine heuristische Regel bestimmt werden. Die meisten Verfahren dieser Art streben in jedem Schritt nach der größtmöglichen Verbesserung des Zielfunktionswerts. Verfahren dieser Art bezeichnet man auch als **Greedy-Verfahren**.

Lokale Suchverfahren starten mit einer zulässigen Ausgangslösung, die z.B. durch eine konstruktive Methode ermittelt wird, und versuchen wiederholt durch lokale Änderungen die aktuelle Lösung zu verbessern. Entscheidend für die Effizienz des Verfahrens ist eine geeignete Definition der **Nachbarschaft**. Diese definiert die Menge der Lösungen, die in einem Schritt erreicht werden können und ist dementsprechend stark vom untersuchten Problem abhängig. Zusätzlich muss ein Prüfschema bestimmt werden, wie die Nachbarschaft durchsucht wird und welche Nachbarn akzeptiert werden. Dazu gibt es unterschiedliche Möglichkeiten. Eine von ihnen ist die **First-Fit-Regel**, bei der die erste Lösung akzeptiert wird, die die bisherige verbessert. Im Gegensatz dazu werden bei der **Best-Fit-Regel** alle Nachbarn untersucht und derjenige mit der größten Verbesserung der Zielfunktion wird ausgewählt.

Sowohl die konstruktiven Methoden wie auch die lokalen Suchverfahren haben den Nachteil, dass nur eine begrenzte Anzahl verschiedener Lösungen erzeugt wird und dass sie meist in einem lokalen Optimum von minderer Qualität stoppen. **Metaheuristiken** sollen diese Nachteile umgehen. Dies sind algorithmische Konzepte, die genutzt werden können, um heuristische Methoden zu definieren, die auf unterschiedlichste Probleme anwendbar sind. Sie führen eine zugrundliegende problem-spezifische Heuristik (konstruktive Methode oder lokales Suchverfahren) zu einem Bereich im Suchraum mit Lösungen von hoher Qualität. Beispiele für Metaheuristiken sind Simulated Annealing, Tabu-Suche, Greedy Randomized Adaptive Search Procedure, Genetische Algorithmen und Ameisenalgorithmen. Diese Verfahren werden im Folgenden erläutert. Eine Übersicht weiterer Metaheuristiken kann in (Gendreau und Potvin, 2010) gefunden werden.

SIMULATED ANNEALING

Beim Simulated Annealing (simuliertes Abkühlen) wird der physikalische Abkühlungsprozess nachgebildet. Lokale Optima sollen verhindert werden, indem auch schlechtere Lösungen mit einer gewissen Wahrscheinlichkeit akzeptiert werden.

Der generelle Ablauf des Simulated Annealing ist in Algorithmus 3.2 dargestellt: ausgehend von einer Anfangslösung $\boldsymbol{x_0}$ wird in jedem Schritt ein Nachbar $\boldsymbol{x'} \in \mathcal{N}(\boldsymbol{x})$ generiert. Ist der Funktionswert $z(\boldsymbol{x'})$ besser als der Funktionswert $z(\boldsymbol{x})$ der vorherigen Lösung wird die Lösung $\boldsymbol{x'}$ akzeptiert. Ist $z(\boldsymbol{x'})$ hingegen schlechter so wird $\boldsymbol{x'}$ nur mit einer gewissen Wahrscheinlichkeit p akzeptiert, die von der Different $z(\boldsymbol{x'}) - z(\boldsymbol{x})$ und einen Parameter T, der Temperatur genannt wird, abhängt.

Die **Akzeptanzwahrscheinlichkeit** p wird meist wie folgt berechnet

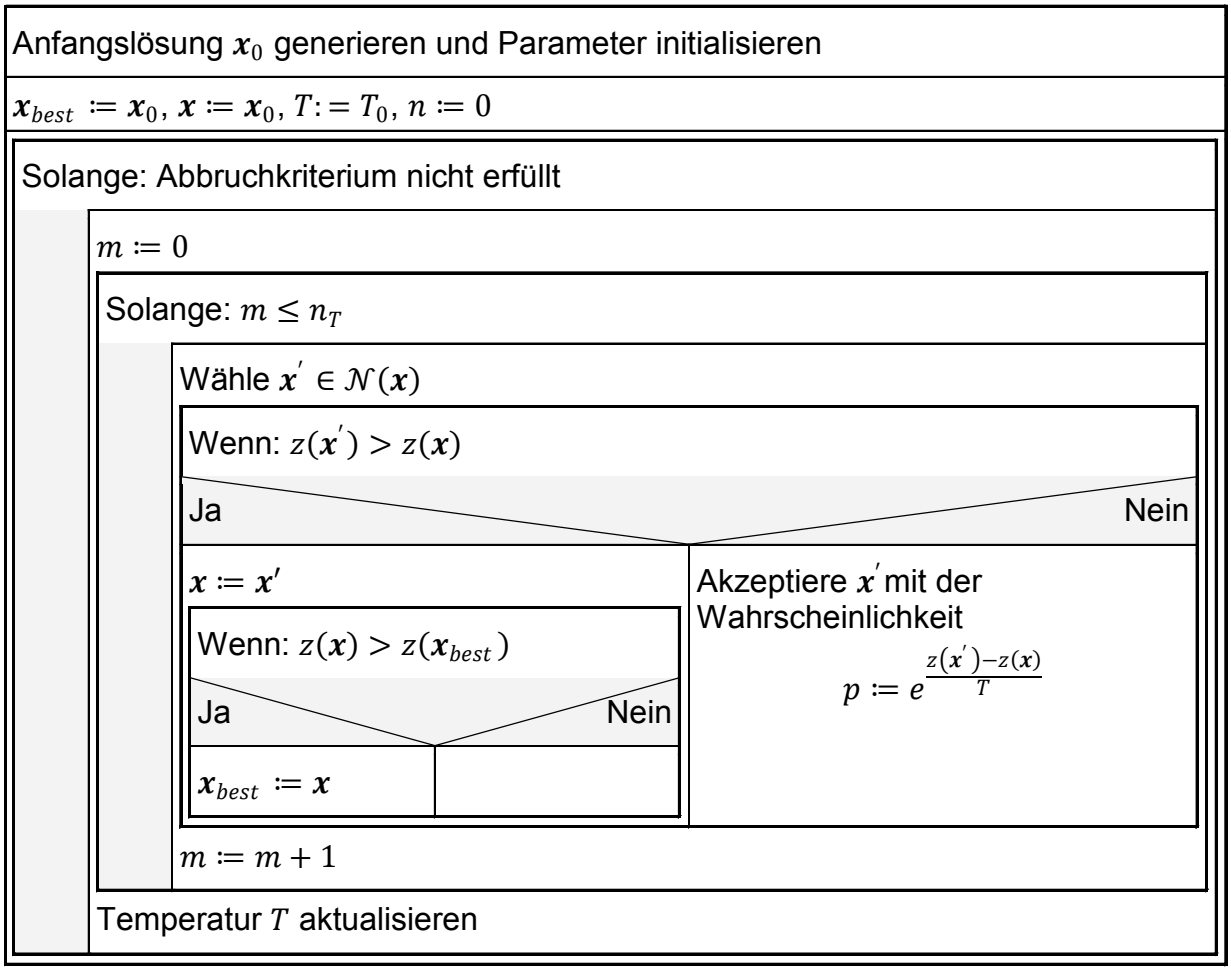
$$p = e^{\frac{z(\boldsymbol{x'})-z(\boldsymbol{x})}{T}}.$$

Die Temperatur T wird während des Algorithmus gemindert und soll im Laufe des Verfahrens langsam gegen 0 streben. Somit ist für große Werte von T die Wahrscheinlichkeit der Akzeptanz schlechterer Werte nahezu 1 und bei kleinen Werten für T $(T \to 0)$ strebt die Akzeptanzwahrscheinlichkeit gegen 0 $(p \to 0)$. Es sind unterschiedliche **Abkühlungsschemata** denkbar (vgl. Gerdes et al. 2004), z.B.:

- Konstant: $T(n) = k$ konstant,
- Arithmetisch: $T(n) = T(n-1) - k$ mit einer konstanten Temperaturreduktion k,

– Geometrisch: $T(n) = \alpha(n) \cdot T(n-1)$.

Algorithmus 3.2: Allgemeiner Ablauf des Simulated Annealing



TABU-SUCHE

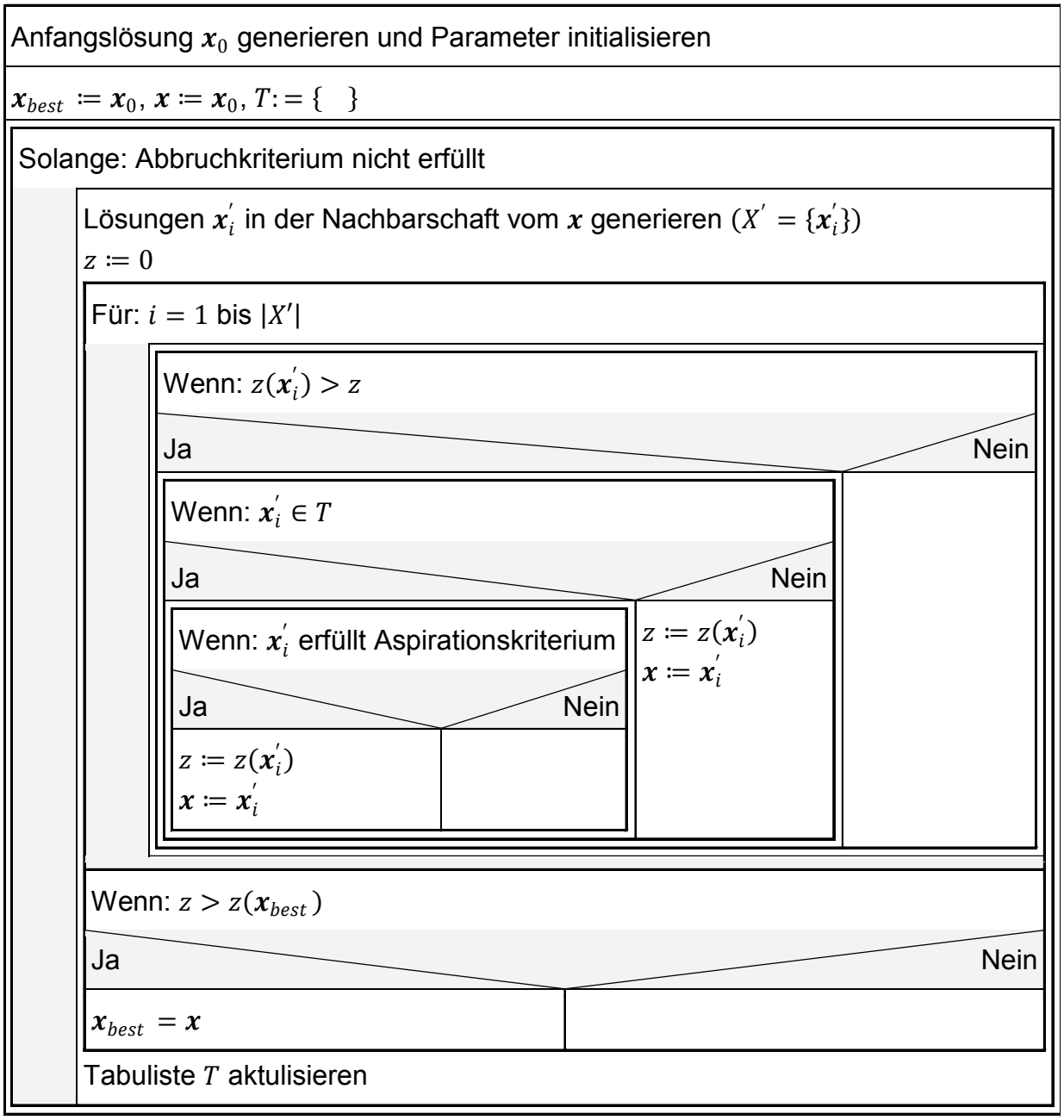
Bei der Tabu-Suche wird zur Steuerung der Suche eine systematische Speicherung eingesetzt. Der allgemeine Ablauf dieses Verfahrens wird in Algorithmus 3.3 dargestellt. In jedem Schritt wird durch eine lokale Suche der bestmögliche Schritt ermittelt von einer Lösung $\boldsymbol{x}$ zu einer Nachbarlösung $\boldsymbol{x}'$. Auch wenn der Funktionswert von $\boldsymbol{x}'$ schlechter ist als der von $\boldsymbol{x}$ wird er dennoch akzeptiert. Dies ist auch der Unterschied zum Simulated Annealing, bei dem eine schlechtere Lösung nur mit einer bestimmten Wahrscheinlichkeit akzeptiert wird.

Um zu verhindern, dass die lokale Suche sofort wieder zu einer vorherigen Lösung zurückkehrt, werden kürzlich akzeptierte Lösungen gespeichert und es wird verboten dorthin

zurückzukehren. Dieser Tabustatus wird für eine bestimmte Anzahl Iterationen aufrechterhalten.

Dadurch werden eventuell Schritte zu besseren, bisher noch nicht untersuchten Lösungen verboten. Ein **Aspirationskriterium** kann dazu genutzt werden den Tabustatus bestimmter Schritte, d.h. Schritte, die zu einer besseren Lösung führen als die aktuell beste Lösung, zu überschreiben.

Algorithmus 3.3: Allgemeiner Ablauf der Tabu-Suche



Beispiel

Das folgende binäre Optimierungsproblem soll beispielhaft das Vorgehen bei der Tabu-Suche verdeutlichen (vgl. Zimmermann 2008):

$$z = \min(20x_1 + 25x_2 - 30x_3 - 45x_4 + 40x_5)$$

u.d.N.

$$x_1 + x_2 - x_3 + x_4 + x_5 \geq 1$$
$$x_1 + x_2 - x_4 + 2x_5 \geq 2$$
$$x_2 + x_3 + x_5 \leq 2$$
$$-x_2 + x_4 + x_5 \leq 1$$
$$x_1, x_2, x_3 x_4, x_5 \in \{0,1\}$$

Bei diesem Beispiel wurden folgende Kriterien festgelegt:

- **Nachbarschaft**: Lösungen, die durch Verändern einer Position des Vektors x entstehen, d.h. ist der Wert der Variable $x_i = 0$ so wird er auf 1 gesetzt und umgekehrt.
- **Länge der Tabuliste**: 3
- **Aspirationskriterium**: Der Tabustatus wird übergangen, wenn der Schritt einen niedrigeren Zielfunktionswert erlangt als zu dem Zeitpunkt, bevor der Schritt verboten wurde.
- **Verletzungen der der Nebenbedingungen**: werden durch Strafkosten, die zu der Zielfunktion hinzuaddiert werden, berücksichtigt. Für die Verletzung der ersten beiden Ungleichungen werden pro Einheit 70 Strafkosten hinzuaddiert und bei Verletzung der dritten und vierten Ungleichung 100.

Die ersten fünf Iterationen sind in Abbildung 3.5 dargestellt. Das Verfahren startet mit der Lösung $x^0 = (1,0,0,0,1)$ mit $z_0 = 60$.

Iteration 1: Es werden die fünf möglichen Lösungen in der Nachbarschaft von x^0 gebildet und durch die Zielfunktion bewertet. Der niedrigste Zielfunktionswert ergibt sich bei der Änderung von x_3 $(0 \rightarrow 1)$. Die Lösung für Iteration 1 lautet $x^1 = (1,0,1,0,1)$ mit $z_1 = 30$.

Iteration 2: Der komplementäre Schritt $x_3 = 0$ mit dem Aspirationskriterium 60 ist ab jetzt für drei Iterationen tabu. Nun werden alle Nachbarn von x^1 gebildet. Den niedrigsten Zielfunktionswert erhält man bei der Änderung von x_3 $(1 \rightarrow 0)$. Dieser Schritt ist aber Tabu und auch das Aspirationskriterium ist nicht erfüllt. Deshalb wird die zweibeste Änderung ausgewählt $x_1 = 0$. Die Lösung für Iteration 2 lautet $x^2 = (0,0,1,0,1)$ mit $z_2 = 80$.

Iteration 3: Nun ist zusätzlich auch der Schritt $x_1 = 1$ mit dem Aspirationskriterium 30 tabu. Den niedrigsten Zielfunktionswert erhält man bei der Änderung von x_1 $(0 \rightarrow 1)$. Dieser Schritt ist aber Tabu und auch das Aspirationskriterium ist nicht erfüllt. Deshalb wird die zweibeste Änderung geprüft. Sie ist auch tabu, aber das Aspirationskriterium ist erfüllt und somit wird die Änderung $x_3 = 0$ ausgewählt. Die Lösung für Iteration 3 lautet $x^3 = (0,0,0,0,1)$ mit $z_3 = 40$.

i	x1	x2	x3	x4	x5	Tabu?	NB1	NB2	NB3	NB4	z ohne Strafe	Strafe	z mit Strafe	Aspirationskriterium erfüllt	Tabuliste
0	1	0	0	0	1						60	0	60		
1	0	0	0	0	1		0	0	1	0	40	0	40		
	1	1	0	0	1		2	2	0	1	85	0	85		
	1	0	1	0	1		0	1	0	0	30	0	30		
	1	0	0	1	1		2	0	1	-1	15	100	115		
	1	0	0	0	0		0	-1	2	1	20	70	90		
2	0	0	1	0	1		-1	0	0	0	10	70	80		x3=0 (60)
	1	1	1	0	1		1	2	-1	1	55	100	155		
	1	0	0	0	1	Tabu	1	1	1	0	60	0	60	60<60 nein	
	1	0	1	1	1		1	0	0	-1	-15	100	85		
	1	0	1	0	0		-1	-1	1	1	-10	140	130		
3	1	0	1	0	1	Tabu	0	1	0	0	30	0	30	30<30 nein	x3=0 (60)
	0	1	1	0	1		0	1	-1	1	35	100	135		x1=1 (30)
	0	0	0	0	1	Tabu	0	0	1	0	40	0	40	40<60 ja	
	0	0	1	1	1		0	-1	0	-1	-35	170	135		
	0	0	1	0	0		-2	-2	1	1	-30	280	250		
4	1	0	0	0	1	Tabu	1	1	1	0	60	0	60	60<60 nein	x3=0 (60)
	0	1	0	0	1		1	1	0	1	65	0	65		x1=1 (30)
	0	0	1	0	1	Tabu	-1	0	0	0	10	70	80	80<80	x3=1 (80)
	0	0	0	1	1		1	-1	1	-1	-5	170	165		
	0	0	0	0	0		-1	-2	2	1	0	210	210		
5	1	1	0	0	1	Tabu	2	2	0	1	85	0	85	68<30 nein	x1=1 (30)
	0	0	0	0	1	Tabu	0	0	1	0	40	0	40	40<40 nein	x3=1 (80)
	0	1	1	0	1	Tabu	0	1	-1	1	35	100	135	135<80 nein	x2=0 (40)
	0	1	0	1	1		2	0	0	0	20	0	20		
	0	1	0	0	0		0	-1	1	2	25	70	95		

Abbildung 3.5: Beispiel zur Tabu-Suche

Iteration 4: Nun ist zusätzlich auch der Schritt $x_3 = 1$ mit dem Aspirationskriterium 80 tabu. Den niedrigsten Zielfunktionswert erhält man bei der Änderung von x_1 $(0 \rightarrow 1)$. Dieser Schritt ist aber Tabu und auch das Aspirationskriterium ist nicht erfüllt. Deshalb wird die zweibeste Änderung ausgewählt $x_2 = 1$. Die Lösung für Iteration 4 lautet $x^4 = (0,1,0,0,1)$ mit $z_3 = 65$.

