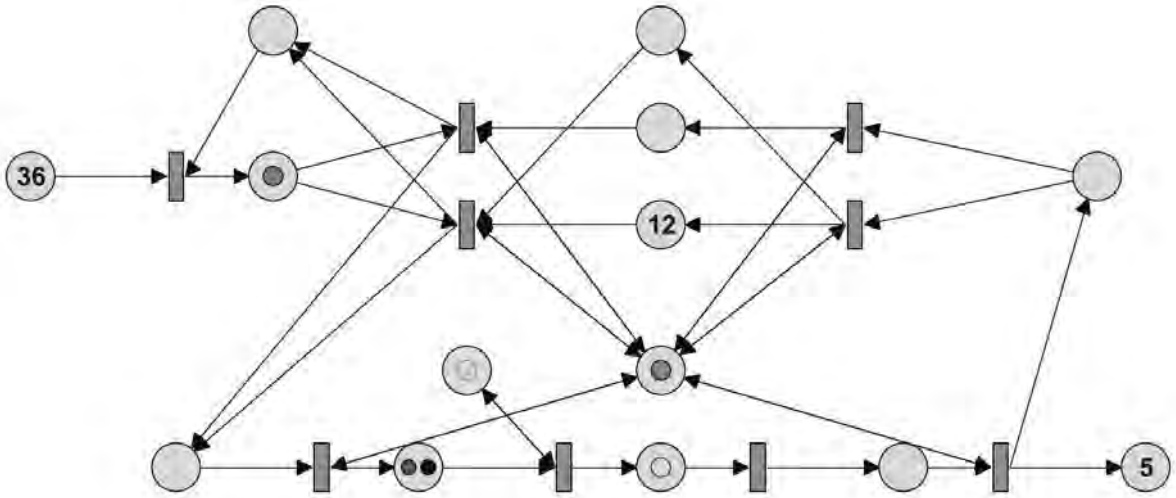
FORTSCHRITTSBERICHT
SIMULATION FBS 15



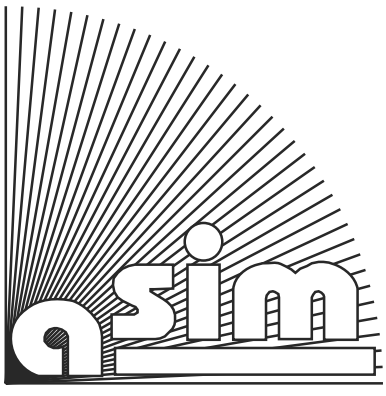
Thomas Löscher

Optimisation of Scheduling Problems Based on Timed Petri Nets

ARGESIM

ISBN Ebook 978-3-903347-15-1
ISBN Print 978-3-901608-65-0

DOI: 10.11128/fbs.15


ARGESIM

Fortschrittsberichte Simulation

FBS Band 15

Herausgegeben von **ASIM**
Arbeitsgemeinschaft **Sim**ulation, Fachausschuss der GI im
Fachbereich ILW – Informatik in den Lebenswissenschaften

Thomas Löscher

Optimisation of Scheduling Problems Based on Timed Petri Nets

ARGESIM / ASIM – Verlag, Wien, 2009
ISBN Print 978-3-901608-65-0

Ebook Reprint 2020
ISBN Ebook 978-3-903347-15-1
DOI: 10.11128/fbs.15

Fortschrittsberichte Simulation

Herausgegeben von **ASIM**, Arbeitsgemeinschaft Simulation, Fachausschuß der GI im Fachbereich ILW – Informatik in den Lebenswissenschaften

Betreuer der Reihe:

Prof. Dr.-Ing. Th. Pawletta (ASIM)
Hochschule Wismar
Phillip-Müller-Str., 23952 Wismar, Germany
Tel: +49-3841-753-406, Fax: +49-3841-753-132
Email: **pawel@mb.hs-wismar.de**

Dr.-Ing. habil. D.P.F. Schwarz (ASIM)
Fraunhofer-Institut für Integrierte Schaltungen
Zeunerstr. 38, 01069 Dresden, Germany
Tel: +49-351- 4640 - 730, Fax: +49-351-4640-703
Email: **schwarz@eas.iis.fhg.de**

Prof. Dr. F. Breitenecker (ARGESIM / ASIM)
Technische Universität Wien
Wiedner Hauptstraße 8 - 10, 1040 Wien, Austria
Tel: +43-1-58801-10115, Fax: +43-1-58801-10199
Email: **Felix.Breitenecker@tuwien.ac.at**

FBS Band 15

Titel: Optimisation of Scheduling Problems Based on Timed Petri Nets

Autor: Dipl.-Ing. Dr. Thomas Löscher
Inst. f. Analysis und Scientific Computing
Technische Universität Wien
Wiedner Hautpstrasse 8-10
1040 Wien, Austria
Email: **thomas@loescher.at**

Begutachter des Bandes:

Prof. Dr. F. Breitenecker, Prof. Dr. G. Mušič

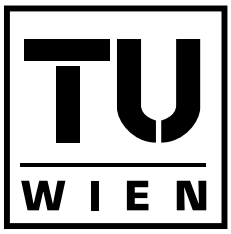
ARGESIM / ASIM – Verlag, Wien, 2009

ISBN Print 978-3-901608-65-0

Ebook Reprint 2020

ISBN Ebook 978-3-903347-15-1

DOI: 10.11128/fbs.15



TECHNISCHE
UNIVERSITÄT
WIEN

VIENNA
UNIVERSITY OF
TECHNOLOGY

DISSERTATION

Optimisation of Scheduling Problems Based on Timed Petri Nets

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker
E 101
Institut für Analysis and Scientific Computing

eingereicht an der Technischen Universität Wien
Fakultät für Mathematik und Geoinformation

von

Dipl.-Ing. Thomas Löscher
9925632
Sandweg 13, 2070 Retz

Wien, im Mai 2007

Kurzfassung

Diese Dissertation beschäftigt sich mit der Modellierung und Simulation von Reihenfolgeproblemen basierend auf Petri Netzen. Im Besonderen werden zeiterweiterte, gefärbte und stochastische Petri Netze verwendet, um spezielle Reihenfolgeprobleme aus dem Bereich von Produktionsprozessen und anderen diskreten Ereignissystemen zu modellieren und implementieren. Die Petri Netz Modelle werden über den Zeitbereich simuliert und die Optimierung der Eingangsreihenfolgen erfolgt mittels implementierter simulationsbasierter Optimierung.

Petri Netze bieten die Möglichkeit Bedingungen auf der höchsten Ebene zu behandeln. Das ist ein großer Vorteil im Vergleich zu ereignis-orientierten Zugängen zur Modellierung und Simulation von diskreten Ereignissystemen. Wenn Konflikte oder Probleme mit der gleichzeitigen Benutzung vorhandener Mittel auftreten, muss eine Strategie zum Auflösen gleichzeitiger Ereignisse implementiert werden. Im Gegensatz dazu realisieren Petri Netze diese Probleme durch die grundlegenden Eigenschaften ihrer Struktur.

In dieser Arbeit wird eine neue Art der Konfliktlösung eingeführt und es wird ein neuer Weg entwickelt um Feuerungsreihenfolgen zu definieren. Dieser neue Zugang bietet die Möglichkeit Reihenfolgeprobleme zu modellieren, ohne dabei abhängig von Konflikten zu sein. Die Optimierung von Reihenfolgeproblemen wird mittels automatisierter Veränderung und Auswertung der verwendeten Reihenfolgen und Parameterspezifikationen durchgeführt. Diese Art des Optimierungsproblems ist zu komplex, um bis zur Optimalität gelöst zu werden. Eine Erfolg versprechende Alternative bietet die Verwendung von Heuristiken, wie Genetische Algorithmen, Simulated Annealing oder Threshold Accepting.

Alle diese Methoden sind in der so genannten MATLAB PetriSimM Toolbox implementiert, die die Fähigkeit der Modellierung, Simulation und Optimierung von zeiterweiterten, gefärbten und stochastischen Petri Netzen besitzt. Im Fall von stochastischen Prozessen ist der Vergleich von alternativen Systemkonfigurationen ein nichttriviales Problem. In dieser Arbeit sind ein sequentieller paarweiser t-Test und Varianzreduktionstechniken implementiert, um die stochastische Optimierung von Reihenfolgeproblemen zu lösen. Im Zuge dieser Arbeit ist die PetriSimM Tool-

box entwickelt und in die MATLAB Umgebung integriert worden. Alle entwickelten Funktionalitäten und Fähigkeiten werden anhand von zwei Fallstudien verglichen und getestet. Die Fallstudien beinhalten die Modellierung, Simulation und Optimierung einer Fertigungszelle und dem bekannten Problem des Handlungsreisenden.
Diese Dissertation ist folgendermaßen strukturiert. Zuerst werden die Grundlagen von Petri Netzen vorgestellt und es wird ein Überblick über Petri Netze und deren Erweiterungen gegeben. Weiters wird die Definition von Petri Netzen formuliert, die es ermöglicht, Reihenfolgeprobleme zu modellieren und es werden die neuen Ansätze zur Definierung von Feuerungsreihenfolgen und Prioritäten beschrieben. Alle verwendeten Optimierungsalgorithmen werden präsentiert und der sequentielle t-Test und die implementierten Varianzreduktionstechniken werden vorgestellt. Als nächstes folgt die MATLAB PetriSimM Toolbox wo grundlegende Fähigkeiten umrissen und erweiterte Funktionalitäten beschrieben werden. Abschließend wird diese Arbeit von der Implementierung und den Optimierungsresultaten der beiden Fallstudien komplettiert.

Abstract

This PhD thesis deals with modelling and simulation of scheduling and sequencing problems based on Petri Nets. In particular, Timed, Coloured, and Stochastic Petri Nets are used to model and implement specific scheduling problems in the field of production processes and other discrete event systems. The Petri Net models are simulated over the time domain and a simulation-based optimisation is implemented to optimise the input sequences.

Petri Nets offer the possibility to handle conditions on the highest layer. This is a big advantage compared to event-oriented approaches for modelling and simulation of discrete event systems. If conflict or resource sharing problems occur, a strategy of solving simultaneous events should be implemented. In contrast, Petri Nets solve these problems through the basic properties of their structure.

In this thesis a new conflict resolution is implemented and a sophisticated way of defining firing sequences is developed. This new approach offers the possibility to model queuing, sequencing or scheduling problems being independent of the appearance of any conflicts. The optimisation of sequencing and scheduling problems works by automated changing and evaluating of the used sequences and parameter specifications. This kind of optimisation problem is too complex to be solved to optimality. A promising alternative is to use heuristics, like genetic algorithms, simulated annealing or threshold accepting.

All these methods are implemented in the so called MATLAB PetriSimM toolbox which offers the capability of modelling, simulation, and optimisation of Timed, Coloured, and Stochastic Petri Nets. In case of stochastic processes the comparison of alternative system configurations is a highly sophisticated problem. In this work a sequential paired t-test and variance reduction techniques are used and implemented to solve the stochastic optimisation for sequencing and scheduling problems. In the course of this PhD thesis the PetriSimM toolbox is developed and embedded in the MATLAB environment. All the implemented features, functionalities and capabilities are compared and tested in two case studies including the modelling, simulation and optimisation of a production cell and the well-known travelling salesman problem.