Iteration 5: Nun ist auch der Schritt $x_2 = 0$ mit dem Aspirationskriterium 40 tabu und zusätzlich wird das erste Element $x_3 = 0$ aus der Tabuliste gestrichen, da die Tabuliste eine Länge von 3 hat. Den niedrigsten Zielfunktionswert erhält man bei der Änderung von x_4 $(0 \rightarrow 1)$. Die Lösung für Iteration 5 lautet $x^5 = (0,1,0,1,1)$ mit $z_4 = 20$.

Hier wird das Verfahren abgebrochen. Es sind verschiedene Abbruchkriterien möglich, z.B. die maximale Anzahl Iterationen wurde erreicht oder es wurde eine bestimmte Anzahl

Iterationen ohne Verbesserung durchgeführt. Die optimale Lösung dieses Beispiels ist $x_{opt} = (1,1,1,0,0)$ mit $z_{opt} = 15$.

GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

Die Greedy Randomized Adaptive Search Procedure, auch **GRASP** genannt, ist eine zufallsbehaftete Greedy-Konstruktionsheuristik, die aus zwei Phasen besteht: der Konstruktionsphase und der lokalen Suchphase (siehe Algorithmus 3.4).

Algorithmus 3.4: Allgemeiner Ablauf der GRASP

Solange: Abbruchkriterium nicht erfüllt	
Konstruiere eine Greedy-Zufallslösung $\boldsymbol{x}$	
Wenn: $\boldsymbol{x}$ unzulässig	
Ja	Nein
Repariere $\boldsymbol{x}$	
Verbessere $\boldsymbol{x}$ in einer lokalen Suche	
Wenn: $z(\boldsymbol{x}) > z(\boldsymbol{x}_{best})$	
Ja	Nein
$\boldsymbol{x}_{best} := \boldsymbol{x}$	

In der Konstruktionsphase wird in jedem Schritt eine Lösungskomponente zu der schon erstellten Teillösung hinzugefügt (siehe Algorithmus 3.5). Dazu wird die Qualität der einzelnen Lösungskomponenten mithilfe einer Greedyfunktion bewertet. Die Greedyfunktion stellt den Gewinnzuwachs dar, wenn das Element in die Teillösung aufgenommen werden würde. Eine bestimmte Anzahl an bestbewerteter Lösungselemente wird in restringierte Kandidatenliste (RKL) aufgenommen. Anschließend wird zufällig eins der Elemente der RKL ausgewählt und zur bisherigen Teillösung hinzugefügt. Diese Prozedur wird solange wiederholt bis kein Kandidatenelement mehr existiert und die Lösung vollständig ist. Wenn die in der Konstruktionsphase gebildete Lösung nicht zulässig ist, ist es notwendig eine Reparaturprozedur anzuwenden, um die Zulässigkeit für den Eintritt in die lokale Suchphase zu erhalten.

Algorithmus 3.5: Konstruktionsphase der GRASP

$\boldsymbol{x} \coloneqq \emptyset$
Initialisiere die Menge der Kandidatenelemente
Berechne die Greedyfunktionswerte der Kandidatenelemente
Solange: mindestens ein Kandidatenelement existiert
Erstelle die restringierte Kandidatenliste (RKL) Wähle zufällig ein Element s aus der RKL $\boldsymbol{x} \coloneqq \boldsymbol{x} \cup \{s\}$ Aktualisiere die Menge der Kandidatenelemente Berechne die Greedyfunktionswerte neu

Anschließend wird versucht die in der Konstruktionsphase gefundene zulässige Lösung mithilfe eines lokalen Suchverfahrens zu verbessern (siehe Algorithmus 3.6). Dazu wird die Nachbarschaft der Lösung solange untersucht bis das lokale Maximum erreicht wird. Hierzu können entweder alle Nachbarn untersucht werden und der Beste unter ihnen wird ausgewählt (**Best-Fit-Regel**) oder es wird der erste Nachbar ausgewählt, der besser ist als die aktuell beste Lösung (**First-Fit-Regel**).

Algorithmus 3.6: Lokale Suchphase der GRASP

Solange: $\boldsymbol{x}$ ist nicht lokal optimal
Finde $\boldsymbol{x}' \in \mathcal{N}(\boldsymbol{x})$ mit $z(\boldsymbol{x}') > z(\boldsymbol{x})$ $\boldsymbol{x} \coloneqq \boldsymbol{x}'$

EVOLUTIONÄRE ALGORITHMEN

Bei evolutionären Algorithmen handelt es sich um naturinspirierte Optimierungsverfahren, die die Prinzipien der biologischen Evolution nachahmen. Die biologische Evolution basiert auf der Theorie von Charles Darwin und erklärt die Vielfalt und Komplexität der Lebewesen. Durch zufällige Variation entstehen Individuen mit vorteilhaften Eigenschaften. Diese Individuen werden durch die natürliche Auslese (**Selektion**) ausgewählt und haben bessere Fortpflanzungschancen, wodurch sie mehr Nachkommen haben und sich die besseren Eigenschaften ausbreiten. Die Variationen können einerseits durch zufällige Veränderungen der Gene hervorgerufen werden (**Mutation**) und anderseits durch den Austausch von Teilchromosomen entstehen (**Rekombination**). Die Evolution ist ein Kreislauf von Neuerschaffung ⇨ Überlebensprüfung ⇨ Vermehrung der Besten ⇨ Verbesserung des

Vorhandenen. Dieser Kreislauf wird auf die evolutionären Algorithmen übertragen. Für die Erläuterung weiterer wichtiger Grundbegriffe der biologischen Evolution siehe z.B. (Kruse et al. 2011).

Die evolutionären Faktoren Mutation, Rekombination und Selektion werden in einen Algorithmus übersetzt (siehe Algorithmus 3.7). Dazu wird zunächst eine Startpopulation mit μ Individuen generiert. Jedes Individuum stellt hierbei eine potenzielle Lösung des Optimierungsproblems dar. Mithilfe einer Fitnessfunktion wird die Qualität jeder potenziellen Lösung bewertet. Anschließend werden in jeder Iteration die Lösungskandidaten in drei Schritten verbessert:

1. **Rekombination**: Es werden für jede Lösung ρ Eltern ausgewählt, meist $\rho = 2$, und deren Merkmale werden zu einer neuen Lösung kombiniert.
2. **Mutation**: Die Lösungen werden einer kleinen, zufälligen Änderung unterzogen.
3. **Selektion**: Auf Basis der Fitnesswerte werden von den erzeugten λ Nachkommen μ ausgewählt.

Diese Schritte werden solange durchgeführt bis das Abbruchkriterium erfüllt ist.

Algorithmus 3.7: Allgemeiner Ablauf des evolutionären Algorithmus (vgl. Kramer 2009)

```
Initialisiere Individuen x_i aus Population P
Bewerte Individuen x_i aus P
Solange: Abbruchkriterium nicht zutrifft
    Für: i = 1 bis λ
        Wähle ρ Eltern aus P und erzeuge x_i durch Rekombination
        Mutiere x_i
        Bewerte x_i
        Füge x_i zu P' hinzu
    Selektiere μ Individueen aus P' und setze P := P'
```

Es gibt verschiedene Grundformen von evolutionären Algorithmen (siehe z.B. Weicker 2007). Im Folgenden soll die **genetischen Algorithmen** näher betrachtet werden. Bei der Implementierung eines genetischen Algorithmus müssen zunächst folgende Randbedingungen beachtet werden (Gerdes 2004):

- **Repräsentation des Suchraums**: Die Elemente werden als Chromosome (Zeichenkette mit fester Länge) mit einem binären Alphabet $\mathcal{A} = \{0,1\}$ kodiert. Hierbei werden die Chromosome in Segmente unterteilt, die den einzelnen Genen entsprechen.

- **Startpopulation**: Es werden μ Chromosome meist zufällig erzeugt. Hierbei liefern einfache Heuristiken meist schon gute Chromosome für die Startpopulation.
- **Fitnessfunktion**: Sie entspricht der Zielfunktion in anderen Optimierungsverfahren. Ihre Bewertung ist entscheidend für die Überlebenswahrscheinlichkeit der Chromosome.
- **Selektion**: Basierend auf der Fitness wird ein Teil der Population ausgewählt, die Restlichen werden verworfen. In der Literatur gibt es zur Selektion unterschiedliche Methoden. Zwei von ihnen sollen an dieser Stelle vorgestellt werden:
  - **Rouletterad-Selektion**: Hierbei werden die besten Lösungen mit einer hohen Wahrscheinlichkeit ausgewählt und schlechte mit einer geringen. Jedes Chromosom $\boldsymbol{x_i}$ bekommt eine zu seiner Fitness proportionalen Wahrscheinlichkeit zugeordnet

$$p(\boldsymbol{x_i}) = \frac{fit(\boldsymbol{x_i})}{\sum_{\boldsymbol{x_j} \in \mathcal{P}} fit(\boldsymbol{x_j})}. \tag{3-1}$$

    Anschließend werden μ aus den λ Chromosomen ausgewählt. Dazu bekommt jedes Chromosom ein Segment gemäß seiner Überlebenswahrscheinlichkeit auf dem Rouletterad. Es wird eine gleichverteilte Zufallszahl erzeugt und das Chromosom, zu dem das entsprechende Segment auf dem Rouletterad gehört, wird ausgewählt.
  - **Wettkampfselektion:** Es werden q Chromosome zufällig (unabhängig von der Fitnessfunktion) ausgewählt. Das beste Chromosom wird in die neue Population übernommen. Dieses Verfahren wird μ-mal durchgeführt. Bereits ausgewählte Chromosome werden wieder in die Ursprungspopulation eingesetzt und können somit im nächsten Wettkampf wieder ausgewählt werden.
- **Genetische Operatoren**:
  - **Mutation**: Jedes Gen wird mit der Wahrscheinlichkeit p_m mutiert, d.h. der Werte des Gens wird von 1 auf 0 geändert oder umgekehrt. Meist wird $p_m = \frac{1}{l}$ gewählt, wobei l die Länge des Chromosoms ist.
  - **Rekombination**: Aus zwei Chromosomen entstehen durch Austausch von Teilketten zwei neue Chromosome. Welche Chromosome gekreuzt werden, wird hierbei zufällig entschieden. Beim **One-Point-Crossover** werden die beiden Chromosome mit einer Wahrscheinlichkeit p_c, meist $0.5 \leq p_c \leq 1$, gekreuzt. Im Falle einer Kreuzung, wird anschließend zufällig ein Kreuzungspunkt ausgewählt, ab dem die Gene miteinander vertauscht werden. Beim **Two-Point-Crossover** werden entsprechend zwei Kreuzungspunkte ausgewählt und beim **Uniform-Crossover** wird für jedes einzelne Gen ausgewürfelt, ob es ausgetauscht wird oder nicht.

Weitere problemangepasste Operatoren und Verfahren der genetischen Algorithmen können in (Gerdes 2004) gefunden werden.

AMEISENALGORITHMEN

Ameisenalgorithmen modellieren das Verhalten von Ameisen bei der Futtersuche, um Optimierungsaufgaben zu lösen. Bei der Suche nach Futter scheiden Ameisen entlang ihres Weges einen Duftstoff aus, der **Pheromon** genannt wird. Dieser Duftstoff wird von den anderen Ameisen wahrgenommen und beeinflusst sie bei ihrer Wegwahl zur Futterstelle. Wurde auf einem Weg mehr Pheromon ablegt als auf den anderen möglichen Wegen, so wird dieser Weg von einer nachfolgenden Ameise mit einer hohen Wahrscheinlichkeit ausgewählt.

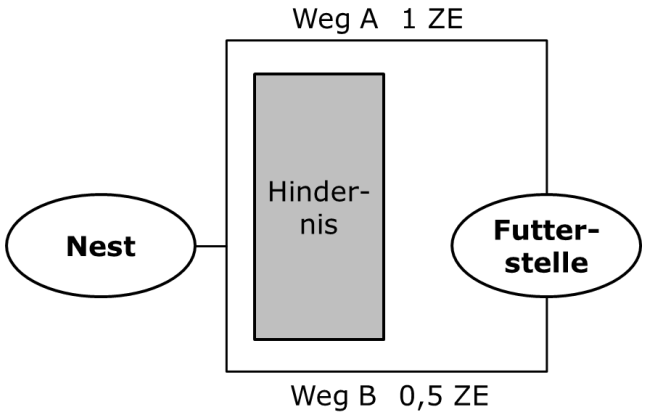


Abbildung 3.6: Vorbild aus der Natur: Futtersuche bei den Ameisen

In Abbildung 3.6 ist ein möglicher Fall der Futtersuche dargestellt, an dem beispielhaft das Verhalten der Ameisen erklärt werden soll. Zwischen dem Nest der Ameisen und der Futterstelle gibt es ein Hindernis, so dass die Ameisen das Futter auf zwei möglichen Wegen erreichen können. Für den langen Weg A benötigen sie 1 Zeiteinheit (ZE) und für den kurzen Weg B 0,5 ZE. Zu jeder ZE starten zwei Ameisen auf den Weg zum Futter. Die ersten beiden Ameisen treffen ihre Wegwahl zufällig, da noch kein Pheromon auf den Wegen hinterlassen wurde. Es werde angenommen, dass sich die eine Ameise für den langen Weg entscheidet (Ameise A) und die andere für den Kurzen (Ameise B). Nach einer ZE starten zwei neue Ameisen. Sie haben auch wieder die Wahl zwischen den beiden Wegen, auf denen nun jedoch schon Pheromonspuren von ihren Vorgängern hinterlassen wurden. Auf dem langen Weg A wurde nach einer ZE einmal Pheromon hinterlassen, auf dem kurzen Weg B allerdings schon zweimal, da die Ameise B nach einer ZE schon wieder am Nest angekommen ist. Die nachfolgenden Ameisen werden mit einer hohen Wahrscheinlichkeit den kurzen Weg B

wählen, da sie die dort nach einiger Zeit vermehrten Pheromonspuren dazu verleiten. Auf diese Weise entsteht auf Weg B eine Ameisenstraße (vgl. Proß 2007).

Dieses Verhalten wurde auf eine Vielzahl von Algorithmen übertragen (siehe Dorigo und Stützle 2004). Im Folgenden werden zunächst die Grundzüge der Ameisenalgorithmen vorgestellt und anschließend wird der MAX-MIN Ant System näher erläutert.

Beim Ameisenalgorithmus bildet jede künstliche Ameise in jeder Iteration zufallsbehaftet eine Lösung für das betrachtete Optimierungsproblem. Diese Lösungen entstehen durch schrittweises Hinzufügen von Lösungskomponenten zu einer Teillösung. Bei dieser Lösungskonstruktion werden zwei Aspekte berücksichtigt:

1. Heuristische Informationen über das zu lösende Problem,
2. Pheromonspuren, die sich dynamisch zur Laufzeit ändern, um die Sucherfahrung der Ameisen miteinzubeziehen.

Algorithmus 3.8: Allgemeiner Ablauf des Ameisenalgorithmus

Initialisierung
Solange: Abbruchkriterium nicht zutrifft
Erstelle Ameisenlösungen
Verwende lokale Suche %optional
Aktualisiere Pheromone

In Algorithmus 3.8 werden die allgemeinen Grundzüge der Armeisenalgorithmen dargestellt. Zu Beginn werden die Parameter des Algorithmus und die Pheromonspuren initialisiert.

Anschließend konstruieren in jedem Schleifendurchlauf m Ameisen jeweils eine Lösung für das betrachtete Optimierungsproblem unter Berücksichtigung der heuristischen Information und der Pheromonspuren. Dazu wird in jedem Konstruktionsschritt die Teillösung $\boldsymbol{x}_t$ um eine Lösungskomponente $c_i^j \in \mathcal{N}(\boldsymbol{x}_t)$ erweitert. Die Menge $\mathcal{N}(\boldsymbol{x}_t)$ umfasst alle Lösungskomponenten, die unter Einhaltung der Zulässigkeit zur Teillösung hinzugefügt werden können. Falls eine Teillösung unter Einhaltung der Zulässigkeit nicht mehr erweiterbar ist, kann entweder die Konstruktion abgebrochen werden oder es kann eine vollständige unzulässige Lösung konstruiert werden. Unzulässige Lösungen können dann in Abhängigkeit des Ausmaßes, mit dem sie die Nebenbedingungen verletzen, bestraft werden.

Die Lösungskomponente, die hinzugefügt werden soll, wird mithilfe eines stochastischen Auswahlverfahrens bestimmt. Hierzu gibt es unterschiedliche Möglichkeiten. Am häufigsten wird die Auswahlregel des Ant System verwendet

$$p(c_i^j|\boldsymbol{x}_t) = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{c_i^l \in \mathcal{N}(\boldsymbol{x}_t)} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} \qquad \forall c_i^j \in \mathcal{N}(\boldsymbol{x}_t). \tag{3-2}$$

Nachdem alle Ameisen ihre Lösungen konstruiert haben, können diese in einer optionalen lokalen Suchphase verbessert werden.

Den Abschluss einer Iteration bildet die Aktualisierung der Pheromonspuren, um die Sucherfahrung der Ameisen widerzugeben. Hierbei werden zwei Mechanismen angewendet:

1. Pheromonablage: erhöht das Pheromonlevel von Lösungskomponenten, die zur Menge X_{gut} der guten Lösungen gehören.
2. Pheromonverwitterung: verringert die Pheromonspur, die von früheren Ameisen abgesondert wurde, um eine schnelle Konvergenz des Algorithmus zu vermeiden.