This PhD thesis is structured as follows. First of all, the basics of Petri Nets are

introduced and an overview is given about the state-of-the-art of Petri Nets and their known extensions. Furthermore, the definition for Petri Nets is formulated to allow the modelling of scheduling problems and the new approaches of defining firing sequences and priorities are described. All used optimisation algorithms are presented and the sequential t-test and the implemented variance reduction techniques are introduced. Next follows the MATLAB PetriSimM toolbox where basic features are outlined and advanced functionalities are described. Finally, the implementation and the results of the optimisation of the two case studies complete this work.

Acknowledgements

The work with this dissertation has been extensive and trying, but in the first place exciting, instructive, and fun. Without help, support, and encouragement from several persons, I would never have been able to finish this work.

First of all, I would like to thank my supervisor Professor Dr. Felix Breitenecker, for his inspiring and encouraging way to guide me to a deeper understanding of knowledge work, and his invaluable comments during the whole work with this dissertation.

I am very grateful to Professor Dr. Gašper Mušič for supervising me during my semester abroad at the university of Ljubljana. His understanding, encouraging and personal guidance have provided a good basis for the present thesis.

I will also give a special thanks to Dr. Dejan Gradišar for a fruitful collaboration and interesting and endless discussions concerning the work.

I am also very grateful to everyone that have read parts of the manuscript, especially Nicole Misener.

At last, to my parents thanks for supporting me with your love and understanding and for enabling me to do my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Petri Nets basically model processes on a very low level. Up to now a lot of extensions of Petri Nets exist. The introduction of time delays makes it possible to simulate over the time domain. The use of colours simplifies the graphical description and due to this fact models of greater complexity can be defined and used. In this work another interesting and important extension is developed to get the capability of modelling scheduling problems. This means that it is possible to build up Petri Nets based on the description of scheduling problems. All conditions and constraints are realised by the basic properties of Petri Nets. The input parameters are characterised by different firing sequences of the selected Petri Net corresponding to sequences of the present scheduling problem. On this account system configurations can be easily changed by the use of different input sequences. All properties of scheduling problems are now described by the use of Petri Nets and the evaluation of the scheduling problem is reduced to a single simulation run of the Petri Net based on the used input sequences.

Starting from this development an automated optimisation can be implemented based on several heuristic methods. The scheduling problem is reduced to a simple combinatorial problem. The objective function is defined by the evaluation of the Petri Net simulation over the time domain. Another aspect of this work is given by the use of stochastic time delays. In this case the stochastic optimisation results in a highly sophisticated problem.

All these extensions and functionalities are developed and implemented in the so called MATLAB PetriSimM toolbox where the user can model, simulate and optimise scheduling problems based on Petri Nets.

1.2 Contents in brief

1.2.1 Petri Nets

In this chapter the basics of Petri Nets are introduced and an overview is given about the state-of-the art of Petri Nets and their known extensions. All kinds and types of Petri Nets are presented and the different simulation algorithms are described. In particular non-timed, timed, coloured and stochastic Petri Nets are considered.

1.2.2 Scheduling Problems and Petri Nets

In this chapter a special definition for Petri Nets is formulated to allow the modelling of scheduling problems. Furthermore the capabilities of defining sequences and priorities are introduced and conflict resolution methods are presented.

1.2.3 Optimisation

In this chapter all implemented optimisation algorithms are presented. In this work Simulated Annealing, Threshold Accepting and Genetic algorithms are used to optimise scheduling problems based on Petri Nets. A sequential paired t-test and variance reduction techniques are used and implemented to solve the stochastic optimisation.

1.2.4 MATLAB PetriSimM Toolbox

In the course of this work the MATLAB PetriSimM toolbox is developed where users can model, simulate and optimise scheduling problems based on Petri Nets. The main features of the toolbox are outlined and advanced functionalities are described. Selected benchmarks round off this chapter.

1.2.5 Case Studies

Two case studies are presented to test and compare the implemented optimisation algorithms. A production cell is modelled and optimised with respect to arrival times and selected due dates. Furthermore the well-known Travelling Salesman Problem is modelled and optimised based on deterministic and stochastic time delays.

1.2.6 Summary and Outlook

Finally, all results and achievements are summarised and possible improvements and future developments are mentioned.

Chapter 2

Petri Nets

2.1 Introduction

Petri Nets were first introduced by Carl Adam Petri [1] in 1962 and are a graphical and mathematical modelling tool which is applicable to many systems. Petri Nets are a formalism for the description of concurrency and synchronisation inherent in modern distributed systems [2, 3, 4].
This chapter shows all kinds and types of Petri Nets used in this work and should give an overview of the state-of-the-art of Petri Nets. In particular, first Place-Transition Petri Nets are considered and all needed and used properties are introduced. Next follows the definition of Timed Petri nets where two different approaches of the implementation of time in Petri Nets are shown, the holding durations and the enabling durations principle. In the further sections the extensions to Coloured and Stochastic Petri Nets are illustrated. Petri Nets offers a lot of possibilities to analyse their structure but this work will focus on the dynamic behaviour and on the simulation of Petri Nets.

2.2 Place-Transition Petri Nets

2.2.1 Introduction

A Place-Transition Petri Net is a bipartite directed graph consisting of two types of nodes, called places and transitions. To simplify matters, Place-Transition Petri nets are abbreviated with the term PN in the following chapters and sections. Further elements of a PN are arcs which represent the connection between places and transitions. Arcs can only connect a place to a transition or vice versa, but they must not connect two places or transitions. Tokens represent the specific value of the conditions held in each place and all these values are summarised in the marking function.

Arcs can have weights which mean the number of involved tokens are needed at least for enabling or the number of tokens which are created. If the weight of an arc is greater than one, the number of the weight will be shown along the arc, otherwise this marking will be omitted (see Figure 2.1). In this work places are drawn by yellow circles, transitions are drawn by cyan rectangles, arcs are drawn by blue lines and tokens are drawn by green circles. Figure 2.1 shows a PN with objects and elements described above. The transitions of the PN control the movement of the tokens between the places and the positions of the tokens define the state of the system [2, 5].

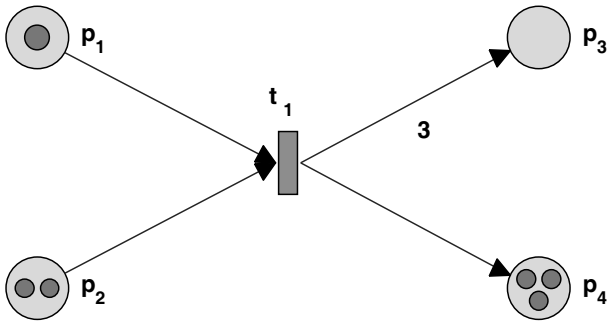


Figure 2.1: Small PN

Enabling of a transition

A transition of a PN is enabled if all its input places are marked at least with the weight of its corresponding input arcs.

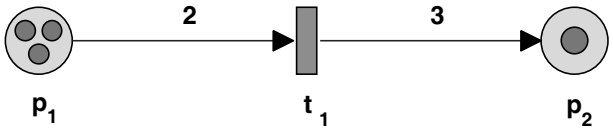


Figure 2.2: Transition t_1 is enabled

Firing of an enabled transition

An enabled transition may fire. If a transition fires, it removes tokens from input places and creates tokens in the output places corresponding to the weight of the involved arcs.

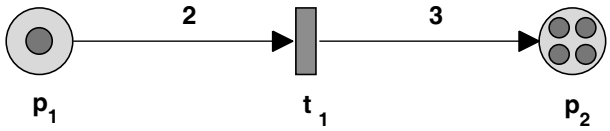


Figure 2.3: Transition t_1 fired

2.2.2 Definition

A PN with u places and v transitions can be represented by the multiple

$$PN = (P, T, I, O, M_0)$$

where

- $P = \{p_1, p_2, \ldots, p_u\}$ is the set of places;
- $T = \{t_1, t_2, \ldots, t_v\}$ is the set of transitions;
- $I : P \times T \rightarrow \mathbb{N}$ is the input arc function. If there exists an input arc with weight k connecting p_i to t_j, then $I(p_i, t_j) = k$
- $O : P \times T \rightarrow \mathbb{N}$ is the output arc function. If there exists an output arc with weight k connecting t_j to p_i, then $O(p_i, t_j) = k$
- $M_0 : P \rightarrow \mathbb{N}$ is the initial state

A transition t_i is enabled by a given marking if, and only if $M(p_j) \geq I(p_j, t_i)$ for all $p_j \in P$ [2, 5].

2.2.3 Linear Algebraic Representation

The input arc function I and the output arc function O are defined through the weight of the arcs. For each function an adjacency matrix can be built between places and transitions:

$$I = \begin{pmatrix} p_1t_1 & \ldots & p_1t_v \\ \vdots & \ddots & \vdots \\ p_ut_1 & \ldots & p_ut_v \end{pmatrix} \qquad p_it_j = \begin{cases} k & \text{if } k \text{ is weight of arc } p_i \text{ to } t_j \\ 0 & \text{otherwise} \end{cases}$$

$$O = \begin{pmatrix} t_1p_1 & \ldots & t_vp_1 \\ \vdots & \ddots & \vdots \\ t_1p_u & \ldots & t_vp_u \end{pmatrix} \qquad t_ip_j = \begin{cases} k & \text{if } k \text{ is weight of arc } t_i \text{ to } p_j \\ 0 & \text{otherwise} \end{cases}$$

The flow of each transition is now defined through the columns of the matrix

$$A = O - I$$

If the transition t_i fires, then the new marking is given by:

$$M(p_j) = M_0(p_j) + A(p_j, t_i) \quad \forall p_j \in P$$

In general, the so called firing vector u is calculated. This binary vector has the length of the number of transitions and it contains for each transition the information if the transition fires or doesn't fire. The new marking of the PN is given by the following matrix multiplication:

$$M = M_0 + A \cdot u$$

The enabling of the transitions depends only on the input matrix of the Petri Net and on the current marking of the places. The firing of the transitions can be additionally influenced by other conditions.

2.2.4 Conflict - Concurrency - Confusion

One of these conditions can be a conflict. A conflict can happen between two or more transitions if at least two transitions have the same input places. Figure 2.4 shows a small PN with a conflict. Transitions t_1 and t_2 are both enabled but the firing of one transition disables the other transition. Therefore a conflict resolution is necessary. Also, parallel activities or concurrency can easily be modelled with a

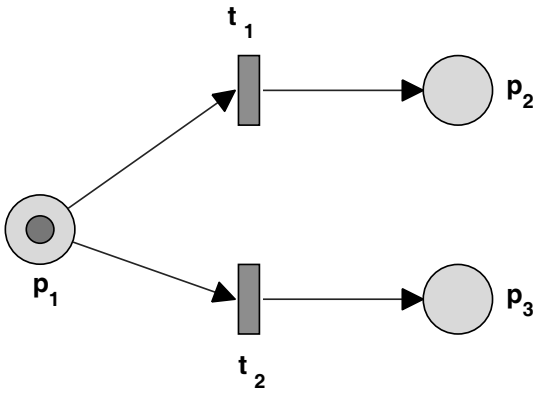


Figure 2.4: Conflict of two transitions

PN. Two events are parallel if both events can occur in any order without conflicts. If conflict and concurrency are mixed up this is called confusion (see Figure 2.5)

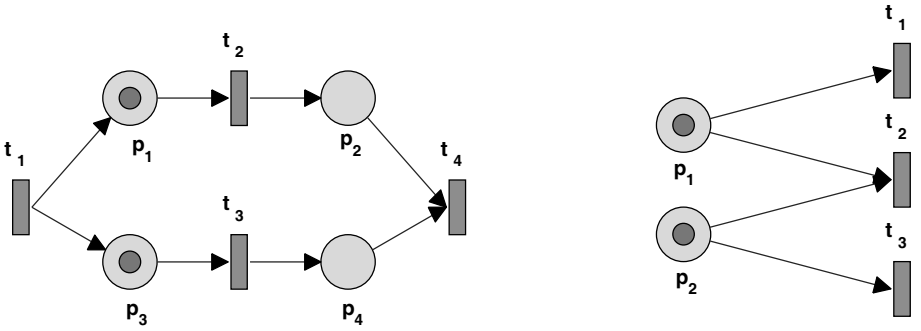


Figure 2.5: Concurrency and confusion

2.2.5 Dynamic Behaviour - Simulation

The dynamic behaviour is determined by the enabling and firing of the transitions. The initial marking M_0 is the start point of the simulation. The next marking M_{k+1} is always dependent on the current marking M_k and on the firing vector u_k:

$$M_{k+1} = M_k + A \cdot u_k$$

The changing of the markings in each state is called the *token game*. A conflict between two or more transitions is resolved randomly. This means each transition has the same probability for firing.

2.2.6 Important Properties

Deadlock

A deadlock represents the end of the simulation. If no transition can fire any more the simulation stops and a deadlock is occurred.

Reachability Set - Reachability Graph

Starting from the initial marking M_0 it is possible to compute the set of all markings reachable from it. This set is called the reachability set ($M(PN)$) and represents the state space of the system. The reachability set contains no information about the transition sequences fired to reach each marking. This information is contained in the reachability graph, where each node represents a reachable state [6]. Figure 2.6 shows an example of a reachbility graph with five different markings.

Reversibility

An important reachability property is reversibility: a PN is said to be reversible if and only if from any state reachable from M_0, it is possible to come back to M_0

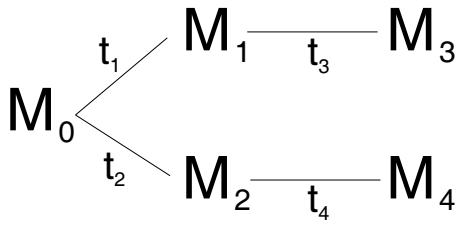


Figure 2.6: Example of reachability graph

itself. Reversibility expresses the possibility for a PN system to come back infinitely often to its initial marking. It is however possible that the state to which we want to be always able to come back to is not the initial marking of the PN. This special marking is called a home state for the PN [6].

Liveness

A PN is said to be live for an initial marking if all transitions can fire an infinite number of times. A very important consequence of liveness is that, if at least one transition is live, then the PN cannot have any deadlock.

Boundedness

A PN is said to be bounded for an initial marking if the number of tokens in all places for all markings of the reachability set is finite. This means that $\forall p \in P : \exists k \in \mathbb{N}_0 : \forall M \in M(PN) : M(p) \leq k$

2.3 Timed Petri Nets

2.3.1 Introduction

Place-Transition Petri Nets offer the analysis of their structure and they build the basis for any extensions. The concept of time is not explicitly given in the original definition of Petri nets and maybe not in the means of the inventor. But however, for the performance evaluation of dynamic systems and scheduling problems it is necessary to introduce time delays. Given that a transition represents an event, it is natural that time delays should be associated with transitions. Time delays may be either deterministic or stochastic.
As described in [7], there are three basic ways of representing time in Petri nets: firing durations, holding durations and enabling durations. The names given to Petri nets augmented with time vary greatly from one researcher to another. In this work the holding durations and enabling durations principles are implemented.

2.3.2 Holding Durations

The principle of holding durations works by classifying tokens into two types, available and unavailable. Available tokens can be used to enable transitions, whereas unavailable tokens cannot. To each transition a time duration is assigned, and when firing occurs, the action of removing and creating tokens happens instantaneously. However, the created tokens are not available to enable new transitions until they have been in their output place for the time specified by the transition that created them. Figure 2.7 shows the firing of a Timed Petri Net with holding durations. Available and unavailable tokens are drawn by small green and small unfilled circles, respectively.

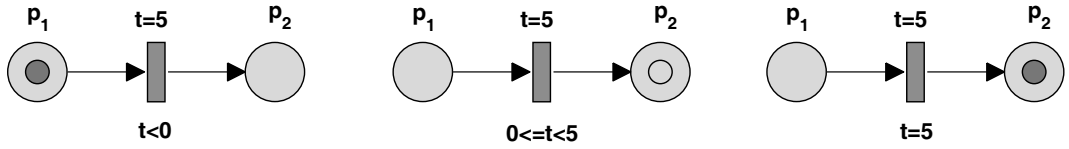


Figure 2.7: TPN with holding durations

Definition

By using holding durations the formal representation of the Timed Petri Net is extended with the information of time. A TPN with u places and v transitions can be represented by the multiple

$$TPN = (P, T, I, O, f, s_0)$$

where

- $P = \{p_1, p_2, \ldots, p_u\}$ is the set of places;
- $T = \{t_1, t_2, \ldots, t_v\}$ is the set of transitions;
- $I : P \times T \to \mathbb{N}$ is the input arc function. If there exists an input arc with weight k connecting p_i to t_j, then $I(p_i, t_j) = k$
- $O : P \times T \to \mathbb{N}$ is the output arc function. If there exists an output arc with weight k connecting t_j to p_i, then $O(p_i, t_j) = k$
- $f : T \to \mathbb{R}_0^+$ is the time delay function which is a constant function and assigns a nonnegative real value $f(t_j)$ to each transition $t_j \in T$.
- s_0 is the initial state

A state of a Timed Petri Net is a triple of functions, one of which describes the distribution of available tokens over the places, the second the distribution of unavailable tokens and the third one is the remaining-holding-time function [8]. A state of a Timed Petri Net is a triple $s = (m, n, r)$ where,

- $m : P \to \mathbb{N}$ is a marking function of available tokens. m defines an $u \times 1$ column vector whose jth entry is $m(p_j)$.
- $n : P \to \mathbb{N}$ is a marking function of unavailable tokens. n defines an $u \times 1$ column vector whose jth entry is $n(p_j)$.
- r is the remaining-holding-time function which assigns the remaining holding time (local clock) to each independent unavailable token in a place, i.e., if the rank of unavailable tokens in a place p_j is equal to l, $n(p_j) = l$, the remaining-holding-time function $r(p)$ defines a vector of l nonnegative real values denoted by $r(p_j) = [r(p_j)[1], r(p_j)[2], \ldots, r(p_j)[l]]$; r is a partial function and it is undefined for all those places for which $n(p_j) = 0$.

A transition t_i is now enabled by a given marking if, and only if $m(p_j) \geq I(p_j, t_i)$ for all $p_j \in P$.

Simulation

The initial state $s_0 = (m_0, n_0, r_0)$ is the starting point of the simulation. Depending on the initial available token function m_0 the firing vector u is computed. If any transition fires, the firing is considered instantaneous. Newly produced tokens receive the value of their local clock as prescribed with transitions that produced them. If no transition can fire at the current state, the global time is increased and the values of the local clocks are decreased respectively, by the minimum of the remaining-holding-time function r. As a consequence at least one of the unavailable tokens is becoming available again and the enabling condition is checked once more. The global time is always increased by the minimum of the remaining-holding-time function. Therefore the simulation function is implemented as event handler. A new state s_{k+1} is computed depending on the current state s_k in the following way:

1. Compute firing vector u_k depending on m_k
2. If no transition fires ($u_k = 0$) time passes on and the local clocks are decreased:
   - $M_k = m_k + n_k$
   - $r_{k+1} = r_k - \min r_k$
   - $n_{k+1}(p_i) = dim(r_{k+1}(p_i)) \quad \forall\, p_i \in P \wedge \exists\, r_{k+1}(p_i)$

- $m_{k+1} = M_k - n_{k+1}$

3. Else ($u_k \neq 0$), time remains the same. New tokens are created with the corresponding local clocks.

   - $M_k = m_k + n_k$
   - $M_{k+1} = M_k + A \cdot u_k$
   - $n_{k+1}(p_i) = n_k(p_i) + O(p_i, t_j) \cdot u_k(t_j) \quad \forall p_i \in P \wedge \forall t_j \in T : f(t_j) > 0 \wedge u_k(t_j) > 0$
   - $r_{k+1}(p_i) = r_k(p_i); r_{k+1}(p_i)[l] = f(t_j) \quad l = n_k(p_i) + 1 \ldots n_{k+1}(p_i) \; \forall p_i \in P \wedge \forall t_j \in T : f(t_j) > 0 \wedge u_k(t_j) > 0$
   - $m_{k+1} = M_{k+1} - n_{k+1}$

2.3.3 Enabling Durations

The enabling duration principle was introduced by Merlin [9] to model recoverable communication protocols. With enabling durations the firing of the transitions happens immediately and the time delays are represented by forcing transitions to be enabled for a specified period of time before they can fire [7]. Figure 2.8 shows the firing of a Timed Petri net with enabling durations. The main advantage of enabling durations can be seen in a Petri Net where confusion appears. It is possible that one transition which has begun to be enabled can be interrupted by another transition in some circumstances.

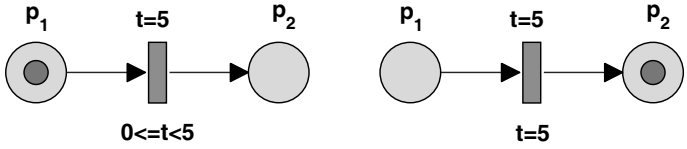


Figure 2.8: TPN with enabling durations

Memory

An important issue that arises at every transition firing when timed transitions are used in a model is how to manage the timers of all the transitions that do not fire. From the modelling point of view, the different policies that can be adopted link the past history of the systems to its future evolution, considering various ways of retaining memory of the time already spent on activities. The question concerns the memory policy of transitions, and defines how to set the transition timers when a

state change occurs, possibly modifying the enabling of transitions [6]. To model the different behaviours arising in real systems, two different ways of keeping track of the past are possible:

- **Enabling Memory** At each transition firing, the timers of all the timed transitions that are disabled are restarted, whereas the timers of all the timed transitions that are not disabled hold their present value. The memory of the past is recorded with an enabling memory variable associated with each transition. The enabling memory variable accounts for the work performed by the activity associated with the transition since the last instant of time its timer was set. In other words, the enabling memory variable measures the enabling time of the transition since the last instant of time it became enabled [6].

- **Age Memory** At each transition firing, the timers of all the timed transitions hold their present values. The memory of the past is recorded with an age memory variable associated with each timed transition. The age memory variable accounts for the work performed by the activity associated with the transition since the time of its last firing. In other words, the age memory variable measures the cumulative enabling time of the transition since the last instant of time when it fired [6].

Multiple Enabling

Special attention must be paid to the timing semantics in the case of timed transitions with an enabling degree larger than one. Borrowing from queueing network terminology, we can consider the following different situations [6].

1. **Single-server semantics:** a firing delay is set when the transition is first enabled, and new delays are generated upon transition firing if the transition is still enabled in the new marking. This means that enabling sets of tokens are processed serially.

2. **Infinite-server semantics:** every enabling set of tokens is processed as soon as it forms in the input places of the transition. Its corresponding firing delay is generated at this time, and the timers associated with all these enabling sets run down to zero in parallel. Multiple enabling sets of tokens are thus processed in parallel.