Hieraus ergibt sich die folgende Formel zur Aktualisierung der Pheromonspuren

$$\tau_{ij} = (1-\rho)\tau_{ij} + \sum_{\boldsymbol{x} \in X_{gut} | c_i^j \in \boldsymbol{x}} g(\boldsymbol{x}),$$

wobei $\rho \in (0,1]$ die Verwitterungsrate ist und $g(\boldsymbol{x}): X \to \mathbb{R}^+$ ist die Bewertungsfunktion, die die Qualität der gefundenen Lösung misst. Handelt es sich um ein Maximierungsproblem, dann muss gelten $z(\boldsymbol{x}) \geq z(\boldsymbol{x}') \Rightarrow g(\boldsymbol{x}) \geq g(\boldsymbol{x}')$ und bei einem Minimierungsproblem muss gelten $z(\boldsymbol{x}) \leq z(\boldsymbol{x}') \Rightarrow g(\boldsymbol{x}) \geq g(\boldsymbol{x}')$.

Die verschiedenen Ameisenalgorithmen unterscheiden sich meist in der Art der Pheromonablage. Hierbei sind verschiedene Arten, wie X_{gut} implementiert werden kann, denkbar. Meist ist $X_{gut} \subseteq (X_{iter} \cup \{\boldsymbol{x}_{best}\})$, wobei X_{iter} die Menge der Lösungen ist, die in der aktuellen Iteration erzeugt wurden und $\boldsymbol{x}_{best}$ ist die aktuell beste Lösung.

Beim **MAX-MIN Ant System** gelten folgende Regeln:

1. Pheromonablage: nur die iterationsbeste oder die aktuell beste Ameise darf Pheromone ablegen. Zudem ist die Bewertungsfunktion gegeben durch $g(\boldsymbol{x}_b) = 1/z(\boldsymbol{x}_b)$ mit $z(\boldsymbol{x})$ als zu minimierende Zielfunktion und $\boldsymbol{x}_b$ ist entweder die iterationsbeste oder die aktuell beste Lösung.

2. Pheromonwerte: der erlaubte Bereich für das Ausmaß der Pheromonspuren ist auf das Intervall $[\tau_{min}; \tau_{max}]$ beschränkt, so dass gilt $\tau_{min} \leq \tau_{ij} \leq \tau_{max} \quad \forall \tau_{ij}$. In (Stützle und Hoos 2000) wird diskutiert, wie diese Grenzen zu setzen sind.
3. Pheromoninitialisierung: die Pheromonspuren werden mit der oberen Pheromongrenze τ_{max} initialisiert. Dies zusammen mit einer kleinen Verwitterungsrate erhöht die Erforschung des Suchgebiets am Start der Suche.
4. Pheromonreinitialisierung: wenn das System stagniert oder wenn keine Verbesserung für eine bestimmte Anzahl aufeinanderfolgender Iterationen generiert werden konnte, werden die Pheromone reinitialisiert.

3.4 Problemspezifische Anpassungen

Die vorgestellten Verfahren zur Ermittlung der OFR müssen jeweils auf dieses spezielle Problem angepasst werden. Gerade bei den vorgestellten Metaheuristiken handelt es sich zunächst um eine abstrakte Abfolge von Schritten, die auf unterschiedlichste Problemstellungen angewendet werden können. Um sie für ein spezielles Problem verwenden zu können, müssen die einzelnen Schritte problemspezifisch implementiert werden. Dazu wird zunächst das Problem der Ermittlung der OFR mithilfe von Matrizen dargestellt (siehe auch Kapitel 2.5), so dass es sich auf folgende Weise ganz allgemein formulieren lässt[1]:

$$\max_{\boldsymbol{x}} \boldsymbol{cx} \quad \text{u. d. N} \begin{cases} A\boldsymbol{x} \leq \boldsymbol{b} \\ \boldsymbol{x} \in \{0,1\}^l \end{cases}, \tag{3-3}$$

wobei **c** und $\boldsymbol{b}$ Vektoren sind, A ist eine Matrix und die Lösung $\boldsymbol{x}$ ist ein binärer Vektor, d.h. seine Einträge können nur 0 oder 1 sein. Der Lösungsvektor hat die Dimension $l = n_t \cdot n_s$ und der Eintrag $x_i = 1$ bedeutet, dass im Schritt $s = \left\lceil \frac{i}{n_t} \right\rceil$ die Transition $i - (s-1) \cdot n_t$ feuert und dementsprechend bedeutet $x_i = 0$, dass sie nicht feuert.[2]

Der Vektor $\boldsymbol{c}$ enthält den Nutzen der einzelnen Transition, der n_s-mal wiederholt wird

$$\boldsymbol{w} = \left(w(t_1), \dots, w\left(t_{n_t}\right)\right)^\top = \left(\omega_1, \dots, \omega_{n_t}\right)^\top$$

$$\boldsymbol{c} = (\boldsymbol{w}, \boldsymbol{w}, \dots, \boldsymbol{w})^\top \in \mathbb{R}^l_{\geq 0}.$$

[1] Es wird zur Erklärung ein Maximierungsproblem zugrunde gelegt. Falls es sich bei den Transitionenbewertungen nicht um Nutzen sondern um Kosten handelt, die minimiert werden sollen, kann ebenso verfahren werden, da $\min(\boldsymbol{cx}) = \max(-\boldsymbol{cx})$.

[2] $s = \lceil i/n_t \rceil$ ist die kleinste ganze Zahl, die größer oder gleich i/n_t ist.

Die Ungleichungsnebenbedingungen (NBl) und (NBu) müssen in jedem Schritt eingehalten werden. Es müssen somit folgende Bedingungen in den einzelnen Schritten gelten:

1.	$\boldsymbol{m_0} - \mathcal{F} \cdot \boldsymbol{x}^1 \geq \boldsymbol{c_l}$	$\boldsymbol{m_0} + \mathcal{G} \cdot \boldsymbol{x}^1 \leq \boldsymbol{c_u}$
2.	$\boldsymbol{m_1} - \mathcal{F} \cdot \boldsymbol{x}^2 \geq \boldsymbol{c_l}$ mit $\boldsymbol{m_1} = \boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1$ $\Rightarrow \quad \boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1 - \mathcal{F} \cdot \boldsymbol{x}^2 \geq \boldsymbol{c_l}$	$\boldsymbol{m_1} + \mathcal{G} \cdot \boldsymbol{x}^2 \leq \boldsymbol{c_u}$ mit $\boldsymbol{m_1} = \boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1$ $\Rightarrow \quad \boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1 + \mathcal{G} \cdot \boldsymbol{x}^2 \leq \boldsymbol{c_u}$
⋮	⋮	⋮
n_s.	$\boldsymbol{m_{n_s-1}} - \mathcal{F} \cdot \boldsymbol{x}^{n_s} \geq \boldsymbol{c_l}$ mit $\boldsymbol{m_{n_s-1}} = \boldsymbol{m_{n_s-2}} + \mathcal{H} \cdot \boldsymbol{x}^{n_s-1} = \cdots =$ $\boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1 + \cdots + \mathcal{H} \cdot \boldsymbol{x}^{n_s-1}$ $\Rightarrow \quad \boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1 + \cdots + \mathcal{H} \cdot \boldsymbol{x}^{n_s-1}$ $- \mathcal{F} \cdot \boldsymbol{x}^{n_s} \geq \boldsymbol{c_l}$	$\boldsymbol{m_{n_s-1}} + \mathcal{G} \cdot \boldsymbol{x}^{n_s} \leq \boldsymbol{c_u}$ mit $\boldsymbol{m_{n_s-1}} = \boldsymbol{m_{n_s-2}} + \mathcal{H} \cdot \boldsymbol{x}^{n_s-1} = \cdots =$ $\boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1 + \cdots + \mathcal{H} \cdot \boldsymbol{x}^{n_s-1}$ $\Rightarrow \quad \boldsymbol{m_0} + \mathcal{H} \cdot \boldsymbol{x}^1 + \cdots + \mathcal{H} \cdot \boldsymbol{x}^{n_s-1}$ $+ \mathcal{G} \cdot \boldsymbol{x}^{n_s-1} \leq \boldsymbol{c_u}$

Hierbei ist $\boldsymbol{m_0}$ der Vektor mit der Anfangsmarkierung des Petri-Netzes und $\boldsymbol{m_k}$ enthält die Markierung nach dem k-ten Feuerungsschritt. Der binäre Vektor $\boldsymbol{x}^i$ enthält die Information, welche Transitionen im i-ten Schritt feuern. Diese Bedingungen werden in der Matrix A und in dem Vektor $\boldsymbol{b}$ zusammengefasst. Hierbei enthält die rechte Seite $\boldsymbol{b}$ der Ungleichungsnebenbedingungen n_s-mal den Term $-(\boldsymbol{c_l} - \boldsymbol{m_0})$ und n_s-mal den Term $(\boldsymbol{c_u} - \boldsymbol{m_0})$

$$\boldsymbol{b} = \left(\underbrace{-(\boldsymbol{c_l} - \boldsymbol{m_0}), \ldots, -(\boldsymbol{c_l} - \boldsymbol{m_0})}_{n_s \, mal} \; \underbrace{(\boldsymbol{c_u} - \boldsymbol{m_0}), \ldots, (\boldsymbol{c_u} - \boldsymbol{m_0})}_{n_s \, mal} \right)^{\mathsf{T}} \in \mathbb{R}^{2 \cdot n_s \cdot n_p}.$$

Die Matrix A hat die Dimension $\left(2 \cdot n_s \cdot n_p \times l\right)$ und setzt sich wie folgt zusammen

$$A = \begin{pmatrix} \mathcal{F} & 0 & 0 & \cdots & 0 \\ -\mathcal{H} & \mathcal{F} & 0 & \cdots & 0 \\ -\mathcal{H} & -\mathcal{H} & \mathcal{F} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -\mathcal{H} & -\mathcal{H} & \cdots & -\mathcal{H} & \mathcal{F} \\ \mathcal{G} & 0 & 0 & \cdots & 0 \\ \mathcal{H} & \mathcal{G} & 0 & \cdots & 0 \\ \mathcal{H} & \mathcal{H} & \mathcal{G} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \mathcal{H} & \mathcal{H} & \cdots & \mathcal{H} & \mathcal{G} \end{pmatrix} \in \mathbb{R}^{2 \cdot n_s \cdot n_p \times l}.$$

Falls nur serielles Feuern erlaubt ist (NBs), müssen die Matrix A und der Vektor $\boldsymbol{b}$ wie folgt erweitert werden

$$\boldsymbol{b} = \left(\underbrace{-(\boldsymbol{c_l} - \boldsymbol{m_0}), \dots, -(\boldsymbol{c_l} - \boldsymbol{m_0})}_{n_s\ mal} \ \underbrace{(\boldsymbol{c_u} - \boldsymbol{m_0}), \dots, (\boldsymbol{c_u} - \boldsymbol{m_0})}_{n_s\ mal} \underbrace{1, \dots, 1}_{n_s\ mal} \right)^{\mathsf{T}} \in \mathbb{R}^{2 \cdot n_s \cdot n_p + n_s}$$

$$A = \begin{pmatrix} \mathcal{F} & 0 & 0 & \cdots & 0 \\ -\mathcal{H} & \mathcal{F} & 0 & \cdots & 0 \\ -\mathcal{H} & -\mathcal{H} & \mathcal{F} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -\mathcal{H} & -\mathcal{H} & \cdots & -\mathcal{H} & \mathcal{F} \\ \mathcal{G} & 0 & 0 & \cdots & 0 \\ \mathcal{H} & \mathcal{G} & 0 & \cdots & 0 \\ \mathcal{H} & \mathcal{H} & \mathcal{G} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \mathcal{H} & \mathcal{H} & \cdots & \mathcal{H} & \mathcal{G} \\ \mathbf{1} & 0 & 0 & \cdots & 0 \\ 0 & \mathbf{1} & 0 & \cdots & 0 \\ 0 & 0 & \mathbf{1} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ 0 & 0 & \cdots & 0 & \mathbf{1} \end{pmatrix} \in \mathbb{R}^{(2 \cdot n_s \cdot n_p + n_s) \times l} \quad \text{mit} \quad \mathbf{1} = \left(\underbrace{1, \dots, 1}_{n_t - mal} \right).$$

Für den Fall, dass eine Zielmarkierung erreicht werden soll (NBz), müssen die Matrix A und der Vektor $\boldsymbol{b}$ in folgender Weise erweitert werden

$$\boldsymbol{b} = \left(\underbrace{-(\boldsymbol{c_l} - \boldsymbol{m_0}), \dots, -(\boldsymbol{c_l} - \boldsymbol{m_0})}_{n_s\ mal} \ \underbrace{(\boldsymbol{c_u} - \boldsymbol{m_0}), \dots, (\boldsymbol{c_u} - \boldsymbol{m_0})}_{n_s\ mal} (\boldsymbol{m_0} - \boldsymbol{m_z}) \right)^{\mathsf{T}} \in \mathbb{R}^{2 \cdot n_s \cdot n_p + n_p}$$

$$A = \begin{pmatrix} \mathcal{F} & 0 & 0 & \cdots & 0 \\ -\mathcal{H} & \mathcal{F} & 0 & \cdots & 0 \\ -\mathcal{H} & -\mathcal{H} & \mathcal{F} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -\mathcal{H} & -\mathcal{H} & \cdots & -\mathcal{H} & \mathcal{F} \\ \mathcal{G} & 0 & 0 & \cdots & 0 \\ \mathcal{H} & \mathcal{G} & 0 & \cdots & 0 \\ \mathcal{H} & \mathcal{H} & \mathcal{G} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ \mathcal{H} & \mathcal{H} & \cdots & \mathcal{H} & \mathcal{G} \\ -\mathcal{H} & -\mathcal{H} & -\mathcal{H} & -\mathcal{H} & -\mathcal{H} \end{pmatrix} \in \mathbb{R}^{(2 \cdot n_s \cdot n_p + n_p) \times l}.$$

Es können auch beide Erweiterungen kombiniert werden.

Im Folgenden sollen zunächst die Definition der Nachbarschaft und der Umgang mit unzulässigen Lösungen für alle Verfahren allgemein erläutert werden. Anschließend wird auf die jeweiligen verfahrensspezifischen Anpassungen eingegangen.

3.4.1 NACHBARSCHAFT

Bei allen vorgestellten Verfahren muss zunächst die Nachbarschaft definiert werden. Hier gibt es verschiedene Möglichkeiten. Nachfolgend werden drei Definitionen vorgestellt, die in den Algorithmen umgesetzt worden sind.

Einkomponentige Änderung: Zwei Lösungen $\boldsymbol{x}$ und $\boldsymbol{x}'$ sind Nachbarn, wenn sie sich nur in einer Komponente unterscheiden, d.h.

$$\left(x_i = x_i' \quad \forall i = \{1, \dots, l\}/\{j\} \ \wedge \ x_j \neq x_j'\right) \ \Leftrightarrow \ \boldsymbol{x}' \in \mathcal{N}(\boldsymbol{x}). \tag{3-4}$$

Wenn $x_j = 1$, dann ist $x_j' = 0$ und umgekehrt.

Zweikomponentige Änderung: Ein Lösungskandidat $\boldsymbol{x}'$ ist ein Nachbar von $\boldsymbol{x}$, wenn gilt

$$\left(x_i = x_i' \quad \forall i = \{1, \dots, l\}/\{j,k\} \ \wedge \ x_j \neq x_j' \ \wedge \ x_k \neq x_k'\right) \Leftrightarrow \ \boldsymbol{x}' \in \mathcal{N}(\boldsymbol{x}). \tag{3-5}$$

Ein- und zweikomponentige Änderungen: Es können auch beide Nachbarschaften kombiniert werden: Alle $\boldsymbol{x}'$ sind Nachbarn, wenn sie sich entweder in einer oder in zwei Komponenten von $\boldsymbol{x}$ unterscheiden.

Wenn in jeder Iteration die gesamte Nachbarschaft von $\boldsymbol{x}$ erzeugt wird, dann sind das im Fall von (3-4) alle Lösungen, die sich nur in einer Komponente von $\boldsymbol{x}$ unterscheiden und im Fall von (3-5) alle Lösungen, die sich in zwei Komponenten unterscheiden.

Die Lösung $\boldsymbol{x} = (1,0,1,1,0)^\mathsf{T}$ hat beispielsweise im Fall der einkomponentigen Änderung die Nachbarn

$$\mathcal{N}_1(\boldsymbol{x}) = \left\{ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\}$$

und im Fall der zweikomponentigen Änderung wären es

$$\mathcal{N}_2(\boldsymbol{x}) = \left\{ \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \right\}.$$

Lässt man ein- und zweikomponentige Änderungen zu, ergibt sich die Menge der aller Nachbarn als $\mathcal{N}_{12}(\boldsymbol{x}) = \mathcal{N}_1(\boldsymbol{x}) \cup \mathcal{N}_2(\boldsymbol{x})$.

3.4.2 Zielfunktion und unzulässige Lösungen

Unzulässige Lösungen können auf unterschiedliche Arten im Algorithmus verankert werden. Eine Lösung ist unzulässig, wenn gilt

$$\exists i \in \{1, \dots, l\}: \ \boldsymbol{a}_i \boldsymbol{x} > b_i,$$

wobei $\boldsymbol{a}_i$ die $i$te Zeile der Matrix A ist und b_i ist die $i$te Komponente des Vektors $\boldsymbol{b}$.

Verbot: Unzulässige Lösungen werden verboten und im Ablauf des Verfahrens werden somit keine unzulässigen Lösungen erzeugt.