3. **Multiple-server semantics:** enabling sets of tokens are processed as soon as they form in the input places of the transition up to a maximum degree of parallelism (say k). For larger values of the enabling degree, the timers

associated with new enabling sets of tokens are set only when the number of concurrently running timers decreases below the value of k.

Definition

By using enabling durations the formal representation of the Timed Petri Net TPN is equal to the holding durations definition. But a state is now a pair of functions, one of which describes the distribution of tokens over the places and one for the enabling time function for each transition and each enabling. A state of a Timed Petri net is a pair $s = (m, r)$ where,

- $m : P \rightarrow \mathbb{N}$ is a marking function of the tokens. m defines an $u \times 1$ column vector whose jth entry is $m(p_j)$.
- r is the enabling time function which assigns the remaining time (local clock) to each enabling of each transition depending on the enabling degree and on the server semantics k of each transition, i.e., if the degree of transition t_i and the server degree are both equal to l, $deg(t_i) = l$, the enabling time function $r(t)$ defines a vector of l nonnegative real values denoted by $r(t_i) = [r(t_i)[1], r(t_i)[2], \ldots, r(t_i)[l]]$; r is a partial function and it is undefined for all those transitions which are not enabled: $deg(t_i) = 0$.

A transition t_i is now enabled by a given marking if, and only if $m(p_j) \geq I(p_j, t_i)$ for all $p_j \in P$.

Simulation

The initial state $s_0 = (m_0, r_0)$ is the starting point of the simulation. Depending on the initial token function m_0 the enabling vector is computed. For all transitions and for each enabling corresponding to the enabling degree and the server semantics the new local clock is set. If any transition fires, the firing is considered instantaneous. For each transition which fires all disabled transitions are checked if they are interrupted. If no transition can fire at the current state, the global time is increased and the values of the local clocks are decreased respectively, by the minimum of the enabling time function r. Now the enabling condition is checked once more and the local clocks are updated again. As a consequence at least one enabled transition can fire and the next iteration is started. Also the enabling duration principle function is implemented as an event handler. A new state s_{k+1} is computed depending on the current state s_k in the following way:

1. Compute enabling vector depending on m_k
2. Compute the degrees of enabling - $degree(t) \quad \forall T$

3. $deg(t_i) = \min(degree(t_i), k(t_i)) \quad \forall t_i \in T \quad k \ldots$ maxium degree

4. $r_{k+1}(t_i) = r_k(t_i)$; $r_{k+1}(t_i)[l] = f(t_i) \qquad l = dim(r_k(t_i)) + 1 \ldots deg(t_i)$
   if $dim(r_k(t_i)) < deg(t_i) \; \forall t_i \in T$

5. Compute firing vector u_k: $u_k(t_i) = 1 \qquad if \min(r_{k+1}(t_i)) = 0$

6. If no transition fires ($u_k = 0$) time passes on and the local clocks are decreased: $r_{k+1} = r_k - \min r_k$

7. Else ($u_k \neq 0$), time remains the same. Enabling time of firing enablings are deleted and disabled enablings are reset if it is necessary.

   - Delete first of $r_k(t_i)[l]$ which $r_k(t_i)[l] = 0$
   - Compute new degree for disabled transitions $degnew(t_j)$
     - Enabling memory: delete $r_k(t_j) \quad \forall t_j \in T : degnew(t_j) < deg(t_j)$
     - Age memory: delete $r_k(t_j)[degnew(t_j) \ldots deg(t_j)] \quad \forall t_j \in T : degnew(t_j) < deg(t_j)$
   - $m_{k+1} = m_k + A \cdot u_k$

2.3.4 Comparison Holding - Enabling

The main difference can be seen in a TPN where confusion appears (Figure 2.9). If holding durations are used, transition t_1 and t_3 fire concurrently at time point zero, placing one token in place P_2 and one in P_5. The token in P_2 becomes available at time point one and the token in P_5 at time point five. No more transitions are enabled. Figure 2.9 (b) shows the holding duration TPN in its final state. If enabling durations are used, initially transitions t_1 and t_3 are enabled and t_1 will fire after one time unit while t_3 will fire after five. Thus, at time point one, t_1 fires placing a token in P_2, enabling transition t_2 to fire in two time units, that is, at time point three. Transition t_3 is still enabled and scheduled to fire at time point five. Thus, t_2 fires (before t_3) at time point three, removing the tokens from P_2 and P_3, and placing a token in P_4. This disables t_3. The final state of the enabling durations TPN is shown in Figure 2.9 (c). This shows that changing the timing policy of the TPN causes a dramatic change to the way the net executes. It is the interruption to the enabling of t_3 in the case of enabling durations that changes the outcome. It was the requirement to model such interruptions in systems with timeouts and where preempting is possible that led to the development of enabling durations [7].
It is is easy to model holding durations using enabling transitions if immediate transitions are used. Immediate transitions have zero enabling durations and always fire

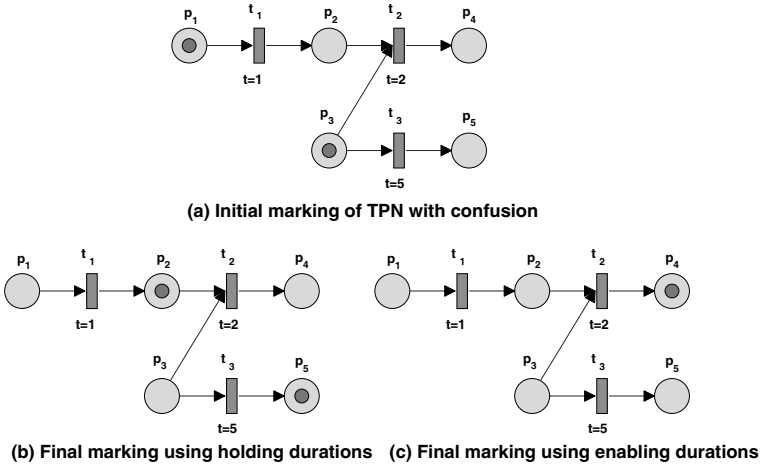


Figure 2.9: Difference between holding and enabling durations

before transitions with nonzero enabling durations. Thus, to represent an holding durations TPN with enabling durations, each transition is preceded by an immediate transition [7]. But this works only for enabling transition with infinite-server semantics. Figure 2.10 shows the enabling durations TPN which is equivalent to the holding durations TPN shown in Figure 2.9 (b).

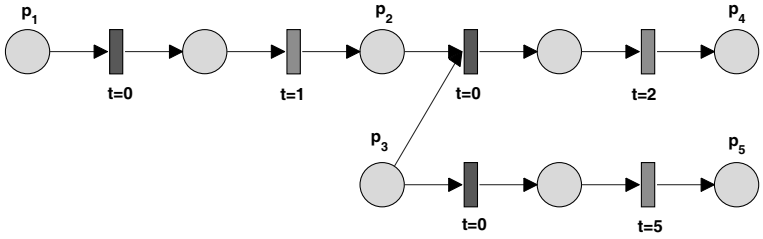


Figure 2.10: Representing holding with enabling times

It is natural to use holding durations in modelling most processes as transitions represent events and generally, once an event starts it does not stop to allow another event to occur. Consequently, the kind of interruption modelled by enabling durations is not applicable for many systems. However, since enabling durations can act like holding durations when immediate transitions and transitions with infinite-server-semantics are used and allow the modelling of interruptions, they give greater modelling flexibility [7].

2.4 Coloured Petri Nets

2.4.1 Introduction

The graphical representation of Petri Nets becomes fairly complex due to modelling real life problems. The main reason is that there is only one type of token. Each process has to be modelled by a separate sub net. There exists a lot of extensions and different types of high level Petri Nets [10]. Coloured Petri Nets ($CPNs$) were first introduced by K. Jensen [11]. In the definition of Coloured Petri nets a type called the colour is attached to a token. This aspect leads to a simplification of the graphical representation of a Petri Net model of a complex system [2]. Figure 2.11 shows the graphical simplification of three sub nets of a PN to one CPN.

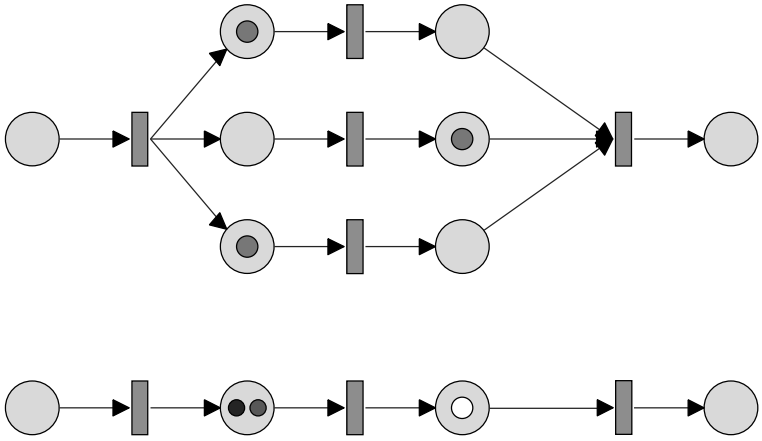


Figure 2.11: Simplification with CPN

2.4.2 Definition

For the definition of $CPNs$ it is needed to define a multi-set. A multi-set m over a non-empty set S, is a function $m : S \rightarrow \mathbb{N}_0$. The non-negative integer $m(s) \in \mathbb{N}_0$ is the number of appearances of the element s in the multi-set m. The defnition of the

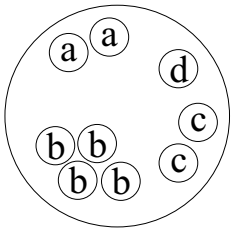


Figure 2.12: A multi-set m

multi-set m in figure 2.12 is fiven by

$$m(s) = \begin{cases} 2 & \text{if } s \in \{a, c\} \\ 4 & \text{if } s = b \\ 1 & \text{if } s = d \end{cases}$$

S_{MS} define the set of all finite multi-sets over S.

A CPN with u places and v transitions can be represented by the multiple

$$CPN = (P, T, C, I, O, M_0)$$

where

- $P = \{p_1, p_2, \ldots, p_u\}$ is the set of places;
- $T = \{t_1, t_2, \ldots, t_v\}$ is the set of transitions;
- $C : P \cup T$ into finite and non-empty sets is the colour function for places and transitions
- $I : C(t) \rightarrow C(p)_{MS}$ is the input arc function $\forall (p, t) \in P \times T$.
- $O : C(t) \rightarrow C(p)_{MS}$ is the output arc function $\forall (p, t) \in P \times T$
- M_0 is the initial state

A transition $t \in T$ is now enabled by a given marking with respect to a colour $c' \in C(t)$ if, and only if $M(p)(c) \geq I(p, t)(c')(c)$ for all $p \in P$ and $c \in C(p)$.

2.4.3 Simulation

Every CPN can be easily unfolded uniquely into a PN so that all concepts relating to $CPNs$ are consistent with those of PNs [2]. Therefore the extensions to a Timed CPN with holding or enabling durations can be done in an analogue way. The four dimensional input and output functions can be transformed into two dimensional matrices by adding each transition colour as a column and by adding each place colour as a row. Also simulation and analysis can be done on the unfolded Petri Net. It should be clear that $CPNs$ are only a means of simplifying the graphical representation of a Petri Net model of a complex system.

2.5 Stochastic Petri Nets

2.5.1 Introduction

Up to now randomness doesn't play an important role for Petri Nets. Only the conflict resolution is decided randomly. Stochastic Petri Nets use random numbers and stochastic distributions to control and model nondeterministic behaviour. In the non-timed case the firing of the transitions can be controlled by a local or global firing probability [2]. This means that it is randomly decided if an enabled transition can fire or not. For Timed Petri Nets the randomness is put into the time delays of the transitions. To each transition an stochastic distribution can be defined. Depending on the selected distributions different definitions of Stochastic Petri Nets exists. Stochastic Petri Nets (SPN) are characterised by exponential distributed firing times or by exponential distributed firing probabilities [2, 12]. Generalised Stochastic Petri Nets ($GSPN$) [6] allows additionally immediate transitions having no time delay and firing without loosing time. Another extension are Deterministic Stochastic Petri Nets ($DSPN$) [6] where deterministic time delays are also used. If no constraints are made for the distributions of the firing time of the transitions this type is called Extended Stochastic Petri Net ($ESPN$) [13]. Depending on the type of the Stochastic Petri Nets and the used distributions the reachability set and the reachability graph can be modelled as Markov Chain, Markov Process, Semi-Markov Process or in general as Stochastic Process [2, 12, 13, 14, 15].

2.5.2 Definition

Non-timed

In this work a global firing probability can be defined for non-timed simulation. To each enabled transition a uniform distributed random value between zero and one is assigned in each iteration. If the assigned values are greater than the defined firing probability (value between zero and one in percent) the enabled transitions are not allowed to fire in this iteration. If the values are smaller than or equal to the firing probability the enabled transition will fire. With this global firing probability stochastic behaviour can be modelled for non-timed Petri Nets.

Timed

$TPNs$ can be extended to each type of Stochastic Petri Net due to the use of proper time delay functions. In this work each kind of Stochastic Petri Nets is abbreviated by SPN. The time delay function is now re-defined for $SPNs$ in the following way:

- $f : T \rightarrow \mathbb{R}_0^+$ is the time delay function which can be an arbitrary stochastic distribution function or a constant function (for deterministic and immediate transitions). It assigns for each firing and for each realisation a non-negative real value $f(t_j)$ to each transition $t_j \in T$.

If the evaluation of a stochastic distribution leads to a negative value or if a constant function with a negative value is defined, the used values for the simulation are set to zero.

Chapter 3

Scheduling Problems and Petri Nets

3.1 Introduction

Scheduling deals with the allocation of resources to activities over time, by respecting precedence, duration, capacity and incompatibility constraints, in order to achieve the optimal use of resources or the optimal accomplishment of tasks. Scheduling involves the arrangement, coordination, and planning of the utilisation of resources to achieve an objective. Of the resources available, time is becoming an important commodity. Time is the resource most often planned and is present in all scheduling. In its simplest form, the overall time cycle required for production or completion is the most usual scheduling situation [16]. Scheduling problems arise in domains like manufacturing, transportation, computer processing and production planning. Many well defined problems like job-shop, flow-shop problem or scheduling used in flexible assembly systems can be found in literature [17, 18].
There exists special classifications of scheduling problems [19, 20, 21] which leads to different approaches of modelling and algorithms. This work provides a general framework for modelling, simulation, analysis and optimisation of scheduling problems based on Timed Petri Nets. No special definitions are given for the scheduling problem besides of some restrictions for the underlying Timed Petri Net. The sequence orders are the only parameters used. All other conditions and specifications are determined by the basic properties of Timed Petri Nets. Simulation is an appropriate tool for evaluating and analysing scheduling problems based on Timed Petri Nets to get the needed performance measures [22]. With respect to the optimtisation problem this work is focused on the overall cycle time as main interest of the performance measures. But using Petri Nets offers a lot of possibilities to get other interesting parameters of the system. For example, utilisation and allocation of the resources can be easily measured and shown by Gantt Charts.

Petri Nets can represent many different applications by the use of their simple elements: places, transitions and tokens [23, 24]. Places stand for states of the system, transitions represent events, tokens realise conditions and arcs control the logical behaviour of the system. Used additionally, time in Petri Nets extends the possibilities of modelling in an essential way. For example, Manufacturing systems and production activities can be modelled with Timed Petri Nets [25, 26, 27, 28]. Now places correspond to resources, transitions represent operations and tokens can be products or in general entities of the system. It is up to the modeller of the system to define and to assign a meaning to the different elements of the Timed Petri Net.
This chapter shows how Timed Petri Nets can be extended by priorities and sequences to get the capability of modelling scheduling problems. Another important part is the exact definition and the restrictions for the TPN in respect of modelling scheduling problems. Further properties like conflict resolution, deadlock and reachablility are treated in the following chapter.

3.2 Definition

The class of scheduling problems which can be modelled as Timed Petri Nets is defined by the following criteria:

- **same initial marking** - Each scheduling problem has to start with the same initial marking. This means that different system configurations can only realised by changing the sequence parameters.
- **sequences assigned to transitions** - The sequence parameters are assigned to the transitions and they correspond to the firing sequences of the transitions.
- **conflicts solved** - All conflicts have to be resolved and determined at the beginning.
- **bounded** - The TPN has to be bounded.
- **not reversible** - The TPN must not be reversible. In this case reversibility is extended with the defined firing sequences. This means that it is not allowed to come back to a home state including the current state and position of sequence parameters.
- **all transitions not live** - No transition is allowed to be live.
- **terminating simulation** - Because of the previous criterion the simulation of the Timed Petri Net is terminating.

3.3 Conflict Resolution

All possible conflicts are determined through the structure of the underlying Petri Net. Input places that are connected to more than one transition allow the occurrence of a conflict depending on the current marking. Conflicts are processed and resolved in an arbitrary manner for each iteration of the simulation function. This should prevent the equal appearance of confusion during the simulation. Transitions get a ranking value between zero and one depending on the selected conflict resolution strategy. This ranking value defines which transition can fire first. Starting with the highest value each transition fires step by step until no transition of the selected conflict is enabled any more. If more than one transition has the same ranking value in the same step the winner of the conflict is randomly decided.
A conflict resolution is necessary if at least two transitions are in conflict. For Petri Nets without time there is the possibility of random decision based on the uniform distribution. Timed Petri Nets offer other approaches of solving the conflict between transitions. Figure 3.1 shows a small Timed Petri Net where a conflict occurs. The two different principles of time implemented to Petri nets leads to a different behaviour during a conflict. In this section the holding durations and enabling durations are compared in relation to solving conflicts.

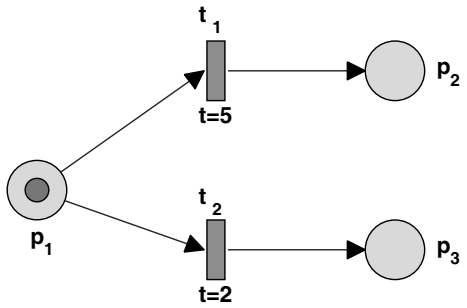


Figure 3.1: Conflict Resolution

3.3.1 Holding

Transitions working with the holding durations principle fire immediately after enabling. Therefore next to the random conflict resolution the following two strategies can be used for transitions with deterministic time delays:

- **SPT** - The transition with the shortest processing time fires first. In figure 3.1 transition t_2 has the shortest time delay. Thus, the conflict is decided in favour of transition t_2.

- **LPT** - The transition with the longest processing time fires first. Transition t_1 of figure 3.1 will fire because it has the longest processing time delay.

These two strategies only work for transitions with deterministic time delays. Using holding durations the time delays are assigned during the firing of the transition. If stochastic time delays are used the underlying stochastic distribution functions are newly evaluated in each firing. Therefore conflicts can not be decided beforehand for stochastic time delays using SPT or LPT strategy.

3.3.2 Enabling

The enabling durations principle works by forcing the transitions to be enabled for a certain time delay. It can be distinguished between immediate transitions without time and transitions with time delays greater than zero. For immediate transitions and transitions with equal time delays random decision is used. Conflicts between transitions with different time delays are solved by the so called race policy. In this case always the transition with the shortest time delay wins. The race policy also works for stochastic time delays because for enabling durations the time delays are assigned before the firing of the transitions.

3.4 Priority

The resolution of all conflicts is an important criterion of scheduling problems modelled as Timed Petri Nets. For this purpose a priority ranking can be established for the firing of the transitions. A priority value can be set to each transition. This value has to be greater or equal than one and defines the priority of the transition. The highest priority is given to 1. If there is a conflict between at least two transitions the transition with the highest priority will always fire. If two or more transitions have the highest priority value it will be randomly decided which transition fires.

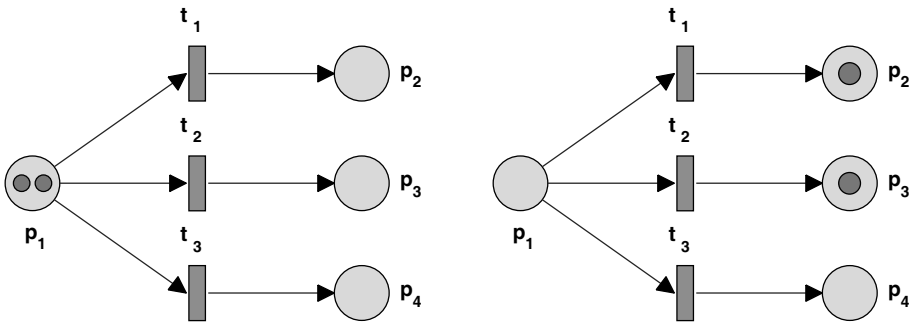


Figure 3.2: Petri Net with priority

Figure 3.2 shows a small Petri Net with defined priorities before and after firing. In this example transition t_1 has the highest priority equal to 1 and transitions t_2 and t_3 have both the priority equal to 2. Place P_1 contains two tokens and therefore a conflict occurs because all three transitions are enabled but only two of them can fire. Transition t_1 fires in any case. Transitions t_2 and t_3 have the same priority and the firing is randomly decided. In this case transition t_2 fires but it is also possible that transition t_3 fires instead of t_2 after restarting the simulation.

3.5 Sequence

Next to the definition of a priority Petri Nets need another important extension to get the capability of modelling scheduling problems. The main input parameters of scheduling problems are represented by the sequences of different tasks. A change of the input parameters leads to a different system configuration and to different results. These sequences can be directly implemented and defined to the transitions of the Petri Net. Disjoint groups of transitions can be selected and to each group a firing rule can be assigned. The transitions are numbered within the group starting from 1 to the selected group length. Now a firing list can be defined for the group of transitions. This list consists of the assigned numbers of the transitions. The values of the list correspond to the firing of the transitions. All transitions of the group are deactivated instead of the transition represented by the first number of the firing list. This means that all deactivated transitions are not able to be enabled by any tokens of the input places. After the firing of the selected transition the next value of the list is taken. If the end of the firing list is reached all transitions are activated again and they behave like normal transitions not controlled by a firing list.