Bestrafung: Bei dieser Variante wird das Maximierungsproblem durch ein Minimierungsproblem mit der Zielfunktion

$$z(\boldsymbol{x}) = \begin{cases} z_{LP} - \boldsymbol{c}\boldsymbol{x}, & A\boldsymbol{x} \leq \boldsymbol{b} \\ z_{LP} + p(\boldsymbol{x}), & A\boldsymbol{x} > \boldsymbol{b} \end{cases} \tag{3-6}$$

ersetzt, wobei z_{LP} der optimale Funktionswert der LP-Relaxierung des Problems ist

$$z_{LP} = \max(\boldsymbol{c}\boldsymbol{x} \mid A\boldsymbol{x} \leq \boldsymbol{b}, \ \ \boldsymbol{x} \in [0;1]^l) \tag{3-7}$$

(vgl. Kapitel 3.3.1) und $p(\boldsymbol{x})$ ist ein **Bestrafungsterm**, der das Ausmaß der Unzulässigkeit angibt. Die Bestrafungsfunktion kann von unterschiedlicher Gestalt sein. Zwei Möglichkeiten, die im Rahmen dieser Arbeit umgesetzt worden sind, sind:

$$p(\boldsymbol{x}) = \sum_{i \in U(\boldsymbol{x})} |\boldsymbol{a}_i \boldsymbol{x} - \boldsymbol{b}_i|$$

$$p(\boldsymbol{x}) = |U(\boldsymbol{x})|$$

$$\text{mit} \ \ U(\boldsymbol{x}) = \{i \in \{1, \dots, l\}: \boldsymbol{a}_i \boldsymbol{x} > \boldsymbol{b}_i\}.$$

Bei der ersten Möglichkeit werden die absoluten Größen der Verletzungen der einzelnen Nebenbedingungen aufsummiert und bei der zweiten Möglichkeit wird nur die Anzahl, wie viele Nebenbedingungen verletzt werden, als Maß herangezogen.

Es kann oft sinnvoll sein unzulässige Lösungen zu erlauben, um Bereiche mit „guten“ Lösungskandidaten überhaupt zu erreichen.

3.4.3 Verfahrensspezifische Anpassungen

Die Nachbarschaft und der Umgang mit unzulässigen Lösungen werden in allen vorgestellten Verfahren, wie in den vorangegangenen Kapiteln beschrieben, definiert. Darüber hinaus gibt

es weitere Anpassungen, die speziell für die einzelnen Verfahren vorgenommen werden müssen. Diese werden in diesem Kapitel vorgestellt.

BRANCH-AND-BOUND-VERFAHREN

Für das Branch-and-Bound-Verfahren wird auf den von MATLAB im Rahmen der Optimization Toolbox bereitgestellten Algorithmus `bintprog()` zurückgegriffen. Dieser Algorithmus nutzt die LP-Relaxation und für das Branching stehen verschiedene Optionen zur Verfügung (siehe Matlab 2012b). Das Optimierungsproblem kann in der Form (3-3) direkt an den Algorithmus übergeben werden. Lediglich der Nutzenvektor $\boldsymbol{c}$ muss mit einem negativen Vorzeichen versehen werden, da der Algorithmus für Minimierungsprobleme konzipiert wurde.

SIMULATED ANNEALING

Die Anpassungen des Simulated Annealing beschränken sich auf die Definition der Nachbarschaft und den Umgang mit unzulässigen Lösungen. In jedem Schritt wird zufällig ein $j \in \{1, \dots, l\}$ ausgewählt und das zugehörige Vektorelement x_j wird entweder von $1 \rightarrow 0$ oder $0 \rightarrow 1$ geändert.

Falls unzulässige Lösungen verboten sind, wird nach der Änderung von $x_j \rightarrow x_j'$ geprüft, ob alle Nebenbedingungen erfüllt sind, d.h. $\boldsymbol{A}\boldsymbol{x}' \leq \boldsymbol{b}$. Falls das nicht der Fall ist, wird die Änderung rückgängig gemacht und es wird ein neues $j \in \{1, \dots, l\}$ zufällig ausgewählt. Die Prozedur wird solange durchgeführt bis eine zulässige Lösung in der Nachbarschaft von $\boldsymbol{x}$ gefunden wurde.

Falls unzulässige Lösungen nicht verboten sind, werden sie über die Zielfunktion (3-6) entsprechend bestraft und es wird mit ihnen wie mit zulässigen Lösungen verfahren.

TABU SUCHE

Bei der Tabu Suche müssen zusätzlich die Verwaltung der Tabuliste und das Aspirationskriterium festgelegt werden. In jedem Schritt werden in der Tabuliste die

geänderten Komponenten eingetragen. Für eine vorgegebene Anzahl von tt Iterationen ist es dann verboten diese Komponentenänderungen wieder rückgängig zumachen. Wenn eine Änderung tt Iterationen in der Tabuliste war, wird sie gelöscht und die entsprechenden Komponenten können wieder geändert werden.

Ergibt jedoch eine Änderung einer Komponente aus der Tabuliste einen besseren Zielfunktionswert als den aktuell Besten, dann wird der Tabustatus von dieser Komponente übergangen und die Änderung wird durchgeführt. Das ist das Aspirationskriterium der Tabu Suche.

GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

Bei der GRASP müssen die Geedyfunktion, die die Qualität jeder Lösungskomponente bewertet, die restringierte Kandidatenliste, aus der zufällig ein Lösungskandidat ausgewählt wird und die lokale Suche, zur Verbesserung des Lösungskandidaten, definiert werden.

Die Greedyfunktion ordnet jeder Lösungskomponente x_i seinen entsprechenden Koeffizienten c_i der Zielfunktion zu

$$g(x_i) = c_i \quad \forall\, i = \{1, \dots, l\}.$$

Gestartet wird die Konstruktionsphase mit $\boldsymbol{x} = \mathbf{0}$. In jedem Schritt wird dann eine Lösungskomponente ausgewählt, die auf eins gesetzt wird. Dazu werden alle Lösungskomponenten, die noch null sind bei denen aber eine Änderung auf eins zulässig ist, mit der Greedyfunktion bewertet. Hierbei ist g_{max} die maximale Bewertung der Lösungskomponenten in der aktuellen Iteration und g_{min} die minimale Bewertung. Alle Komponenten deren Greedyfunktionswert die Bedingung

$$g(x_j) \geq g_{max} - \alpha(g_{max} - g_{min}) \quad \text{mit} \quad \alpha \in [0,1]$$

erfüllt, werden in die restringierte Kandidatenliste aufgenommen, aus der im Anschluss zufällig eine Komponente ausgewählt wird. Dieses Verfahren wird solange durchgeführt bis entweder keine zulässige Lösung mehr erreicht werden kann, wenn eine weitere Lösungskomponente auf eins gesetzt werden würde oder wenn die zu Beginn zufällig bestimme Anzahl Iterationen erreicht wurde.

In der anschließenden lokalen Suchphase wird in jeder Iteration entweder mit der ein- oder zweikomponentigen Änderung der erstbeste oder der beste Nachbar gesucht. Die lokale Suche

wird abgebrochen, wenn kein besserer Nachbar als die aktuell beste Lösung gefunden werden kann, d.h. ein lokales Optimum wurde erreicht.

EVOLUTIONÄRE ALGORITHMEN

Als evolutionärer Algorithmus ist ein genetischer Algorithmus implementiert worden. Dazu müssen die Fitnessfunktion und die genetischen Operationen auf das Problem der Ermittlung der OFR angepasst werden.

Für die Fitnessfunktion wird

$$fit(\boldsymbol{x}) = \boldsymbol{cx} + 1$$

verwendet. Durch die Addition von eins ist die Wahrscheinlichkeit immer größer null und in (3-1) wird die Division durch null unterbunden, wenn die Rouletterad-Selektion verwendet wird. Bei dieser Fitnessfunktion können nur zulässige Lösungen (Chromosome) bewertet werden. Das bedeutet, dass die Chromosome nach Rekombination und Mutation repariert werden müssen, da es bei diesen beiden Operationen zu unzulässigen Lösungen kommen kann.

Wenn die Wettkampfselektion eingesetzt wird, kann auch die Funktion in (3-6) als Fitnessfunktion verwendet werden. Da für diese Selektionsart keine fitnessproportionalen Wahrscheinlichkeiten berechnet werden müssen und somit auch Minimierungsprobleme betrachtet werden können.

AMEISENALGORITHMEN

Beim Ameisenalgorithmus konstruieren zunächst m Ameisen ihre Lösung. Jede von ihnen startet mit $\boldsymbol{x} = \boldsymbol{0}$ und wählt pro Schritt auf Basis der heuristischen Information und der Pheromonspuren zufällig eine Transition zur Feuerung aus, d.h. $x_i = 1, i \in \{1, \dots, l\}$. Dies entspricht der einkomponentigen Änderung in (3-4). Hierbei werden die Pheromonspuren und die heuristischen Informationen den Transitionen zugeordnet. Die heuristische Information ergibt sich aus dem Koeffizienten der Zielfunktion

$$\eta_i = c_i + 1, \qquad i = 1, .., l.$$

Die Addition von eins verhindert die Auswahlwahrscheinlichkeit null und zudem die Division durch null in (3-2). Für die Pheromonablage wurden die bereits vorgestellten Prinzipien des MAX-MIN Ant System implementiert mit der Bewertungsfunktion

$$g(\boldsymbol{x}) = \frac{1}{1 + (z_{LP} - z_b)},$$

wobei z_{LP} der Zielfunktionswert des relaxierten Problems in (3-7) ist und z_b ist entweder der Zielfunktionswert der iterationsbesten oder der aktuell besten Ameise, je nachdem welche Ameise zur Pheromonablage ausgewählt wird.

4 DIE PNMAT

Die in Kapitel 2 und 3 vorgestellten Konzepte wurden in MATLAB implementiert, um die Modellierung, Simulation und Optimierung mit Petri-Netzen zu ermöglichen. Die entwickelten Definitionen und Algorithmen dienen hierbei als Spezifikation für das Petri-Netz-Simulationstool, das *PNmat* genannt wird. Die Entwicklung der *PNmat* wurde mit der MATLAB-Version R2012b durchgeführt.

4.1 MATRIXSCHREIBWEISE

MATLAB ist eine kommerzielle Software zur Lösung mathematischer Probleme und zur Darstellung der Ergebnisse. MATLAB ist primär für Berechnungen mit Matrizen ausgelegt. Dies leitet sich auch aus dem Namen MATrix LABoratory ab. Aus diesem Grund müssen die Definitionen und Algorithmen aus Kapitel 2, die mithilfe von Mengen dargestellt werden, in Matrizen überführt werden. Diese Transformation wurde bereits in Kapitel 2.5 erläutert und Tabelle 4.1 fasst die Ergebnisse zusammen.

Tabelle 4.1: Darstellung der Petri-Netz-Elemente mithilfe von Matrizen

Petri-Netz-Element	Kapitel 2	Matrixschreibweise
Kanten und Gewichte von Plätzen zu Transitionen	$F \subseteq (P \times T)$ $f: (F \cup G, m) \to \mathbb{N}_0$	$\mathcal{F} = \begin{pmatrix} f_{1,1} & \cdots & f_{1,n_t} \\ \vdots & \ddots & \vdots \\ f_{n_p,1} & \cdots & f_{n_p,n_t} \end{pmatrix}$ $f_{i,j} = \begin{cases} f(p_i \to t_j), & (p_i \to t_j) \in \mathcal{F} \\ 0, & (p_i \to t_j) \notin \mathcal{F} \end{cases}$
Kanten und Gewichte von Transitionen zu Plätzen	$G \subseteq (T \times P)$ $f: (F \cup G, m) \to \mathbb{N}_0$	$\mathcal{G} = \begin{pmatrix} g_{1,1} & \cdots & g_{1,n_t} \\ \vdots & \ddots & \vdots \\ g_{n_p,1} & \cdots & g_{n_p,n_t} \end{pmatrix}$ $g_{i,j} = \begin{cases} f(t_j \to p_i), & (t_j \to p_i) \in \mathcal{G} \\ 0, & (t_j \to p_i) \notin \mathcal{G} \end{cases}$
Markierung des Petri-Netzes	$m: P \to \mathbb{N}_0$	$\boldsymbol{m} = \left(m(p_1), m(p_2), \dots, m\left(p_{n_p}\right)\right)^\top$ $= \left(m_1, m_2, \dots, m_{n_p}\right)^\top$

Lokale Lösungstypen	$e: P \to \{1,2,3\}$	$\boldsymbol{e} = \left(e(p_1), e(p_2), \dots, e\left(p_{n_p}\right)\right)^\top = \left(e_1, e_2, \dots, e_{n_p}\right)^\top$
Globaler Lösungstyp	$e \to \{1,2,3\}$	$e \in \{1,2,3\}$
Bewertung der Kanten von Plätzen zu Transitionen	$\mathcal{p}_{\text{out}}: F \to \{[0,1]: e(p_i) = 1, \mathbb{R}_{\geq 0}: e(p_i) = 2 \vee e(p_i) = 3\}$	$\mathcal{P}_{out} = \begin{pmatrix} \mathcal{p}_{1,1} & \cdots & \mathcal{p}_{1,n_t} \\ \vdots & \ddots & \vdots \\ \mathcal{p}_{n_p,1} & \cdots & \mathcal{p}_{n_p,n_t} \end{pmatrix}$ $\mathcal{p}_{i,j} = \begin{cases} \mathcal{p}_{\text{out}}(p_i \to t_j), & (p_i \to t_j) \in \mathcal{F} \\ -1, & (p_i \to t_j) \notin \mathcal{F} \end{cases}$
Bewertung der Kanten von Transitionen zu Plätzen	$\mathcal{p}_{\text{in}}: G \to \{[0,1]: e(p_i) = 1, \mathbb{R}_{\geq 0}: e(p_i) = 2 \vee e(p_i) = 3\}$	$\mathcal{P}_{in} = \begin{pmatrix} \mathcal{p}_{1,1} & \cdots & \mathcal{p}_{1,n_t} \\ \vdots & \ddots & \vdots \\ \mathcal{p}_{n_p,1} & \cdots & \mathcal{p}_{n_p,n_t} \end{pmatrix}$ $\mathcal{p}_{i,j} = \begin{cases} \mathcal{p}_{\text{in}}(t_j \to p_i), & (t_j \to p_i) \in \mathcal{G} \\ -1, & (t_j \to p_i) \notin \mathcal{G} \end{cases}$
Bewertung der Transitionen	$\mathcal{w}: T \to \{[0,1]: e = 1, \mathbb{R}_{\geq 0}: e = 2 \vee e = 3\}$	$\boldsymbol{\mathcal{w}} = \left(\mathcal{w}(t_1), \dots, \mathcal{w}\left(t_{n_t}\right)\right)^\top = \left(\omega_1, \dots, \omega_{n_t}\right)^\top$
Minimale Kapazitäten	$c_l: P \to \mathbb{N}_0$	$\boldsymbol{c_l} = \left(c_l(p_1), c_l(p_2), \dots, c_l\left(p_{n_p}\right)\right)^\top = \left(c_{l_1}, c_{l_2}, \dots, c_{l_{n_p}}\right)^\top$
Maximale Kapazitäten	$c_u: P \to \mathbb{N}_0$	$\boldsymbol{c_u} = \left(c_u(p_1), c_u(p_2), \dots, c_u\left(p_{n_p}\right)\right)^\top = \left(c_{u_1}, c_{u_2}, \dots, c_{u_{n_p}}\right)^\top$
Testkanten	$\mathcal{T} \subseteq (P \times T)$ $f: (F \cup G \cup \mathcal{T} \cup \mathcal{I}, m) \to \mathbb{N}_0$	$\mathcal{T} = \begin{pmatrix} t_{1,1} & \cdots & t_{1,n_t} \\ \vdots & \ddots & \vdots \\ t_{n_p,1} & \cdots & t_{n_p,n_t} \end{pmatrix}$ $t_{i,j} = \begin{cases} f(p_i \to t_j), & (p_i \to t_j) \in \mathcal{T} \\ -1, & (p_i \to t_j) \notin \mathcal{T} \end{cases}$
Hemmkanten	$\mathcal{I} \subseteq (P \times T)$ $f: (F \cup G \cup \mathcal{T} \cup \mathcal{I}, m) \to \mathbb{N}_0$	$\mathcal{I} = \begin{pmatrix} i_{1,1} & \cdots & i_{1,n_t} \\ \vdots & \ddots & \vdots \\ i_{n_p,1} & \cdots & i_{n_p,n_t} \end{pmatrix}$ $i_{i,j} = \begin{cases} f(p_i \to t_j), & (p_i \to t_j) \in \mathcal{I} \\ -1, & (p_i \to t_j) \notin \mathcal{I} \end{cases}$
Heiße und kalte Transitionen	$T_h \subseteq T$ $T_c \subseteq T$	$\boldsymbol{\hbar} = \left(x_1, x_2, \dots, x_{n_t}\right)^\top$ $x_j = \begin{cases} 1, & t_j \in T_h \\ 2, & t_j \in T_c \end{cases}, \quad j = 1, \dots, n_t$

Feuerungsstrategie	$(F1)$ Alle freigeschalteten Transitionen feuern. $(F2)$ Eine zufällig ausgewählte Transition der freigeschalteten Transitionen feuert. $(F3)$ Es gibt heiße und kalte Transitionen. Die heißen Transitionen feuern und bei den kalten Transitionen kann der Benutzer entscheiden, ob sie feuern oder nicht.	$s \in \{1,2,3\}$
Konfliktlösung	$(K1)$ Lokale Konfliktlösung $(K2)$ Globale Konfliktlösung	$k \in \{1,2\}$
Bewertungsart	$(B1)$ Kantenbewertetes Petri-Netz $(B2)$ Transitionenbewertetes Petri-Netz	$b \in \{1,2\}$
Aktive Transitionen	$TA \subseteq T$	$\boldsymbol{a} = (x_1, x_2, \dots, x_{n_t})^\top$ $x_j = \begin{cases} 0, & t_j \notin TA \\ 1, & t_j \in TA \end{cases}$
Freigeschaltete Transitionen	$TE \subseteq TA$	$\boldsymbol{e} = (x_1, x_2, \dots, x_{n_t})^\top$ $x_j = \begin{cases} 0, & t_j \notin TE \\ 1, & t_j \in TE \end{cases}$
Feuernde Transitionen	$TF \subseteq TE$	$\boldsymbol{f} = (x_1, x_2, \dots, x_{n_t})^\top$ $x_j = \begin{cases} 0, & t_j \notin TF \\ 1, & t_j \in TF \end{cases}$
Aktive Transitionen, die bereits von ihren Inputplätzen freigeschaltet worden sind	$TAe_{in} \subseteq TA$	$\boldsymbol{ae} = (x_1, x_2, \dots, x_{n_t})^\top$ $x_j = \begin{cases} 0, & t_j \notin TAe_{in} \\ 1, & t_j \in TAe_{in} \end{cases}$
Nutzen der Transitionen bei der Optimierung	$w: T \to \mathbb{R}_{\geq 0}$	$\boldsymbol{w} = \left(w(t_1), \dots, w(t_{n_t})\right)^\top = (\omega_1, \dots, \omega_{n_t})^\top$

4.2 IMPLEMENTIERUNG DER PROZESSE IN MATLAB

Mithilfe der Matrixschreibeweise aus Tabelle 4.1 können die in Kapitel 2 aufgeführten Prozesse – Aktivierung, Konfliktlösung/Freischaltung, Feuerungsauswahl, Feuerung und Aktualisierung des Petri-Netzes – in MATLAB implementiert werden. Abbildung 4.1 stellt die Abfolge der einzelnen Prozesse während der Petri-Netz-Simulation dar. Jeder Prozess wurde mithilfe von Funktionen in MATLAB umgesetzt.

Zunächst wird das Petri-Netz durch die in Tabelle 4.1 vorgestellten Matrizen und Vektoren definiert. Ausgehend von dieser Struktur des Petri-Netzes werden die aktiven Transitionen bestimmt. Anschließend werden mögliche Konflikte gelöst und die entsprechenden Transitionen freigeschaltet. Unter diesen werden je nach Auswahl der Feuerungsstrategie die feuernden Transitionen bestimmt. Diese werden gefeuert, die neue Markierung wird berechnet und die zugrundeliegenden Matrizen und Vektoren werden aktualisiert.

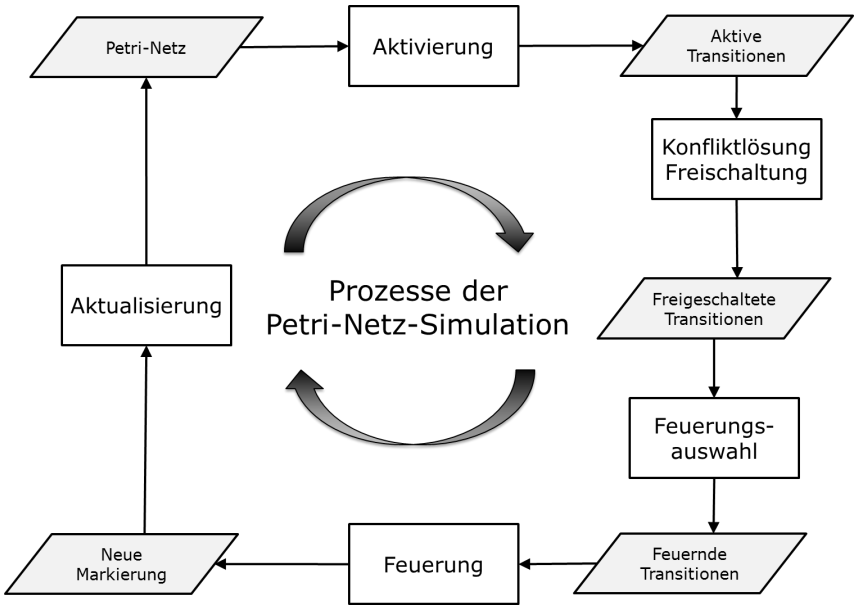


Abbildung 4.1: Abfolge der Prozesse bei der Petri-Netz-Simulation

Nachfolgend werden bedeutende Auszüge aus dem Programmiercode vorgestellt.[1] Alle m-files der PNmat und deren Aufgabe sind in A-Tabelle 2 im Anhang zusammengefasst.

[1] Ausgelassene Code-Abschnitte werden durch drei Punkte symbolisiert.

4.2.1 AKTIVIERUNG

Bei der Aktivierung müssen die Bedingungen aus Definition 2.21 erfüllt sein. Dies wurde auf folgende Weise mithilfe der Matrizenschreibweise in MATLAB umgesetzt:

```
M  = repmat(m,1,n_t);                          %Markierung
Cl = repmat(cl,1,n_t);                         %minimale Kapazität
Cu = repmat(cu,1,n_t);                         %maximale Kapazität

TA = sum(M-F>=Cl)'==repmat(n_p,n_t,1) &        %1.
     sum(M+G<=Cu)'==repmat(n_p,n_t,1) &        %2.
     sum(M<I)'==repmat(n_p,n_t,1) &            %3.
     sum(M>T)'==repmat(n_p,n_t,1);             %4.
```

Der MATLAB-Befehl `repmat` bewirkt die Wiederholung der jeweiligen Vektoren, so dass jeweils eine Matrix der Größe $(n_p \times n_t)$ entsteht.

Eine Transition kann nur aktiv werden, wenn folgende Bedingungen erfüllt sind:

1. Alle Inputplätze, die über eine „normale“ Kante mit der Transition verbunden sind, haben eine Markierung, die bei Verringerung um das Kantengewicht immer noch größer oder gleich der minimalen Kapazität ist **und**
2. alle Outputplätze haben eine Markierung, die bei Erhöhung um das Kantengewicht immer noch kleiner oder gleich der maximalen Kapazität ist **und**
3. alle Inputplätze, die über eine Hemmkante mit der Transition verbunden sind, haben eine Markierung, die kleiner als das Kantengewicht ist **und**
4. alle Inputplätze, die über eine Testkante mit der Transition verbunden sind, haben eine Markierung, die größer als das Kantengewicht ist.

Der Vektor `TA` enthält dann an der Position j den Eintrag 1, wenn die j-te Transition aktiv ist, und 0, wenn nicht (siehe Tabelle 4.1). Der Aktivierungsprozess wurde in der MATLAB-Funktion `activation.m` umgesetzt.

4.2.2 KONFLIKTLÖSUNG UND FREISCHALTUNG

Bei der Konfliktlösung und der damit verbundenen Freischaltung ausgewählter aktiver Transitionen wird zunächst geprüft, ob ein Input- bzw. Outputkonflikt nach Definition 2.16 vorliegt.

```
K=m-F*TA<cl;              %Outputkonflikt?
K=m+G*TAein>cu;           %Inputkonflikt?
```

Im Konfliktfall wird die Freischaltung der Transitionen entweder lokal, wie in Algorithmus 2.6 beschrieben, oder global durchgeführt. Hierbei stehen verschiedene Bewertungsmethoden zur Verfügung – Wahrscheinlichkeiten, Nutzen und Nutzenquotient – die entweder an die Kanten oder an die Transitionen geschrieben werden können. Die in Algorithmus 2.2 und Algorithmus 2.3 vorgestellten Abläufe zur lokalen Konfliktlösung und die in Algorithmus 2.4 und Algorithmus 2.5 beschriebenen Abläufe zur globalen Konfliktlösung wurden entsprechend in MATLAB umgesetzt. Der Freischaltungsprozess wird durch die MATLAB-Funktion `enabling.m` ausgeführt.

4.2.3 FEUERUNGSAUSWAHL

Für die Auswahl der zu feuernden Transitionen wurden die drei in Kapitel 2.3 vorgeschlagenen Strategien umgesetzt. Bei der Strategie $(F3)$ entscheidet der Benutzer, welche der kalten Transitionen gefeuert werden. In der MATLAB-Funktion `allCombs.m` wird die Oberfläche graphisch aufbereitet, so dass der Benutzer die kalten Transitionen, die feuern sollen, anklicken kann (siehe Kapitel 4.4.5).

```
if fm==1            %alle freigeschalteten Transitionen feuern
   TF = TE;
elseif fm==2        %nur eine zufällig ausgewählte Transition feuert
   Z = rand(size(TE));
   [c,idx] = max(Z.*TE);
   TF(idx) = 1;
elseif fm==3        %heiße Transitionen feuern und bei den kalten
                    %kann der Benutzer entscheiden
   TF = allCombs(…);
end
```

Die Feuerungsauswahl wurde in der MATLAB-Funktion `firingSelection.m` umgesetzt.

4.2.4 FEUERUNG UND AKTUALISIERUNG

In der MATLAB-Funktion `firingProcess.m` wird die neue Markierung des Petri-Netzes nach Feuerung der ausgewählten Transitionen berechnet (vgl. Definition 2.13). In der Matrixschreibweise ergibt sich folgende „einfache“ Gleichung:

```
m_new = m+(G-F)*TF;
```

Hierbei enthält der Vektor `TF` an den Stelle j den Eintrag 1, wenn die Transition t_j feuert, und den Eintrag 0, wenn sie nicht feuert.

4.2.5 PETRI-NETZ-SIMULATION

Gestartet wird die Simulation mit MATLAB-Funktion `PNsim.m`. Nach dem Aufruf dieser Funktion hat man die Möglichkeit, ein bestehendes Petri-Netz zu laden oder mithilfe eines Assistenten ein neues Petri-Netz zu erstellen. In Kapitel 4.4 wird die Erstellung eines Petri-Netzes näher erläutert. Erstellte Petri-Netze werden immer in einer **MATLAB-Struktur** gespeichert. MATLAB-Strukturen fassen Daten unterschiedlichen Typs zu einer Einheit zusammen. Die Komponenten einer Struktur werden durch ihre Daten gebildet und haben einen Namen. Auf die Komponenten kann über ihre Namen mithilfe des Punkt-Operators zugegriffen werden. Die Struktur eines Petri-Netzes hat immer den Namen `PN` und enthält die in A-Tabelle 1 (siehe Anhang) zusammengefassten Komponenten.

Die Daten der Petri-Netz-Struktur werden in der Funktion `draw_pn.m` so aufbereitet, dass sie als Verbindungsmatrix an die modifizierte `biograph`-Funktion der Bioinformatics-Toolbox übergeben werden können. Die `biograph`-Funktion sorgt für die graphische Darstellung des Petri-Netzes und wird in Kapitel 4.3 erläutert.

Anschließend werden in der Funktion `sim_PN.m` die zuvor beschriebenen Prozesse der Reihe nach aufgerufen, bis der Benutzer den in Abbildung 4.1 dargestellten Simulationskreislauf beendet. Die Funktion ruft sich selbest wieder auf, wenn der Benutzer nach der Abarbeitung eines Prozesses auf einen Button klickt, und geht dabei in den nächsten Prozessschritt über. Nachfolgend ist eine verkürzte Version dieser Funktion dargestellt. Die Prozesse 5 und 6 werden nur bei der Optimierung bzw. bei der Animation aufgerufen, sie sind kein Bestandteil des eigentlichen Simulationskreislaufs.

```
function g = sim_PN(...)
  global PNbio          %Biograph des Petri-Netzes zur graphischen
                        %Darstellung
  global PN             %Struktur des Petri-Netzes
  setParam;             %Parameter setzen zur Verkürzung der
                        %Schreibweise (PN. kann weggelassen werden)
  if process==1         %Darstellung des Petri-Netzes nach dem Laden
                        %oder dem Erstellen
```

```
    ...
    dolayout(PNbio)            %Berechnet die Knotenpositionen und
                               %die Kantentrajektorien
    g=biograph.bggui(PNbio);   %Konstruktor, Darstellung des Petri-
                               %Netzes als Biograph
    ...
    uicontrol(...,'Callback',{@sim_PN, ...});  %Selbstaufruf in
                                               %Prozess 2
    uicontrol(...,'Callback','close');  %Ende der Simulation
    setMenu(...);              %Menüleiste der Oberfläche festlegen
elseif process==2    %Aktivierung
    TA = activation(...);      %Berechnung der aktiven Transitionen
    ...
    set(PNbio.nodes(find([zeros(n_p,1);TA]>0)),
    'Color',[1 1 0],'TextColor',[0 0 0]);   %Farbe der aktiven
                                    %Transitionen auf Gelb setzen
    dolayout(PNbio)            %siehe oben
    g=biograph.bggui(PNbio);   %siehe oben
    ...
    uicontrol(...,'Callback',{@sim_PN, ...});  %Selbstaufruf in
                                               %Prozess 3
    uicontrol(...,'Callback', {@plotResults, ...});   %Ende der
                        %Simulation und Darstellung der Ergebnisse
    setMenu(...);              %siehe oben
elseif process==3    %Freischaltung und Feuerungsauswahl
    TE = enabling(...);        %Freischaltung
    ...
    set(PNbio.nodes(find([zeros(n_p,1);TE]>0)),
    'Color',[0 1 0],'TextColor',[0 0 0]);   %Farbe der frei-
                       %geschalteten Transitionen auf Grün setzen
    dolayout(PNbio)            %siehe oben
    g=biograph.bggui(PNbio);   %siehe oben
    setMenu(...);              %siehe oben
    ...
    uicontrol(...,'Callback', 'uiresume(gcbf)')%Nach Auswahl der
                         %feuernden Transition weiter in Prozess 4
    TF=firingSelection(...);   %je nach Auswahl der
                            %Feuerungsstrategie, werden die
                            %feuernden Transitionen bestimmt
elseif process==4    %Feuerung
    m=firingProcess(...);      %Berechnung der neuen Markierung
    benefit=calBenefit(...);   %Berechnung des Nutzens bei
                       %Bewertung durch Nutzen oder Nutzenquotienten
    ...
    set(PNbio.nodes(find([zeros(n_p,1);TT]>0)),
    'Color',[1 0 0],'TextColor',[0 0 0]);   %Farbe der gefeuerten
                                    %Transitionen auf Rot setzen
    for i=1:n_p                %Tokenanzahl in den Plätzen
                            %aktualisieren (graphische Darstellung)
        set(PNbio.nodes(i),'Label',num2str(m(i)));
    end
    updateWeights(...);        %Kantengewichte aktualisieren
    [M,st,TFmat,B]=saveMarking(...); %Speichert die Informationen
        %des Simulationsschritts für die abschließende Darstellung
    dolayout(PNbio)            %siehe oben
    g=biograph.bggui(PNbio);   %siehe oben
    ...
```

```
    uicontrol(...,'Callback',{@sim_PN, ...});  %Selbstaufruf in
                                               %Prozess 2
    uicontrol(...,'Callback', {@plotResults, ...});   %siehe oben
    setMenu(...);              %siehe oben
  elseif process==5    %Optimierung
    dolayout(PNbio)            %siehe oben
    g=biograph.bggui(PNbio);   %siehe oben
    ...
    uicontrol(...,'Callback', 'uiresume(gcbf)')%Start der
                                               %Optimierung
    uicontrol(...,'Callback','close');   %Ende der Optimierung
    setMenu(...);              %siehe oben
  elseif process==6    %Animation
    ...
    for i=1:n_p                %Tokenanzahl in den Plätzen
                           %aktualisieren (graphische Darstellung)
        set(PNbio.nodes(i),'Label',num2str(m(i)));
    end
    updateWeights(...);        %Kantengewichte aktualisieren
    dolayout(PNbio)            %siehe oben
    g=biograph.bggui(PNbio);   %siehe oben
    ...
    uicontrol(...,'Callback',{@sim_PN, ...});  %Selbstaufruf in
                                               %Prozess 2
    uicontrol(...,'Callback', {@plotResults, ...});  %Ende der
                        %Animation und Darstellung der Ergebnisse
    setMenu(...);              %siehe oben
  end
end
```

4.2.6 PETRI-NETZ-ANIMATION

In der Simulation werden schrittweise die einzelnen Prozesse durchlaufen und graphisch dargestellt (siehe Kapitel 4.4.5). In der Animationsoption wird entweder die ganze Simulation als Video aufgezeichnet oder es wird nur das Petri-Netz nach einer eingegebenen Anzahl an Simulationsschritten dargestellt und nicht jeder einzelne Schritt.

Bei der Videooption wird zunächst mithilfe des MATLAB-Befehls `avifile` eine AVI-Datei erstellt. Mit dem Befehl `hardcopy` wird das auf dem Bildschirm nicht sichtbare Petri-Netz-Fenster in einer Bilddatei gespeichert, die anschließend mit dem Befehl `im2frame` in ein Einzelbild für das Video (movie frame) umgewandelt wird. Mithilfe des MATLAB-Befehls `addframe` wird dieses Einzelbild zu dem Video hinzugefügt.

```
aviobj = avifile(filemov,'fps',fps1);  %Erstellt eine neue AVI-Datei
for i=1:n_steps                        %n_steps ist die Anzahl der
                                       %Simulationsschritte
    TA = activation(...);              %Aktivierung
```

```
    ...
    g=biograph.bggui(PNbio);          %Darstellung des Petri-Netzes
                                  %(nicht sichtbar auf dem Bildschirm)
    img = hardcopy(g.hgFigure, '-dzbuffer', '-r0');    %speichert
                                      %das Bild in einer Datei ab
    aviobj = addframe(aviobj, im2frame(img));   %Fügt das Bild zur
                                                %AVI-Datei hinzu
    ...
    TE = enabling(PN,m,F,G,TA);       %Freischaltung
    ...
    g=biograph.bggui(PNbio);          %siehe oben
    img = hardcopy(g.hgFigure, '-dzbuffer', '-r0');%siehe oben
    aviobj = addframe(aviobj, im2frame(img));   %siehe oben
    m=firingProcess(m,F,G,TF);        %Feuerung
    setParam;                         %Parameter neu setzen
    benefit=calBenefit(PN,TF,m);
    ...
    g=biograph.bggui(PNbio);          %siehe oben
    img = hardcopy(g.hgFigure, '-dzbuffer', '-r0');%siehe oben
    aviobj = addframe(aviobj, im2frame(img));   %siehe oben
end
aviobj = close(aviobj);            %AVI-Datei schließen
...
```

4.2.7 OPTIMIERUNG DER FEUERUNGSREIHENFOLGE

Für die Lösung des in Kapitel 3 vorgestellten Optimierungsproblems zur Ermittlung der optimalen Feuerungsreihenfolge (OFR) wurden die vorgestellten Näherungsverfahren mit den problemspezifischen Anpassungen als MATLAB-Funktionen implementiert. Bei dem exakten Branch-and-Bound-Verfahren wurde auf die bereits in MATLAB intergierte Funktion `bintprog()` zur Lösung binärer Optimierungsprobleme zurückgegriffen. Dazu muss das Problem, wie in Kapitel 3.4 erläutert, mithilfe von Matrizen und Vektoren formuliert werden. Die Vektoren $\boldsymbol{c}$ und $\boldsymbol{b}$ sowie die Matrix A werden der MATLAB-Funktion `bintprog()` übergeben. Sie werden in MATLAB wie folgt aufgebaut:

```
c=repmat(-c,n_steps,1);   %Aufbau des Vektors c (Zielkoeffizienten)
b=[repmat(-(cl-m0),n_steps,1);repmat(cu-m0,n_steps,1)];   %Aufbau
                %des Vektors b (rechte Seite der Nebenbedingungen)
%Aufbau der Matrix A (Nebenbedingungen)
A=zeros(2*n_steps*n_p,n_steps*n_t);
A(1:n_p,1:n_t)=F;
A(n_steps*n_p+1:n_steps*n_p+n_p,1:n_t)=G;



for i=2:n_steps
   A((i-1)*n_p+1:i*n_p,1:(i-1)*n_t)=repmat(-(G-F),1,i-1);
   A((i-1)*n_p+1:i*n_p,(i-1)*n_t+1:i*n_t)=F;
```

```
   A(n_steps*n_p+(i-1)*n_p+1:n_steps*n_p+(i)*n_p,1:
                                        (i-1)*n_t)=repmat(G-F,1,i-1);
   A(n_steps*n_p+(i-1)*n_p+1:n_steps*n_p+(i)*n_p,
                                        (i-1)*n_t+1:i*n_t)=G;
end

%---Nur für serielles Feuern---
A=[A;zeros(n_s,n_s*n_t)];
for i=1:n_s
   A(2*n_s*n_p+i,(i-1)*n_t+1:i*n_t)=ones(1,n_t);
end
b=[b;ones(n_s,1)];
%-----------------------------

%---Nur für Erreichen einer Zielmarkierung (m>=mz)---
A=[A;repmat(-H,1,n_s)];
b=[b;m0-mz];
%-------------------------------------------------

%Löschen der Zeilen für Plätze ohne obere Kapazität
idx=find(b==Inf);
A(idx,:)=[];
b(idx)=[];

[Tfall,benefit,exitflag,output] = bintprog(f,A,b)   %Optimierung
```

Die Funktionen für die Näherungsverfahren sind auch so aufgebaut worden, dass die Vektoren ***c*** und ***b*** sowie die Matrix *A* übergeben werden müssen.