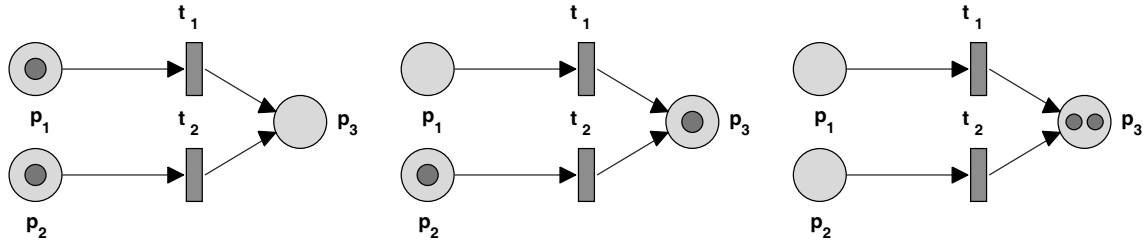


Figure 3.3: Petri Net with sequence

Figure 3.3 shows the firing of a Petri Net with a defined sequence. In this example the sequence $(1, 2)$ is assigned to the transitions t_1 and t_2. In the left Petri Net transition t_1 is enabled and transition t_2 is deactivated due to the firing list. The middle Petri Net shows the marking after firing of transition t_1. Transition t_2 is activated again and now enabled whereas transition t_1 is deactivated in this step. The final marking

is shown in the right Petri Net of figure 3.3 after firing of transition t_2.
On the one hand transitions can be selected to assign priorities and on the other hand transitions can be added to groups to define firing lists. These two possibilities are disjoint. This means that either a transition can have a priority or a transition can be member of a sequence group. If a priority is needed for sequence transitions a sequence priority can be defined. This sequence priority controls the behaviour of conflicts between transitions of different sequence groups. Priorities are necessary if conflicts between transitions should be solved in a special way. The following ranking and hierarchy is used for the interacting of the different kinds of transitions:

1. **Sequence:** Transitions which are members of a sequence group always have the highest priority. Conflicts inside sequence transitions are determined by the sequence priority.

2. **Priority:** Transitions with priority get the second highest priority.

3. **Normal:** Transitions without any priorities or sequences are defined as normal transitions. Conflicts of normal transitions are solved through the strategies mentioned in the previous section.

3.6 Deadlock

The termination of the simulation is an important and an essential criterion for modelling scheduling problems based on Timed Petri Nets. Depending on the selected input sequences each system configuration leads to different simulation results. The

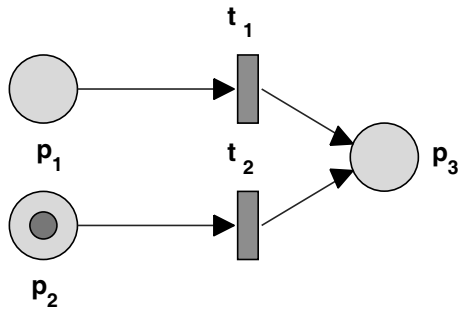


Figure 3.4: Sequence deadlock

end of a simulation corresponds to a final marking of the reachability graph. The final markings are reached during the simulation through a desired natural deadlock. Because of the sequence extensions of Timed Petri Nets a new kind of deadlock can occur during the simulation. If the transition activated due to the firing list is not

able to be enabled and no other transition is enabled in the current simulation step a sequence deadlock happens. This sequence deadlock represents an invalid system configuration and it can be separately detected.
Figure 3.4 shows this new kind of deadlock. Transition t_1 and t_2 are added to a sequence group containing the firing list $(1, 2)$. Transition t_1 is activated but not enabled and therefore a sequence deadlock occurs and the simulation stops.

3.7 Reachability

The reachability graph contains all possible markings and ways through the underlying Petri Net. The extension of defining sequences to Petri Nets may lead to a different reachability set and graph for Timed Petri Nets modelling scheduling problems. The alternate activating and deactivating of transitions temporary change the natural behaviour of the Petri Net. Starting with the same initial marking each system configuration finds its own way through the reachability graph during the simulation. The changing of the input sequence can also modify parts of the reachability graph. But the simulation has to be used to create each reachability graph for all possible input parameters. Therefore it is very time-consuming to compute all possible markings beforehand.

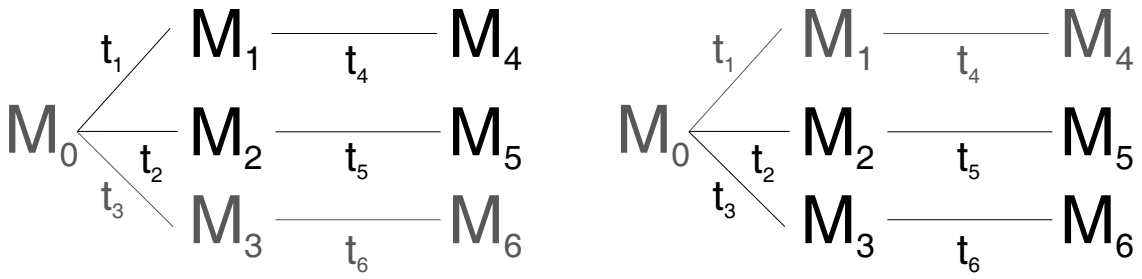


Figure 3.5: Way through reachability graph

Figure 3.5 schematises the way through the reachability graph for two different input sequences. The markings and transitions used are coloured red in this case. Timed Petri Nets need exact definitions and constraints for modelling scheduling problems. By means of these criteria the way through the reachability graph is determined for each system configuration. The use of deterministic or stochastic time delays highly influence the way through the reachability graph for each input sequence. The two different types of time delays are discussed in the following sections in relation to the holding and enabling duration principles.

Deterministic

If only deterministic time delays are used and all criteria are fulfilled each system configuration has an unique and determined way through the reachability graph. For the holding and enabling duration principle all used times are deterministic and therefore each conflict based on time delays assigned to transitions is solved in an unique way. Timed Petri Nets adapted for scheduling problems contain no probabilistic part. Therefore, each input sequence leads to the same result in each simulation run. Two different system configurations can be directly compared with regard to the optimisation of scheduling problems.

Stochastic

For stochastic time delays each system configuration may has different ways through the reachability graph using holding durations. The markings of a Stochastic Timed Petri Net can have small modifications due to the different evaluations of the underlying stochastic distribution functions. For scheduling problems modelled with holding durations all possible conflicts are solved and therefore the difference can only be in the time of firing. On this account the visited markings of the reachability graph can be different for each simulation run of the same input sequences. A certain number of simulation runs is needed to compare two different system configurations but this problem is treated in the following chapter.
The race policy of enabling durations causes another constraint for solving conflicts if stochastic time delays are used. In the deterministic case the race policy always determines a conflict in the same way, meaning that the transition with shortest time delay can fire. For stochastic time delays this decision depends on the respective values of the used stochastic distribution function. The probabilistic controlled preempting can model a fault detection for a system but for scheduling problems it doesn't really seems useful with respect to the interesting performance measure. If the two possible ways through the rechability graph differ too much the simulation runs can lead to a great difference of the overall duration. Thus, the stochastic optimisation is negatively affected because of the increase of the variance. For modelling scheduling problems based on Timed Petri Nets this problem should be considered.

Chapter 4

Optimisation

4.1 Introduction

Scheduling problems basically belong to the field of combinatorics. A set of tasks should be ordered to build an optimum corresponding to certain constraints. An optimum can be both a maximum and a minimum. Therefore the optimisation of a scheduling problem can be a minimising or maximising problem. In this work all constraints and specifications of scheduling problems are determined by the basic properties of Timed Petri Nets. The sequence orders are the only parameters used. All possible permutations of these sequences build the solution space of the scheduling problem. In general a so called objective or fitness function is defined which assigns a certain value to each solution of the solution space. This fitness value is used to define the quality of the selected solution. In this case the objective function is determined through the underlying Timed Petri Net. The evaluation of the fitness value is performed by simulation resulting in the overall cycle time of the system. For maximising problems this value should be as great as possible and respectively for minimising problems it should be as small as possible. But both problem types are equivalent. The multiplication of the objective value of a minimising problem by -1 results in a maximising problem and vice versa. Thus, in the following only minimising problems are considered.
In principle there are two possibilities to solve such optimisation problems. On the one hand exact solutions can be computed forming an exact optimum of the selected problem. Scheduling problems belong to the class of NP-hard problems [19] and therefore exponential run-time would be needed to compute an exact solution. On the other hand it is possible to apply approximation algorithms like heuristic algorithms [19, 29]. These algorithms have polynomial run-time and produce solutions that are guaranteed to be within a fixed percentage of the actual optimum. Any approach without formal guarantee of performance can be considered a heuristic. Such

approaches are useful in practical situations if no better methods are available [19]. This chapter introduces the used heuristic algorithms. Starting with the simple local search more sophisticated approaches like simulated annealing and threshold accepting are considered. Furthermore Genetic algorithms are discussed and the deterministic optimisation is extended with the capability of due dates. Another important part is the stochastic optimisation of scheduling problems using a sequential paired t-test and variance reduction techniques.

4.2 Local Search

Local search is an iterative procedure which moves from one solution in the search space S to another as long as necessary. In order to systematically search through S, the possible moves from a solution s to the next solution should be restricted in some way. To describe such restrictions a neighbourhood structure $N : S \rightarrow 2^S$ is introduced on S. For each $s \in S$, $N(s)$ describes the subset of solutions which can be reached in one step by moving from s. The set $N(s)$ is called the neighbourhood of s. Usually it is not possible to calculate the neighbourhood structure $N(s)$ beforehand because S has an exponential size. To overcome this difficulty, a set AM of allowed modifications $F : S \rightarrow S$ is introduced. For a given solution s, the neighbourhood of s can be defined by $N(s) = \{F(s) \,|\, F \in AM\}$.
Using these definitions, a local search method may be described as follows. Each iteration starts with a solution $s \in S$ and choose a solution $s' \in N(s)$ or a modification $F \in AM$ which provides $s' = F(s)$. Based on the values of the objective function $f : S \rightarrow \mathbb{R}$, $f(s)$ and $f(s')$, the starting solution of the next iteration is chosen. According to different criteria used for the choice of the starting solution of the next iteration different types of local search techniques are arisen [19].
The iterative improvement algorithm takes the solution with the smallest value of the objective function and can be formulated as follows:

Algorithm Iterative Improvement:
begin
  choose initial solution $s \in S$
  repeat
    generate neighbour solution $s' \in N(s)$
    if $f(s') \leq f(s)$ **then**
      $s := s'$
  until $f(s') \leq f(s), \forall\, s' \in N(s)$
end

4.3 Simulated Annealing

Local search algorithms are simple to implement and quick to execute, but they have the main disadvantage that they terminate in the first local minimum which might give an objective function that deviates substantially from the global minimum. The reason why a local search algorithm terminates in the first local minimum it encounters is that only transitions corresponding to a decrease in the objective function are accepted by the algorithm. Alternatively, an algorithm should be considered which attempts to avoid becoming trapped in a local minimum by sometimes accepting transitions corresponding to an increase in the objective function. Simulated Annealing is an example of the latter approach where in addition to cost-decreasing transitions, cost-increasing transitions are accepted with a non-zero probability which decreases gradually as the algorithm continues its execution [30].
Simulated Annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. The algorithm is based upon that of Metropolis et al. [31], which was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. Kirkpatrick et al. [32] proposed in 1983 that the Metropolis algorithm forms the basis of an optimisation technique for combinatorial and other problems.
The pseudo-code of the local search algorithm is extended for the Simulated Annealing algorithm in the following way:

Algorithm Simulated Annealing:
begin
  generate initial solution $s \in S$
  $i := 0$
  repeat
    generate neighbour solution $s' \in N(s)$
    if $f(s') \leq f(s)$ **then**
      $s := s'$
    else if $min(1, exp(\frac{f(s)-f(s')}{c_i})) > random([0,1])$ **then**
      $s := s'$
    $c_{i+1} := g(c_i)$
    $i := i + 1$
  until termination criterion
end

Simulated Annealing is a method which seeks to avoid being trapped in a local minimum. It is a randomised method because:

- s' is chosen randomly from $N(s)$
- in the $i-th$ step s' is accepted with probability

$$min(1, exp(\frac{f(s) - f(s')}{c_i}))$$

where (c_i) is a sequence of positive control parameters with $\lim_{i\to\infty} c_i = 0$.