4.3 GRAPHISCHE OBERFLÄCHE

Als Grundlage für die graphische Oberfläche der *PNmat* wurde der **Biograph** der Bioinformatics-Toolbox von MATLAB herangezogen (MathWorks 2012a). Die Objekt-Konstruktor-Funktion `Biograph` erstellt durch eine vorgegebene Verbindungsmatrix ein Biograph-Objekt, das graphisch in Form eines gerichteten Graphen dargestellt werden kann. Alle positiven Einträge der Verbindungsmatrix, die nicht auf der Diagonalen liegen, kennzeichnen verbundene Knoten dieses Graphen. Zudem stehen zahlreiche Methoden zur Verfügung, um beispielsweise die Positionen der Knoten zu berechnen (`dolayout`), den Graph zu zeichnen (`view`) oder um Eigenschaften der Knoten und Kanten zu erhalten (`getnodesbyid` und `getedgesbyid`). Viele Eigenschaften des Biographen lassen sich ändern. Zum Beispiel können Form, Farbe und Größe der Knoten verändert werden.

Der Biograph für das Petri-Netz wird in dem MATLAB-file `draw_PN.m` erstellt. In dieser Funktion werden die benutzerdefinierten Angaben über das Petri-Netz verarbeitet und als Verbindungsmatrix der `Biograph`-Funktion übergeben.

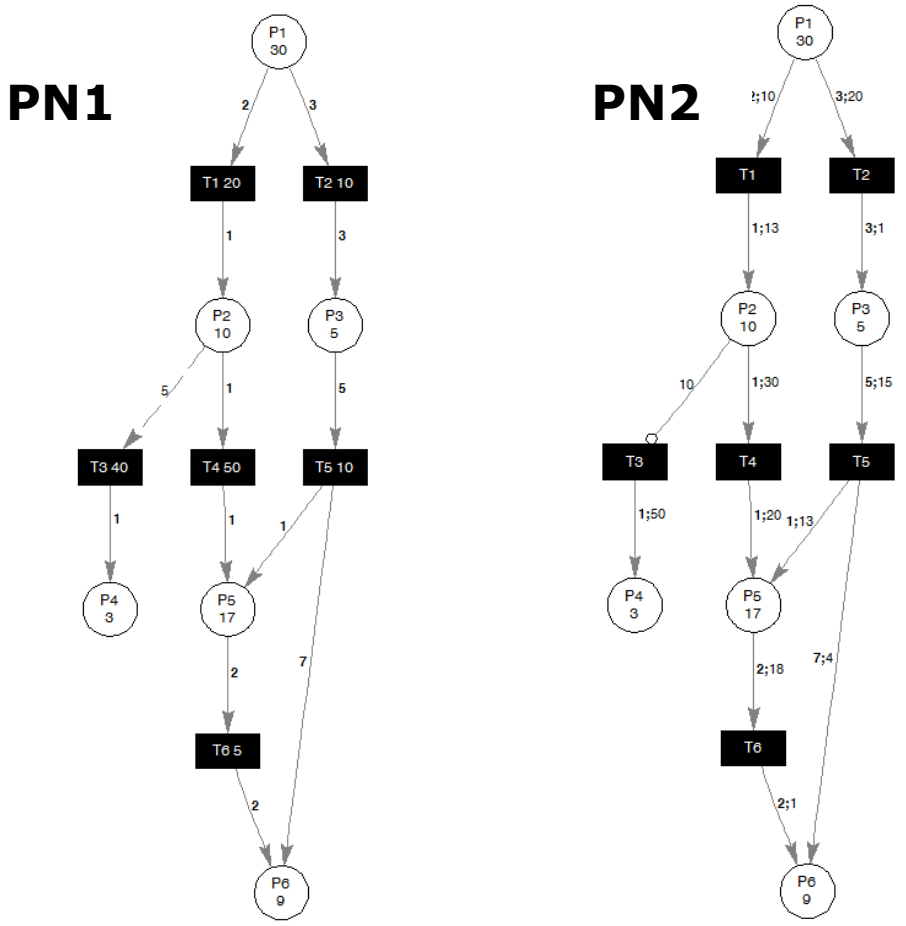


Abbildung 4.2: Darstellung eines Petri-Netzes mithilfe von modifizierten Biographen; PN1: Petri-Netz mit Transitionenbewertungen und Testkante; PN2: Petri-Netz mit Kantenbewertungen und Hemmkante

Der Code des Biographen musste an einigen Stellen angepasst werden, um dem Petri-Netz sein charakteristisches Aussehen zu geben und alle wesentlichen Information graphisch darzustellen. Nachfolgend sind die wesentlichen Änderungen an den m-files des ursprünglichen Biographen aufgelistet, wobei jeweils in Klammern das m-file vermerkt wurde, das die Änderungen enthält:

– Knotenbeschriftungen: der Name des Petri-Netz-Elements steht im Knoten und zusätzlich bei Plätzen die aktuelle Tokenanzahl und bei Transitionen die Bewertung, falls das Petri-Netz transitionenbewertet ist (`@biograph\@node\hgUpdate.m`).

- Bewertung der Kanten: zusätzlich zu dem Kantengewicht ist die Bewertung an der Kante vermerkt, falls das Petri-Netz kantenbewertet ist. Das Kantengewicht ist fett gedruckt und die Bewertung folgt nach einem Semikolon (`@biograph\@edge\hgUpdate.m`).
- Test- und Hemmkanten: Testkanten werden als Pfeil mit einer gestrichelten Linie gekennzeichnet und Hemmkanten haben eine Kreisendung (`@biograph\@edge\hgUpdate.m`)
- Figure: Größe, Position, Titel, Sichtbarkeit usw. des Fensters, auf dem das Petri-Netz dargestellt wird, wurden angepasst (`@biograph\@bggui\bggui.m`)

Zwei Beispiele für die Darstellung eines Petri-Netzes mit der *PNmat* sind in Abbildung 4.2 abgebildet. Das Petri-Netz *PN1* zeigt ein transitionenbewertetes Petri-Netz mit einer Testkante und *PN2* ist ein Beispiel für ein kantenbewertetes Petri-Netz mit einer Hemmkante.

4.4 BENUTZUNGSHINWEISE (USER GUIDE)

In diesem Abschnitt wird die Benutzung der *PNmat* schrittweise erläutert. Es wird die Installation sowie die Erstellung, Simulation, Optimierung und Animation eines Petri-Netzes schrittweise am Beispiel des Petri-Netzes *PN1* aus Abbildung 4.2 dargestellt.

4.4.1 INSTALLATION

Zur leichteren Bedienbarkeit wurden die m-files der *PNmat* in einem App zusammengefasst, so dass nur diese App installiert werden muss und nicht die zahlreichen m-files. Zur Installation dieser App geht man in MATLAB auf die Registerkarte *APPS*, klickt auf *Install App* und wählt die Datei *PNmat.mlappinstall* aus. Anschließend erscheint das in Abbildung 4.3 markierte Icon in der App-Liste.

Abbildung 4.3: Installation der PNmat

4.4.2 PETRI-NETZE ERSTELLEN

Ein Wizard hilft bei der schrittweisen Erstellung von Petri-Netzen. Dazu startet man das App *PNmat*. Es erscheint der folgende Auswahldialog und die Auswahl *create* startet den Wizard.

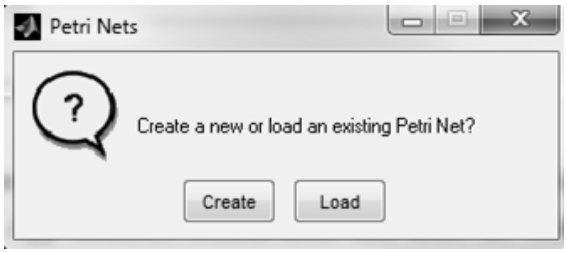


Abbildung 4.4: Auswahldialog beim Start der PNmat

SCHRITT 1: EIGENSCHAFTEN DES PETRI-NETZES FESTLEGEN

Auf dem in Abbildung 4.5 dargestellten Auswahldialog werden die Eigenschaften, die das Petri-Netz im Ganzen betreffen, festgelegt. Die Eigenschaften der Plätze, Transitionen und Kanten folgen später.

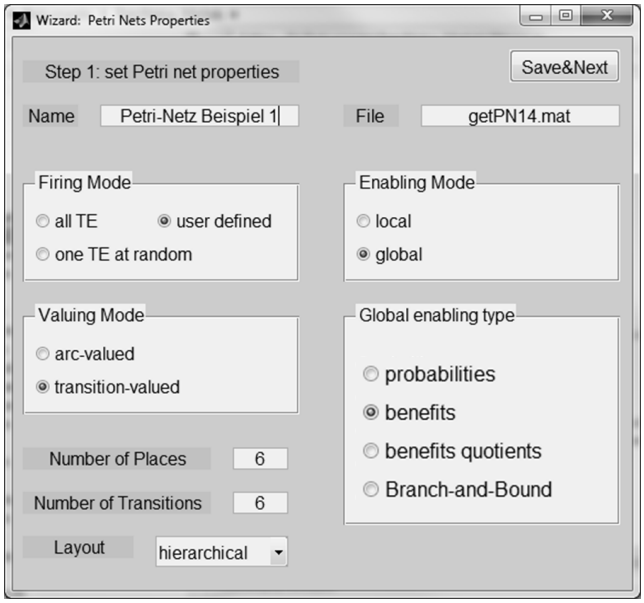


Abbildung 4.5: Eigenschaften des Petri-Netzes festlegen

Nachfolgend werden die einzelnen Kriterien im Detail erläutert:

- `Name`: Name des Petri-Netzes, der in der Titelleiste erscheint.
- `File`: Name der *mat*-Datei, in der das Petri-Netz gespeichert wird
- `Firing Mode`: Feuerungsstrategie (siehe Kapitel 2.3)

    - `all TE`: Feuerungsstrategie ($F1$) Alle freigeschalteten Transitionen feuern.
    - `one at random`: Feuerungsstrategie ($F2$) Eine der freigeschalteten Transitionen feuert, wobei die Auswahl zufällig getroffen wird.
    - `User defined`: Feuerungsstrategie ($F3$) Es gibt heiße und kalte freigeschaltete Transitionen. Die heißen Transitionen feuern auf jeden Fall und bei den kalten Transitionen kann der Benutzer entscheiden, ob sie feuern oder nicht.
- `Enabling Mode`: Konfliktlösungsstrategie (siehe Kapitel 2.2)
    - `local`: Jeder Platz löst seinen Konflikt alleine.
    - `global`: Alle Plätze lösen ihre Konflikte gemeinsam.
- `Valuing Mode`: Bewertungsart für die Konfliktlösung (siehe Kapitel 2.2)
    - `arc-valued`: Konfliktlösung durch Kantenbewertungen.
    - `transition-valued`: Konfliktlösung durch Transitionenbewertungen.
- `Number of Places`: Anzahl der Plätze, die das Petri-Netz enthalten soll.
- `Number of Transitions`: Anzahl der Transitionen, die das Petri-Netz enthalten soll.
- `Global enabling type`: Wenn unter `Enabling Mode global` ausgewählt wurde, öffnet sich dieses Feld, um die Bewertungsart festzulegen (siehe Kapitel 2.2.2)
    - `probabilities`: Wahrscheinlichkeiten (Algorithmus 2.4)
    - `benefits`: Nutzen (Algorithmus 2.5)
    - `benefit quotients`: Nutzenquotienten (Algorithmus 2.5)
    - `Branch-and-Bound`: Bewertung durch Nutzen und Konfliktlösung mithilfe eines Branch-and-Bound-Ansatzes.

SCHRITT 2: EIGENSCHAFTEN DER PLÄTZE FESTLEGEN

Im zweiten Schritt werden die Eigenschaften der Plätze in die in Abbildung 4.6 dargestellte Tabelle eintragen. Dazu gehören die Namen, die Anzahl Token, die die Plätze zu Beginn enthalten sollen, und die minimalen und maximalen Kapazitäten. Hierbei bedeutet die Abkürzung *Inf*, dass es keine obere Kapazitätsgrenze für diesen Platz gibt.

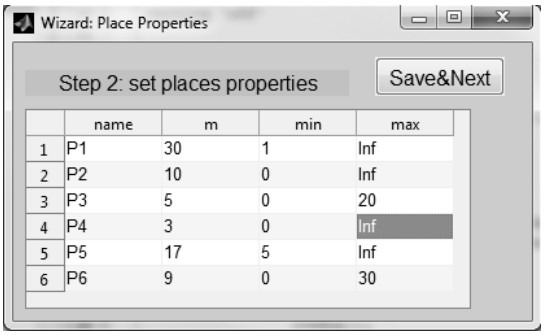


Abbildung 4.6: Eigenschaften der Plätze festlegen

SCHRITT 3: EIGENSCHAFTEN DER TRANSITIONEN FESTLEGEN

Nach den Eigenschaften der Plätze werden die Eigenschaften der Transitionen in die entsprechende Tabelle eingetragen (siehe Abbildung 4.7).

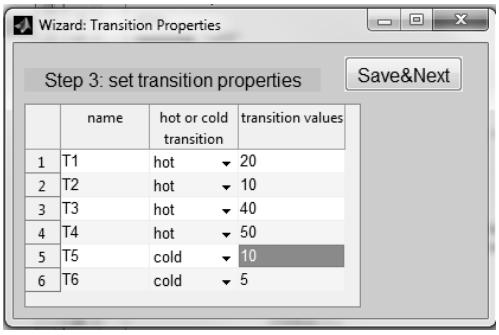


Abbildung 4.7: Eigenschaften der Transitionen festlegen

In der ersten Spalte kann der Name der Transition eingetragen werden. Die zweite Spalte enthält ein Auswahlmenü mit den Einträgen `hot` und `cold`. Hiermit kann entschieden werden, ob die Transition heiß oder kalt ist, sofern in Schritt 1 als Feuerungsstrategie benutzerdefiniert gewählt wurde. Die dritte Spalte erscheint nur, wenn in Schritt 1 ein transitionenbewertetes Petri-Netz ausgewählt wurde. In diesem Fall können in der dritten Spalte die Bewertungen der Transitionen eingetragen werden.

SCHRITT 4 - 6: EIGENSCHAFTEN DER KANTEN FESTLEGEN

Die Festlegung der Kanteneigenschaften teilt sich in drei Schritte auf. Zunächst muss bestimmt werden, welche Plätze mit welchen Transitionen verbunden sein sollen.

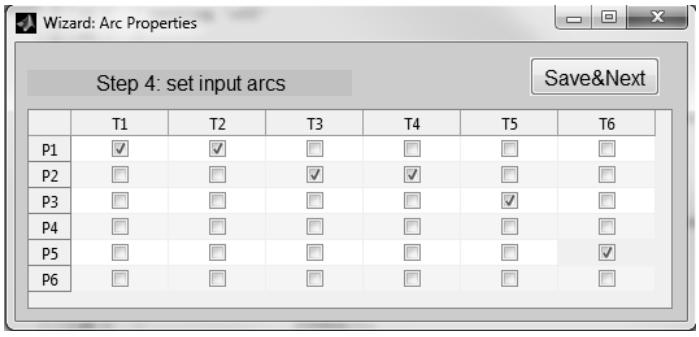


Abbildung 4.8: Eigenschaften der Inputkanten festlegen

Die Kanten, die von Plätzen ausgehen und in Transitionen münden, werden **Inputkanten** genannt. Durch Anklicken der Kästchen in Abbildung 4.8 wird eine Kante von dem jeweiligen Platz zu der jeweiligen Transition erzeugt.

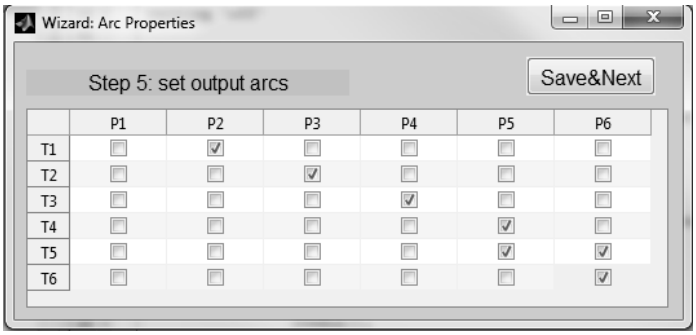


Abbildung 4.9: Eigenschaften der Outputkanten festlegen

Genauso werden in Schritt 5 die Kanten, die von den Transitionen ausgehen und in den Plätzen münden, erzeugt. Diese Kanten werden **Outputkanten** genannt.

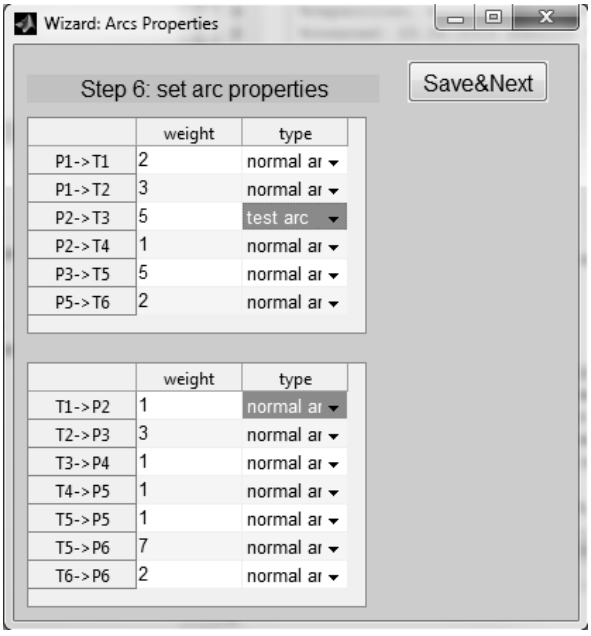


Abbildung 4.10: Gewichte und Typen der Kanten festlegen

Abschließend müssen in Schritt 6 noch die Kantengewichte eingegeben werden. Zudem kann bei den Inputkanten über ein Auswahlmenü die Art der Kante bestimmt werden. Zur Auswahl stehen die „normale“ Kante sowie die in Kapitel 2.4.2 vorgestellten Test- und Hemmkanten. Falls das Petri-Netz kantenbewertet ist, erscheint noch eine weitere Spalte, in der die Kantengewichte eingegeben werden können. Abbildung 4.11 zeigt das fertige Petri-Netz, das entsteht, wenn man die Einstellungen wie in den obigen Dialogen wählt.

4.4.3 PETRI-NETZE LADEN

Erstellte Petri-Netze werden in MATLAB-Strukturen gespeichert, die jederzeit wieder geladen werden können, um das Petri-Netz zu simulieren oder zu ändern. Dazu startet man die *PNmat*-App in MATLAB und wählt in dem Dialog in Abbildung 4.4 *load* aus. Es erscheint ein Standarddialog zum Öffnen von Dateien, in dem die gewünschte Petri-Netz-Struktur ausgewählt werden kann. Die Petri-Netz-Strukturen werden als *mat*-Datei gespeichert.

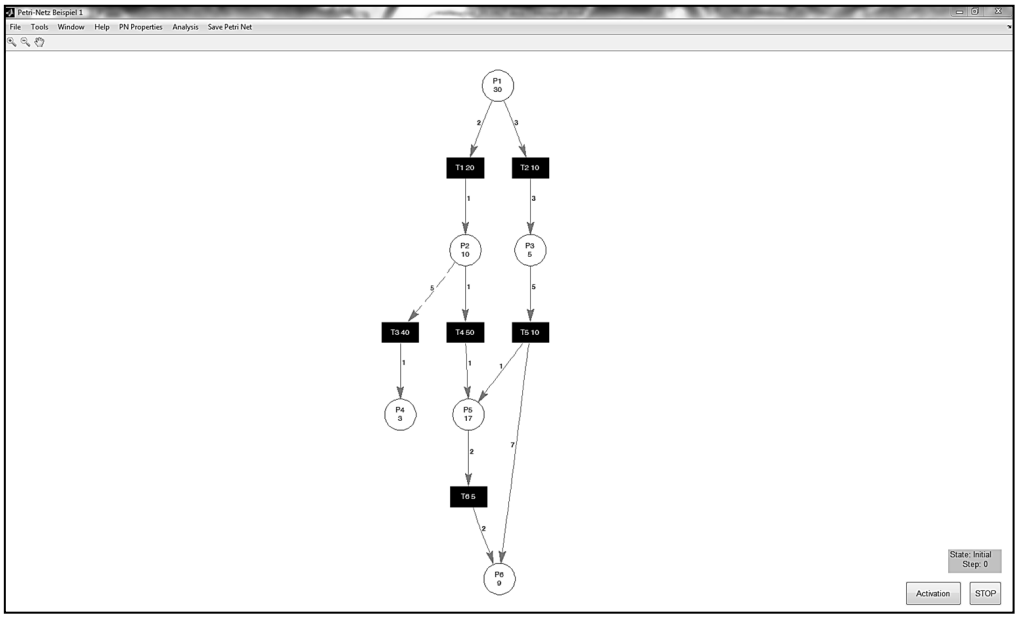


Abbildung 4.11: Fertiges Petri-Netz

4.4.4 PETRI-NETZE ÄNDERN UND SPEICHERN

Über den Menüpunkt *PN Properties* können bereits erstellte Petri-Netze geändert werden. Dieser Menüpunkt gliedert weiter in die Unterpunkte: *Petri Net*, *Places*, *Transitions* und *Arcs*.

Unter *Petri Net* können die in Schritt 1 erläuterten Einstellungen verändert werden (siehe Abbildung 4.5). Jedoch kann nicht die Anzahl der Plätze und Transitionen geändert werden und auch der Name der Datei kann dort nicht geändert werden. Zurzeit ist es noch nicht möglich Plätze oder Transitionen zu löschen oder hinzuzufügen.

Unter *Places* können die in Schritt 2 erläuterten und in Abbildung 4.6 dargestellten Einstellungen verändert werden. Zusätzlich kann die Markierung auf die Anfangsmarkierung zurückgesetzt werden. Dazu muss der Button *Reset m0* geklickt werden.

Die in Schritt 3 dargelegten Eigenschaften für Transitionen können unter dem Menüpunkt *Transitions* angepasst werden.

Die Eigenschaften der Kanten können unter *Arcs* angepasst werden. Dieser Menüpunkt untergliedert sich weiter in die Punkte *Arc Properties*, *Input Arcs* und *Output Arcs*. *Arc Properties* beinhaltet die Anpassung der Kantengewichte, der Art der Kante und bei einem kantenbewerteten Petri-Netz zusätzlich die Bewertungen (siehe Abbildung 4.10). Unter *Input Arcs* und *Output Arcs* können Kanten hinzugefügt oder gelöscht werden. (siehe Abbildung 4.8 und Abbildung 4.9).

Das veränderte Petri-Netz kann unter dem Menüpunkt *Save Petri Net* gespeichert werden.

4.4.5 PETRI-NETZE SIMULIEREN

Nachdem ein Petri-Netz erstellt oder geladen wurde, kann es schrittweise simuliert werden. Dazu werden die einzelnen Prozesse graphisch dargestellt (siehe Abbildung 4.1).

Beim Klick auf *Activation* unten rechts in der Ecke des Petri-Netz-Fensters (siehe Abbildung 4.11), werden alle aktiven Transitionen gelb markiert. In dem Petri-Netz *PN1* in Abbildung 4.2 sind alle Transitionen aktiv (siehe Abbildung 4.12 links). Anschließend werden beim Klick auf *Enabling* alle freigeschalteten Transitionen grün markiert, also alle diejenigen der aktiven Transitionen, die auch gleichzeitig feuern können (siehe Abbildung 4.12 Mitte). Wenn die benutzerdefinierte Feuerungsstrategie gewählt wurde, werden alle kalten freigeschalteten Transitionen hellgrün markiert und sind zunächst nicht zum Feuern freigegeben. Durch Anklicken werden diese Transitionen freigegeben und verändern ihre Farbe zum gleichen grün, wie das der heißen Transitionen. In Abbildung 4.12

(Mitte) sind die Transitionen $T5$ und $T6$ kalt und die restlichen sind heiß und sind somit ohne Benutzereingriff zum Feuern freigegeben.

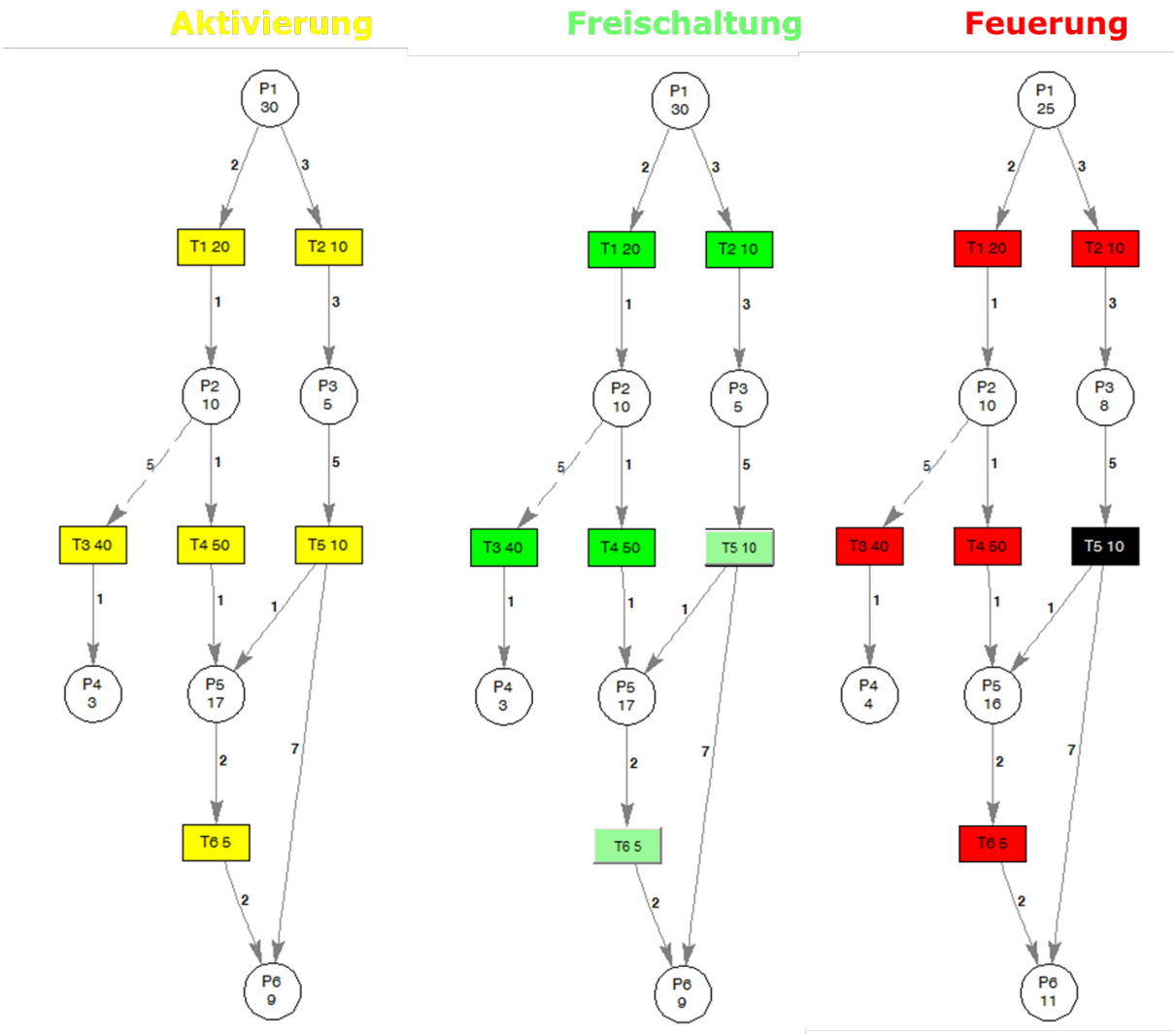


Abbildung 4.12: Schrittweise Simulation; links: gelbe Transitionen sind aktiv; Mitte: grüne Transitionen sind freigeschaltet, neongrüne sind heiß und lindgrüne sind kalt (benutzerdefinierte Feuerungsstrategie); rechts: rote Transitionen wurden gefeuert

Wählt man nur Transition $T6$ zum Feuern aus und klickt anschließend auf *Firing*, entsteht das rechte Petri-Netz in Abbildung 4.12. Alle gefeuerten Transitionen sind rot markiert und in den Plätzen wurden die Token aktualisiert. Nun hat man die Möglichkeit durch Klicken auf *Next Step* einen weiteren Simulationsschritt durchzuführen, der wieder in die drei beschriebenen Prozesse unterteilt ist.

Zudem hat man die Möglichkeit die Simulation durch einen Klick auf *STOP* zu beenden. Dann öffnet sich ein weiteres Fenster mit den Ergebnissen der Simulation. Die Ergebnisse der Simulation von *PN1* nach 10 Schritten sind in Abbildung 4.13 dargestellt.

Über das Auswahlmenü auf der rechten Seite kann man die Plätze auswählen, deren Tokenverläufe angezeigt werden sollen. Zusätzlich erhält man beim Klick auf *Markings* eine Tabelle mit den Markierungen der einzelnen Schritte und beim Klick auf *Firings* wird angezeigt, welche Transitionen in welchem Schritt gefeuert wurden. Die Simulationsergebnisse werden außerdem in der Datei *simulationResults.mat* gespeichert. Sie enthält die Petri-Netz-Struktur, die in A-Tabelle 1 zusammengefasst wird, und zudem die Matrix `M` mit den Markierungen pro Schritt, die Matrix `TFmat` mit den gefeuerten Transitionen pro Schritt und die Matrix `B` mit dem Nutzen pro Schritt, wenn die Nutzenbewertung ausgewählt wurde. Diese Datei wird bei einer neuen Simulation ersetzt. Wenn die Ergebnisse weiterhin benötigt werden, muss der Dateiname geändert werden.

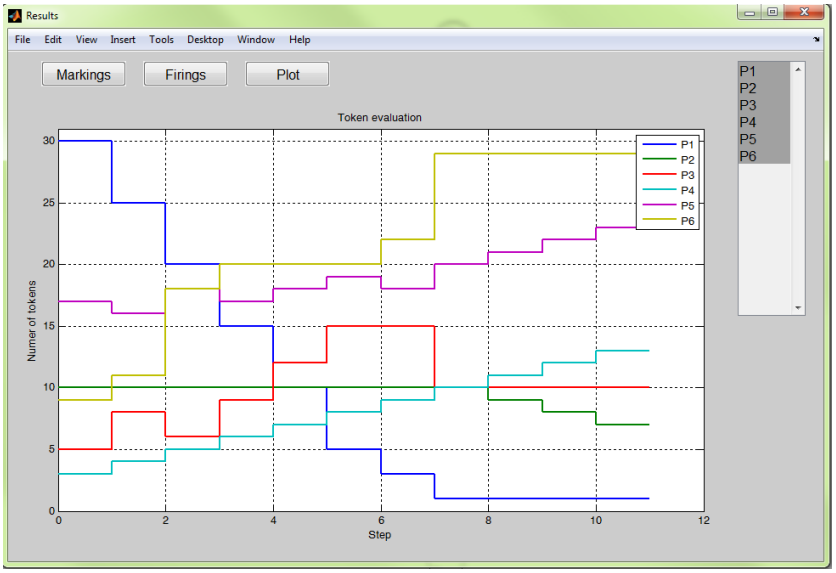


Abbildung 4.13: Simulationsergebnisse von PN1 nach 10 Schritten

4.4.6 Petri-Netze animieren

Bei der zuvor vorgestellten Simulation eines Petri-Netzes gelangt man durch Klicken von Prozess zu Prozess und von einem Schritt zum nächsten. Bei einer großen Anzahl an Simulationsschritten kann diese Art lange dauern und mühsam werden. Deshalb wird bei der Animation eine vorgegebene Anzahl Simulationsschritte im Hintergrund durchlaufen. Diese können entweder als Video aufgenommen und anschließend abgespielt werden oder es kann das Petri-Netz im Endzustand angesehen werden.

Für die Animation wählt man im Menü unter *Analysis* die Option *Animation* aus. Es erscheint ein Dialog, in dem die Anzahl der Simulationsschritte eingegeben werden kann. Anschließend

kann man wählen, ob man im Anschluss ein Video von allen Simulationsschritten ansehen möchte (*All steps*) oder nur das Endergebnis (*Result*). Wählt man die erste Option, kann im nächsten Dialog der Dateiname des Videos und die Bildfrequenz (*frames per second*) eingegeben werden. Anschließend wird das Video erzeugt, ohne dass auf dem Bildschirm etwas sichtbar ist. Nach der Fertigstellung hat man die Möglichkeit sich das Video mit dem Movie Player von MATLAB anzuschauen.

4.4.7 PETRI-NETZE OPTIMIEREN

Bei der Optimierung eines Petri-Netzes soll die optimale Feuerungsreihenfolge gefunden werden, die nach einer vorgegebenen Anzahl Schritten z.B. den maximalen Nutzen liefert (siehe Kapitel 3). Dazu wählt man im Menü *Analysis* die Option *Optimization* aus. Es öffnet sich ein Dialog, in den man die gewünschte Anzahl Schritte eingeben kann. Enthält das erstellte oder geladene Petri-Netz noch keine Bewertungen für die einzelnen Transitionen, öffnet sich eine Tabelle, in die diese eingegeben werden können. Zudem kann das Verfahren, das zur Lösung des Optimierungsproblems genutzt werden soll, ausgewählt werden. Zur Auswahl stehen die in Kapitel 3.3 vorgestellten Verfahren:

- Branch-and-Bound-Verfahren,
- Simulated Annealing,
- Tabu-Suche,
- Greedy Randomized Adaptive Search Procedure,
- Genetischer Algorithmus,
- Ameisenalgorithmus.

Wenn alle Einstellungen vorgenommen wurden, erscheint in der unteren rechten Ecke des Petri-Netz-Fensters der Button *START*, mit dem die Optimierung gestartet werden kann. Danach kann man die einzelnen Schritte der optimalen Feuerungsreihenfolge durchklicken (*Next Step*) oder man kann sich die Simulationsergebnisse als Plot anschauen (*PLOT,* siehe Abbildung 4.13).

5 ZUSAMMENFASSUNG UND AUSBLICK

In dieser Arbeit ist ein Petri-Netz-Formalismus zur qualitativen Modellierung von praxisrelevanten Prozessen und Systemen aus den unterschiedlichsten Anwendungsbereichen vorgestellt worden. Dieser Formalismus ist mithilfe von MATLAB implementiert worden, um eine graphische Modellierung, Simulation und Optimierung mit Petri-Netzen zu ermöglichen. Das Tool *PNmat* ist universell einsetzbar und nicht auf einen Anwendungsbereich begrenzt. Zunächst soll es primär in der Lehre zum Einsatz kommen, um Studierende mit der Petri-Netz-Modellierung praxisrelevanter Prozesse vertraut zu machen.

Der in (Proß 2013) entwickelte Petri-Netz Formalismus *xHPN* (extended Hybrid Petri Nets) ist im Bereich der qualitativen Modellierung um die in Tabelle 5.1 zusammengefassten Punkte erweitert worden.

Dieser weiterentwickelte Petri-Netz-Formalismus wurde in MATLAB umgesetzt, um eine graphische Modellierung und Simulation zu ermöglichen. Petri-Netze können mithilfe eines Assistenten einfach, schnell und intuitiv erstellt werden. Zudem können bereits erstellte Petri-Netze geladen und verändert werden. Die Animation bietet die Möglichkeit ein Video von der Simulation aufzunehmen und bei der Optimierung wird die Feuerungsreihenfolge mit maximalem Nutzen bzw. minimalen Kosten gefunden.