The interpretation of this probability is as follows. If $f(s') \leq f(s)$, then s is replaced by s' with probability one. If, on the other hand, $f(s') > f(s)$, then s is replaced by s' with some probability. This probability decreases with increasing i. In other words, a local minimum can be left, but the probability for doing so will be low after a large number of steps. In the Simulated Annealing algorithm $random[0, 1]$ denotes a function which yields a uniformly distributed random value between 0 and 1. Furthermore, the sequence (c_i) is created by a function g, i.e. $c_{i+1} = g(c_i)\ \forall\, i$ [19].

4.3.1 Neighbourhood Functions

The neighbourhood function contains the allowed modifications for creating neighbour solutions of a given solution based on one move or step. In the following three neighbourhood functions are introduced which are used and implemented in this work. Depending on the underlying scheduling problem each neighbourhood function can have its advantages in finding good and feasible neighbour solutions. In case of sequences with repetition a generalised version of the neighbourhood functions can be used to overcome ineffective moves. If the given solution consists of more input sequences the selected neighbourhood function will only change one randomly selected sequence.

Order-based Function

The order-based neighbourhood function selects two tasks at random and exchanges them in the sequence. In the following example task 6 and task 8 are selected and exchanged:

input sequence: 1 4 10 5 **6** 9 7 **8** 3 2
order-based: 1 4 10 5 **8** 9 7 **6** 3 2

For sequences with repetition it can happen that the same task can be selected twice. An exchange of these two selected tasks would have no effect and exactly the same solution would be created. To overcome this problem only different tasks can be

selected in the general order-based neighbourhood function. Now marked task 3 and task marked 4 are selected and exchanged:

input sequence:	1 2 1 5 **3** 3 5 **4** 6 4
general order-based:	1 2 1 5 **4** 3 5 **3** 6 4

Swap-based Function

The swap-based neighbourhood function selects one task at random and exchanges it with an adjacent task. In the following example task 9 is selected and exchanged with task 7:

input sequence:	1 4 10 5 6 **9** **7** 8 3 2
swap-based:	1 4 10 5 6 **7** **9** 8 3 2

In the generalised version of the swap-based neigbourhood function it is not possible to exchange two equal tasks. In this example marked task 1 is selected and exchanged with the marked adjacent task 5:

input sequence:	1 2 **1** **5** 3 3 5 4 6 4
general swap-based:	1 2 **5** **1** 3 3 5 4 6 4

Position-based Function

The position-based neighbourhood function deletes a randomly selected task of the sequence and puts it to a newly inserted place at an arbitrary position. In other words first two positions of tasks are selected at random. The task at the smaller position is deleted and newly inserted at the larger position. The tasks between the the two positions are shifted one position to the left. In the following example task 5 is deleted and newly inserted at the former position of task 3:

input sequence:	1 4 10 **5** 6 9 7 8 **3** 2
position-based:	1 4 10 6 9 7 8 **3** **5** 2

For sequences with repetition it can happen that the same task is repeated one after the other. If the two randomly selected positions cover a part of similar tasks the position-based neighbourhood function will have no effect. To overcome this problem a generalised version of the position-based neighbourhood function is defined. In the following example the marked task 1 is deleted and newly inserted at the former position of the marked task 6:

input sequence:	1 2 **1** 5 3 3 5 4 **6** 4
general position-based:	1 2 5 3 3 5 4 **6** **1** 4

4.3.2 Cooling Strategies

Most features in Simulated Annealing like solution space, neighbourhood functions or objective function are fixed by definition. The only feature which is variable during the calculation is the sequence of the control parameter. For Simulated Annealing this control parameter can be denoted as the temperature. Therefore one of the most important features in simulated annealing is the choice of the annealing schedule, and many attempts have been made to derive or suggest good schedules. The annealing procedure involves first "melting" the system at a high temperature, then repeatedly lowering the temperature by a constant factor α ($0 < \alpha < 1$), taking enough steps at each temperature to keep the system close to equilibrium, until the system approaches the ground state [33]. In this work three different cooling strategies are used and implemented.

Exponential

The exponential schedule is defined by

$$T(i) = T_0 \alpha^i$$

where i is the step count.

Linear

The linear schedule is defined by

$$T(i) = T_0 - \alpha i$$

where i is the step count. If the value of the linear schedule would be negative it is set to zero.

Logarithmic

The logarithmic schedule is defined by

$$T(i) = \frac{T_0}{\log(i + d)}$$

where i is the step count and parameter $d > 0$

Figure 4.1 shows the curves of the three cooling strategies. The starting temperature is set to 250. For the exponential schedule the constant factor α is set to 0.98. For the linear schedule α is equal to 1. In this example both cooling strategies yield zero or a value near to zero after 250 iterations. The logarithmic schedule will lead

the system to the global minimum state in the limit of infinite time. In this case parameter d is set to 1. If parameter $d < e - 1 = 1.71828183$ the logarithm of (i+d) is smaller than 1 in the first step. Therefore an increase of the temperature happens in the first step which can be seen in figure 4.1.

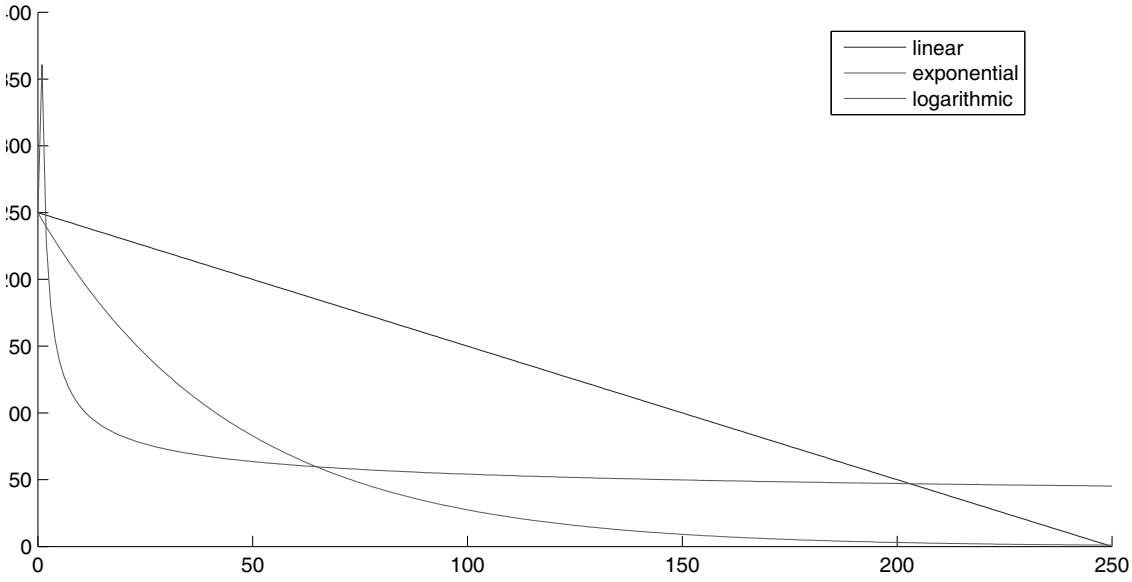


Figure 4.1: Cooling Strategies

4.3.3 Termination Criteria

The termination criterion is an important part and concept of heuristic algorithms. For all iterative algorithms the type and time of the termination is essential. It can happen that the goodness of the solution is significantly improved if more iterations are done and simulated. For Simulated Annealing the following termination or stop criteria can be chosen and the optimisation stops if one of these criteria is fulfilled:

- **Iterations** - If a certain number of iterations is reached the simulation stops.
- **Time Limit** - If the current simulated time exceeds the given time limit the simulation stops.
- **Function Limit** - If a certain value of the objective function is reached the simulation stops.

4.4 Threshold Accepting

The Threshold Accepting algorithm (TA) is one of the youngest heuristic algorithms. Dueck and Scheuer [34] proposed Threshold Accepting as a variance of Simulated Annealing in 1990. The essential difference between SA and TA consists of the different acceptance rules. TA accepts every new configuration which is not much worse than the old one whereas SA accepts worse solutions only with rather small probabilities. An apparent advantage of TA is its greater simplicity. It is not necessary to compute probabilities or to make random decisions. The pseudo-code for the Threshold Accepting algorithms can be defined in the following way:

Algorithm Threshold Accepting:
begin
  generate initial solution $s \in S$
  $i := 0$
  repeat
    generate neighbour solution $s' \in N(s)$
    if $f(s') \leq f(s)$ **then**
      $s := s'$
    else if $f(s') - f(s) < t_i$ **then**
      $s := s'$
    $t_{i+1} := g(t_i)$
    $i := i + 1$
  until termination criterion
end

A newly generated solution $s' \in N(s)$ is now accepted if the difference $f(s') - f(s)$ is smaller than some non-negative threshold t. The threshold t is a positive control parameter which is gradually reduced in an analogue way to the temperature of the Simulated Annealing algorithm. The neighbourhood functions defined in the previous section can also be used for the TA algorithm. Further, the cooling strategies for the temperature can be used for the threshold. The TA algorithm also has the same termination criteria as the SA algorithm.

4.5 Genetic Algorithms

Genetic algorithms (GA) are numerical optimisation algorithms inspired by both natural selection and natural genetics. The method is a general one, capable of being applied to an extremely wide range of problems. Genetic algorithms were developed by John Holland [35] in the late sixties. They combine survival of the fittest among string structures with a structured yet randomised information exchange to form a search algorithm with some of the innovative flair of human search. In the original definition the different solutions of the search space are represented and encoded in binary strings but also other encodings are possible. Especially for combinatorial and scheduling problems the solutions can be encoded directly. This means that the sequence of tasks can be directly used in the algorithm. In the GA lingo all used terms and elements have special names. A solution of the problem is called individual or phenotype and its representation is called genome, chromosome or genotype. In case of scheduling problems those two definitions are coincided. The search space of the selected problem is called fitness landscape and the objective function is called fitness function. Compared to Simulated Annealing or Threshold Accepting Genetic algorithms are initialised with a population of individuals which are usually random and be spread throughout the search space. A typical algorithm then uses three operators, selection, crossover, and mutation to direct the population over a series of time steps or generations towards convergence at the global optimum [36, 37]. The pseudo-code for a GA can be defined in the following way:

Algorithm GA:
begin
  $i := 0$
  generate initial population $P(i)$
  evaluate individuals in $P(i)$
  repeat
    $i := i + 1$
    select $P(i)$ from $P(i-1)$
    recombine individuals in $P(t)$
    evaluate individuals in $P(t)$
  until termination criterion
end

Genetic algorithms are suitable for problems with an unknown search space and for cases where no other methods can be used. The algorithm and its operators do not need any problem specific information. Therefore scheduling problems can be solved and optimised by the use of Genetic algorithms [38, 39, 40].