Bisher können mit der *PNmat* nur qualitative Petri-Netze erstellt und simuliert werden, d.h. die Zeit wird nicht berücksichtigt. Im Rahmen fortführender Arbeiten könnte der gesamte Umfang des *xHPN*-Formalismus in die *PNmat* integriert werden. Zunächst könnte der diskrete Formalismus mit einer Zeitbetrachtung versehen werden, indem jeder Transition eine Verzögerung zugeordnet wird. Anschließend könnte dieses diskrete Konzept in ein kontinuierliches überführt werden, um abschließend beide Welten in einem hybriden Petri-Netz zu vereinigen.

Zudem ist die *PNmat* ein erster Schritt, um den in (Bachmann et al. 2014) und (Kleine-Döpke 2013) vorgestellten gefärbten Petri-Netz-Formalismus zu simulieren. Der gravierende Unterschied ist hierbei, dass die Plätze in gefärbten Petri-Netzen verschiedenartige Token haben können (z.B. blaue, rote und grüne Token). Zum Feuern einer Transition sind dann

beispielsweise ein blauer und drei rote Token notwendig. Im Rahmen des Projekts *gefärbte Petri-Netze* des *FSP AMMO* soll diese Arbeit in Zukunft weiter vorangetrieben werden.

Ein weiterer Ansatzpunkt für zukünftige Arbeiten ist die graphische Oberfläche. Hier sollten Plätze und Transitionen aus bereits erstellten Petri-Netzen entfernt und hinzugefügt werden können. Zudem würde es die Modellbildung vereinfachen, wenn die Petri-Netze per Drag-and-Drop erstellt werden könnten.

Tabelle 5.1: Erweiterungen des xHPN-Formalismus in dieser Arbeit

xHPN in (Proß 2013)	Erweiterungen dieser Arbeit
Konflikte werden nur lokal gelöst.	Konflikte können lokal oder global gelöst werden.
Konflikte werden lokal durch Bewertungen an den Kanten gelöst.	Konflikte werden lokal oder global entweder durch Bewertungen an den Kanten oder den Transitionen gelöst.
Die Kantenbewertungen können Prioritäten oder Wahrscheinlichkeiten sein.	Die Kanten- und Transitionenbewertungen können in Form von Wahrscheinlichkeiten oder Nutzen vorliegen. Bei Kantenbewertungen können es zudem Nutzenquotienten sein. Die Bewertung durch Prioritäten ist in der Nutzenbewertung enthalten und wird deshalb nicht als zusätzliche Möglichkeit angeboten.
Die lokalen Konflikte werden mit Algorithmus 2.1 - Algorithmus 2.3 gelöst.	Lokale Konflikte werden mit Algorithmus 2.1 - Algorithmus 2.3, Algorithmus 2.6 und Algorithmus 2.7 gelöst. Globale Konflikte werden mit Algorithmus 2.5 und Algorithmus 2.6 gelöst.
Alle freigeschalteten Transitionen feuern.	Alle freigeschalteten Transitionen feuern. Eine der freigeschalteten Transitionen feuert, wobei die Auswahl zufällig getroffen wird. Es gibt heiße und kalte freigeschaltete Transitionen. Die heißen freigeschalteten Transitionen feuern auf jeden Fall und bei den kalten Transitionen kann der Benutzer entscheiden, ob sie feuern oder nicht.
-	Optimierung von Petri-Netzen.

Abschließend lässt sich sagen, dass mit der *PNmat* ein mächtiges Tool zur Verfügung steht, mit dem praxisrelevante Prozesse geplant, dargestellt und optimiert werden können. Es können sowohl bereits existierende Prozesse aus unterschiedlichsten Anwendungsbereichen bezüglich deren Effizienz (minimale Kosten, maximaler Nutzen, …) optimiert werden, als auch neue Prozesse vor deren Realisierung ausgiebig getestet und anschließend bestmöglich umgesetzt werden.

ANHANG

Die folgende Tabelle fasst die Variablen der Petri-Netz-Struktur zusammen.

A-Tabelle 1: Komponenten der Petri-Netz-Struktur PN

Name	Typ	Bedeutung
`at`	Zahl	Kanten- (at=1) oder transitionenbewertetes (at=2) Petri-Netz
`bfOpt`	Vektor $(n_t \times 1)$	Transitionenbewertungen für die Optimierung
`cl`	Vektor $(n_p \times 1)$	Minimale Kapazitäten
`cu`	Vektor $(n_p \times 1)$	Maximale Kapazitäten
`e`	Vektor $(n_p \times 1)$	Lokale Lösungstypen
`eg`	Zahl	Globaler Konfliktlösungstyp
`file`	String	Name der Datei, in der die Petri-Netz-Struktur gespeichert ist
`fm`	Zahl	Feuerungsstrategie
`Fm`	Inline-Funktion	Gewichte von Plätzen zu Transitionen als Funktionen der Markierung
`Fstr`	String	Kanten und Gewichte von Plätzen zu Transitionen
`Gm`	Inline-Funktion	Gewichte von Transitionen zu Plätzen als Funktionen der Markierung
`Gstr`	String	Kanten und Gewichte von Transitionen zu Plätzen
`hc`	Vektor $(n_t \times 1)$	Heiße oder kalte Transitionen
`I`	Matrix $(n_p \times n_t)$	Hemmkanten
`layout`	String	Layout des Petri-Netzes (hierarchical, equilibrium, radial)
`locGlo`	Zahl	Lokale oder globale Konfliktlösung
`m0`	Vektor $(n_p \times 1)$	Anfangsmarkierung
`n_p`	Zahl	Anzahl der Plätze
`n_t`	Zahl	Anzahl der Transitionen
`name_p`	String	Namen der Plätze
`name_pn`	String	Name des Petri-Netzes
`name_t`	String	Namen der Transitionen
`p_in`	Matrix $(n_p \times n_t)$	Bewertungen der Kanten von Transitionen zu Plätzen
`p_out`	Matrix $(n_p \times n_t)$	Bewertungen der Kanten von Plätzen zu Transitionen
`T`	Matrix $(n_p \times n_t)$	Testkanten
`tp`	Vektor $(n_t \times 1)$	Transitionenbewertungen

Die folgende Tabelle fasst alle m-files der *PNmat*, deren Aufgabe und Verwendung zusammen. Zudem wird aufgelistet, welche anderen Funktionen in der jeweiligen Funktion benötigt werden.

A-Tabelle 2: m-Files der PNmat

m-File	Aufgabe	Verwendung in	Ruft auf

`@biograph`	Modifizierte m-files zur Erstellung des Biographen zur Darstellung des Petri-Netzes.	`allCombs`
		`animation`
		`draw_pn`
		`optimization`
		`setMenu`
		`sim_PN`
			-

`activation`	Berechnet die aktiven Transitionen.	`animation,` `sim_PN`	`setParam`
`allcombs`	Stellt die kalten Transitionen bei der benutzerdefinierten Feuerungsstrategie dar und ermittelt welche Transitionen gefeuert werden sollen.	`firingSelection`	`@biograph` `sim_PN`

`animation`	Führt die Petri-Netz-Animation aus.	`setMenu`	`@biograph`
			`activation`
			`calBenefit`
			`enabling`
			`firingProcess`
			`saveMarking`
			`setParam`
			`updateWeights`

`calBenefit`	Berechnet den Nutzen eines Simulations-schrittes bei Bewertung der Transitionen bzw. der Kanten durch Nutzen oder Nutzen-quotienten.	`animation,`
		`optimization`
		`sim_PN`
			`setParam`

`cell2str`	Wandelt die Kantengewichte von einem Cell Array in einen String um.	`setMenu` `wizardPN`	-

`draw_PN`	Aufbereitung der Daten der Petri-Netz-Struktur, um sie als Verbindungmatrix an die modifizierte Biograph-Funktion zu übergeben. Graphische Darstellung des Petri-Netzes.	`optimization`
		`PNsim`
		`setMenu`
		`wizardPN`
			`@biograph`
			`sim_PN`
			`setParam`

enable_benefit_ global	Globale Konfliktlösung durch Nutzenbewertungen	enabling	-
enable_benefit_in	Lokale Input-Konfliktlösung durch Nutzenbewertungen	enabling	-
enable_benefit_out	Lokale Output-Konfliktlösung durch Nutzenbewertungen	enabling	-
enable_benefitBB_ global	Globale Konfliktlösung durch Nutzenbewertungen und Branch-and-Bound-Methode	enabling	-
enable_benefitQuo_ in	Lokale Input-Konfliktlösung durch Nutzenquotienten-bewertungen	enabling	-
enable_benefitQuo_ out	Lokale Output-Konfliktlösung durch Nutzenquotienten-bewertungen	enabling	-
enable_benefitBB_in	Lokale Input-Konfliktlösung durch Nutzenbewertungen und Branch-and-Bound-Methode	enabling	-
enable_benefitBB_ out	Lokale Output-Konfliktlösung durch Nutzenbewertungen und Branch-and-Bound-Methode	enabling	-
enable_prob_global	Globale Konfliktlösung durch Wahrscheinlichkeiten	enabling	-
enable_prob_in	Lokale Input-Konfliktlösung durch Wahrscheinlichkeiten	enabling	-
enable_prob_out	Lokale Output-Konfliktlösung durch Wahrscheinlichkeiten	enabling	-
enabling	Löst die Konflikte und berechnet die freigeschalteten Transitionen.	animation, sim_PN	enable_benefit_ global enable_benefit_ in enable_benefit_ out enable_benefitB B_global enable_benefit Quo_in enable_benefit Quo_out

			enable_benefit BB_in enable_benefit BB_out enable_prob_ global enable_prob_in enable_prob_out setParam.m
figurePosition	Berechnet die Position für das Petri-Netz-Fenster.	setMenu	-
firingProcess	Berechnet die neue Markierung des Petri-Netzes nach der Feuerung.	animation optimization sim_PN	-
firingSelection	Wählt je nach Strategie die zu feuernden Transitionen unter den freigeschalteten aus.	sim_PN	allcombs
OFS_GA.m	Ermittelt die OFR mittels genetischen Algorithmus.	optimization	-
OFS_GRASP.m	Ermittelt die OFR mittels GRASP.	optimization	-
OFS_MMAS.m	Ermittelt die OFR mittels Ameisen-algorithmus.	optimization	-
OFS_SA.m	Ermittelt die OFR mittels Simulated Annealing.	optimization	-
OFS_TS.m	Ermittelt die OFR mittels Tabu-Suche.	optimization	-
optimization	Führt die Optimierung durch, bei der die Feuerungs-reihenfolge mit maximalen Nutzen gefunden wird.	setMenu	@biograph calBenefit draw_PN firingProcess plotResults saveMarking setMenu updateWeights
plotResults	Stellt die Ergebnisse der Simulation dar und speichert sie in der Datei *simulation-Results.mat* ab.	optimization sim_PN	setParam
PNsim	Startet die Simulation. Wahl zwischen Petri-Netz laden oder erstellen.	-	wizardPN, draw_PN
saveMarking	Speichert die	animation	-

	Markierung, den Simulationsschritt, die gefeuerten Transitionen und den Nutzen für die abschließende Darstellung.	optimization setMenu	
setMenu	Erstellt das Menü des Petri-Netz-Fensters.	optimization sim_PN	@biograph animation figurePosition saveMarking
setParam	Setzt die Parameter der Petri-Netz-Struktur für eine verkürzte Schreibweise.	activation allCombs animation calBenefit draw_PN enabling firingSelection optimization plotResults PNsim sim_PN updateWeights wizardPN	-
sim_PN	Ruft die für die Simulation notwendigen Prozesse der Reihe nach auf und aktualisiert die Darstellung.	allCombs animation draw_PN firingSelection simPN setMenu	@biograph activation calBenefit enabling firingProcess firingSelection plotResults saveMarking setMenu setParam updateWeights
str2cell	Wandelt die Kantengewichte von einem String in ein Cell Array um.	draw_PN setMenu wizardPN	-
updateWeights	Aktualisiert Kantengewichte bei Veränderung durch den Benutzer und nach Feuerung, da funktionale Kantengewichte möglich sind.	animation, optimization setMenu sim_PN	setParam
wizardPN	Assistent zur Erstellung von Petri-Netzen	PNsim	cell2str draw_PN setParam str2cell

LITERATURVERZEICHNIS

Bachmann B, Kleine-Döpke T, Kruse HJ, Ochel L, Proß S (2014) Petri-Netz-Formalismen und Lösungsansätze für allgemeine Konfliktsituationen bei Feuerprozessen in Petri-Netz-Modellen, Forschungsschwerpunkt AMMO, FH Bielefeld, in Vorbereitung

David R, Alla H (2001) On Hybrid Petri Nets. Discrete Event Dynamic Systems: Theory and Applications(11): 9–40

David R, Alla H (2010) Discrete, Continuous, and Hybrid Petri Nets. Springer Verlag, Berlin Heidelberg

Domschke W, Drexl A (2005) Einführung in Operations Research. Springer Verlag, Berlin Heidelberg

Dorigo M, Stützle T (2004) Ant Colony Optimization. MIT Press, Cambridge, Massachusetts

Gendreau, M, Potvin, JY (Eds.) (2010) Handbook of metaheuristics (Vol. 146). Springer Verlag

Gerdes I, Klawonn F, Kruse R (2004) Evolutionäre Algorithmen. Springer Verlag

Haußer F, Luchko Y (2011) Mathematische Modellierung mit MATLAB® - Eine praxisorientierte Einführung. Spektrum Akademischer Verlag

Hofestädt R, Thelen S (1998) Quantitative modeling of biochemical networks. In Silico Biology 1(1):39–53

Imboden DM, Koch S (2008) Systemanalyse: Einführung in die mathematische Modellierung natürlicher Systeme. Springer Verlag, Berlin, Heidelberg, New York

Kleine-Döpke T (2013) Konfliktlösungen für den nebenläufigen Feuerprozess bei gefärbten Petri-Netzen und ihre Anwendungen, Bachelorarbeit, FH Bielefeld

Kramer O (2009) Computational Intelligence - Eine Einführung. Springer-Verlag, Berlin, Heidelberg

Kruse R, Borgelt C, Klawonn F, Moewes C, Ruß G, Steinbrecher M (2011) Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze. Vieweg+Teubner Verlag

MathWorks (2012a) MATLAB Bioinformatics Toolbox™ User's Guide R2012b

MathWorks (2012b) MATLAB Optimization Toolbox™ User's Guide R2012b

Matsuno H, Tanaka Y, Aoshima H, Doi A, Matsui M, Miyano S (2003) Biopathways representation and simulation on hybrid functional Petri net. In Silico Biology 3(3):389–404

Nassi I, Shneiderman B (1973) Flowchart techniques for structured programming. ACM SIGPLAN Not.8:12-26

Ortlieb P, v. Dresky C, Gasser I, Günzel S (2009) Mathematische Modellierung - Eine Einführung in zwölf Fallstudien. Vieweg+Teubner

Petri CA (1962) Kommunikation mit Automaten. Dissertation, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik, Bonn

Pomberger G, Dobler H (2008) Algorithmen und Datenstrukturen – Eine systematische Einführung in die Programmierung. Pearson Studium

Proß S (2007) Entwicklung eines Entscheidungsunterstützungssystems zur optimalen Bestückung von Kommissionierlagern auf der Grundlage von Ameisenheuristiken. Diplomarbeit, Fachhochschule Bielefeld

Proß S (2013) Hybrid Modeling and Optimization of Biological Processes. Dissertation, Universität Bielefeld

Proß S, Bachmann B (2012) PNlib – An Advanced Petri Library for Hybrid Process Modeling. In: Proceedings of the Modelica Conference, Munich

Suhl L, Mellouli T (2009) Optimierungssysteme: Modelle, Verfahren, Software, Anwendung. Springer-Verlag, Berlin, Heidelberg

Stützle T, Hoos HH (2000) MAX-MIN Ant System. In: Future Generation Computer Systems 16:889–914

Weicker K (2007) Evolutionäre Algorithmen. Teubner Verlag, Wiesbaden

Zimmermann HJ (2008) Operations Research – Methoden und Modell. Für Wirtschaftsingenieure, Betriebswirte, Informatiker, Vieweg & Sohn Verlag, Wiesbaden

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

Abkürzung	Bedeutung
MKP	Mehrdimensionales Knapsackproblem
OFR	Optimale Feuerungreihenfolge
GRASP	Greedy Randomized Adaptive Search Procedure
RKL	Restringierte Kandidatenliste
ZE	Zeiteinheit
xHPN	erweitertes hybrides Petri-Netz (extended hybrid Petri Net)
PNlib	Petri Net library (Modelica-Bibliothek zur Petri-Netz-Simulation)
PNmat	matrizenbasiertes Petri-Netz-Simulationstool in Matlab

Autoren und Kontakte

Autorin

Dr. rer. nat. Sabrina Proß

Kontaktdaten

Fachhochschule Bielefeld
Fachbereich Ingenieurwissenschaften und Mathematik
FSP Angewandte Mathematische Modellierung und Optimierung

Am Stadtholz 24
33609 Bielefeld

Tel.: +49.521.106-7403
Fax: +49.521.106-7176
E-Mail: ammo@fh-bielefeld.de
Web: www.fh-bielefeld.de/ammo

Ansprechpartner: Prof. Dr. Dr. Rainer Ueckerdt

Tel.: +49.521.106-7409
Fax: +49.521.106-7176
E-Mail: sabrina.pross@fh-bielefeld.de

Ansprechpartner: Dr. rer. nat. Sabrina Proß

Veröffentlichungsreihe:

AMMO – Berichte aus Forschung und Technologietransfer

Heft 1: Dezember 2013, AMMO-Team, *Informationen über den Forschungs- und Entwicklungsschwerpunkt Angewandte Mathematische Modellierung und Optimierung.*

Heft 2: April 2014, R. Walden und V.-M. Roemer, *Methoden der quantitativen rechnergestützten CTG-Analyse.*

Heft 3: Juli 2014, R. Ueckerdt, H.-W. Schmidt, M. Weber, E. Mindlina, *Entwicklung einer Dispatcherfunktion zur Überprüfung von Nominierungsmengen in der Betriebsführung von Erdgasspeichern.*

ISSN 2198-4824

Herausgeber: Sprecher FSP AMMO
Fachhochschule Bielefeld