4.5.1 Fitness Scaling

Fitness scaling converts the raw fitness scores that are returned by the fitness function to values in a range that is suitable for the selection function. The selection function uses the scaled fitness values to select the parents of the next generation. The selection function assigns a higher probability of selection to individuals with higher scaled values. The range of the scaled values affects the performance of the Genetic algorithm. If the scaled values vary too widely, the individuals with the highest scaled values reproduce too rapidly, taking over the population gene pool too quickly, and preventing the Genetic algorithm from searching other areas of the solution space. On the other hand, if the scaled values vary only a little, all individuals have approximately the same chance of reproduction and the search will progress very slowly [41]. In this work the following fitness scaling functions are used:

- **Rank** - The fitness scaling function rank scales the raw scores based on the rank of each individual instead of its score. The rank of an individual is its position in the sorted scores. The rank of the most fit individual is 1, the next most fit is 2, and so on. Rank fitness scaling removes the effect of the spread of the raw scores

- **Top** - Top scaling scales the top individuals equally. The number of top individuals can be defined. Each of the selected individuals is assigned an equal scaled value, while the rest are assigned the value 0

- **Proportional** - Proportional scaling makes the scaled value of an individual proportional to its raw fitness score.

- **Shift linear** - Shift linear scaling scales the raw scores so that the expectation of the fittest individual is equal to a constant multiplied by the average score.

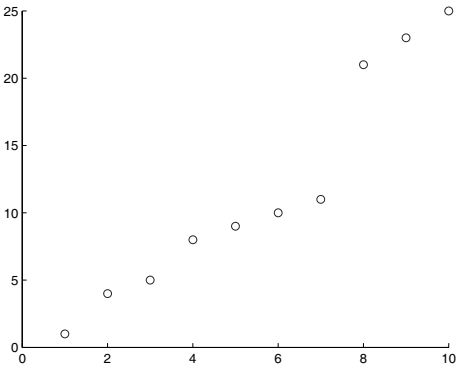


Figure 4.2: Raw scores of sorted individuals

Figure 4.2 shows the raw scores of selected sorted individuals. The raw scores are scaled with the four different scaling functions and the results of the scaled scores can be seen in figure 4.3.

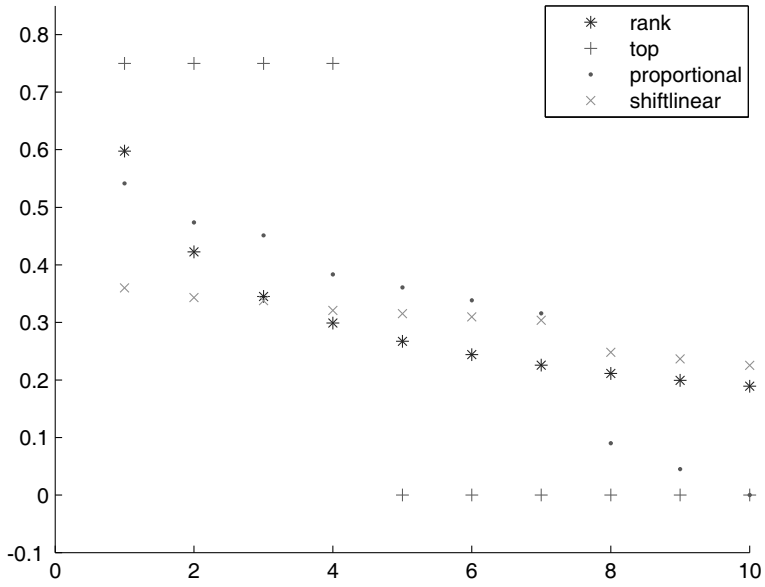


Figure 4.3: Results of scaling functions

4.5.2 Selection

Selection attempts to apply pressure upon the population in a manner similar to that of natural selection found in biological systems. Poorer performing individuals are weeded out and better performing, or fitter individuals have a greater than average chance of promoting the information they contain within the next generation. The selection function chooses parents for the next generation based on their scaled values from the fitness scaling function. An individual can be selected more than once as a parent, in which case it contributes its genes to more than one child [41]. In this work the following selection functions are used:

- **Roulette** - Roulette selection chooses parents by simulating a roulette wheel, in which the area of the section of the wheel corresponding to an individual is proportional to the individual's expectation. The algorithm uses a random number to select one of the sections with a probability equal to its area.

- **Tournament** - Tournament selection chooses each parent by choosing a certain number of players defined as tournament size at random and then choosing the best individual out of that set to be a parent. Selection pressure can be easily

adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

- **Remainder** - Remainder selection assigns parents deterministically from the integer part of each individual's scaled value and then uses roulette selection on the remaining fractional part. For example, if the scaled value of an individual is 2.3, that individual is listed twice as a parent because the integer part is 2. After parents have been assigned according to the integer parts of the scaled values, the rest of the parents are chosen stochastically. The probability that a parent is chosen in this step is proportional to the fractional part of its scaled value.

- **Stochastic uniform** - The Stochastic uniform function lays out a line in which each parent corresponds to a section of the line of length proportional to its scaled value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. The first step is a uniform random number less than the step size.

- **Uniform** - Uniform selection chooses parents randomly and therefore it is not a very effective search strategy. But it can be used for testing the algorithm.

4.5.3 Reproduction

On the one hand the reproduction depends on the fitness of the individuals selected through the selection function and on the other hand the reproduction is controlled by the operations Elitism, Mutation and Crossover. In this section those operations and the producing of the next generation are discussed.

Elitism

Elitism is a strategy to prevent from loosing the best solutions of the current population. The fittest individuals are copied to the population in the next generation. These individuals are called elite children. The number of individuals which are guaranteed to survive to the next generation is called elite count. Elitism can very rapidly increase performance of GA, because it prevents losing the best found solution to date. But setting elite count to a high value causes the fittest individuals to dominate the population, which can make the search less effective.

Creating Next Generation

At each step, the Genetic algorithm uses the current population to create the children that make up the next generation. The algorithm selects a group of individuals in the

current population, called parents, who contribute their information — the entries of their vectors — to their children. Three types of children are building the next generation:

- Elite children
- Crossover children
- Mutation children

The crossover rate specifies the fraction of each population, other than elite children, that are made up of crossover children. A crossover fraction of 1 means that all children other than elite individuals are crossover children, while a crossover fraction of 0 means that all children are mutation children.

4.5.4 Mutation

Mutation is a genetic operator that is used to alter one or more genes in an individual. Mutation provides genetic diversity and enable the Genetic algorithm to search a broader space. This operator is used to prevent GAs from stagnating at local optima. In this work the neighbourhood functions for SA and TA defined in the previous sections can be used for mutation in an analogous manner:

- Order-based mutation
- Swap-based mutation
- Position-based mutation

4.5.5 Crossover

The backbone of genetic search are the crossover operators. They combine the genotypes of two parents in the hopes to produce two even more promising children. The different solutions of scheduling problems can be directly encoded for the Genetic algorithm. Therefore special crossover operators have to be defined. In this work several crossover operators are implemented and used. The first two operators are order-based crossover (OX1) and partially matched crossover (PMX) proposed by Goldberg [36]. Next follows a second order-based crossover (OX2, GOX), a position-based crossover (PX, GPX) and an uniform crossover (UX, GUX) operator introduced by Bierwirth [42] and Mattfeld [29]. For the three latter operators generalised versions exist which can be used in case of sequences with repetition.

Order-based Crossover 1

The order-based crossover operator works by first picking two crossing sites uniformly at random. In the example below these crossing points are defined at position 4 and 7. In the next step the numbers of this matching section are exchanged to build the first part of the two children. Then the holes are filled starting with the first task next to the right hand side of the matching section. If the selected number is not yet in the child it is added and otherwise the next number is considered until all holes are filled. Child 1 gets its remaining numbers from parent 1 and, respectively, child 2 from parent 2. OX1 tends to respect the relative position of the tasks.

parent 1:	1	2	3	|4	5	6	7|	8
parent 2:	2	6	4	|5	7	1	3|	8
child 1:	*	*	*	|5	7	1	3|	*
child 2:	*	*	*	|4	5	6	7|	*
child 1:	2	4	6	|5	7	1	3|	8
child 2:	2	1	3	|4	5	6	7|	8

Partially Matched Crossover

The partially matched crossover operator starts with the same step as the order-based crossover operator. The matching section is exchanged and afterwards each parent inherits all possible tasks to each offspring. The remaining holes are filled through a positionwise exchange of the missing tasks defined by the matching section. The following exchange is used in the example below: $1 \leftrightarrow 6 \wedge 3 \leftrightarrow 7 \leftrightarrow 5 \leftrightarrow 4 \Leftrightarrow 3 \leftrightarrow 4$. In contrast to the order-based crossover the partially matched crossover operator tends to respect the absolute position of the tasks.

parent 1:	1	2	3	|4	5	6	7|	8
parent 2:	2	6	4	|5	7	1	3|	8
child 1:	*	2	*	|5	7	1	3|	8
child 2:	2	*	*	|4	5	6	7|	8
child 1:	6	2	4	|5	7	1	3|	8
child 2:	2	1	3	|4	5	6	7|	8

Order-based Crossover 2

The OX2 operator selects randomly a matching section. The length of the matching section can be arbitrary defined for each operator but it should be chosen in between one half and one third of the total length. For that both parents can inherit nearly the same amount of information to the children. In the next step all numbers of the matching section are deleted in the respective parent. The remaining tasks and the respective matching section are merged, whereas the matching section is inserted at the position of the first occurrence of the first task in the matching section. Again the OX2 operator tends to inherit the relative order of the operations.

parent 1:	~~1~~	2	~~3~~	|4	~~5~~	6	~~7~~|	8
parent 2:	2	6	4	|5	7	1	3|	8
child 1:	2	4	|5	7	1	3|	6	8

parent 1:	1	2	3	|4	5	6	7|	8
parent 2:	2	~~6~~	~~4~~	|~~5~~	~~7~~	1	3|	8
child 2:	2	|4	5	6	7|	1	3	8

In the generalised order-based crossover a so called index is defined to extend the order-based crossover in case of sequences with repetition. The index shows a counter for the occurrence of all tasks in the sequence. Now the selected tasks are deleted and merged corresponding to their indices. Using this technique it may happen that the indices of some tasks get displaced. Hence GOX introduces an implicit mutation into the children chromosome. But the choice of the crossover cut takes care of this to be rare. Because GOX arises from a generalisation of OX2, its application to ordinary permutations (all counting indices are 1) generates the same results than OX2.

parent 1:	~~2~~	|1	~~2~~	2	~~3~~	~~1~~|	3	3	2	~~1~~
index 1:	~~1~~	|1	~~2~~	3	~~1~~	~~2~~|	2	3	4	~~3~~
parent 2:	1	|2	2	1	3	1|	2	3	2	3
index 2:	1	|1	2	2	1	3|	3	2	4	3
child 1:	|2	2	1	3	1|	1	2	3	3	2

parent 1:	2	|1	2	2	3	1|	3	3	2	1
index 1:	1	|1	2	3	1	2|	2	3	4	3
parent 2:	~~1~~	|2	~~2~~	~~1~~	~~3~~	1|	~~2~~	3	2	3
index 2:	~~1~~	|1	~~2~~	~~2~~	~~1~~	3|	~~3~~	2	4	3
child 2:	|1	2	2	3	1|	2	1	3	2	3

Position Crossover

The position and the generalised position crossover work similarly to the order-based and generalised order-based crossover. The only difference is that PX and GPX insert the tasks of the matching section in the child at that position where it occurs in the parent. Therefore the absolute order is respected by neglecting the relative order of the operations.

parent 1:	~~1~~	2	~~3~~	|4	~~5~~	6	~~7~~|	8
parent 2:	2	6	4	|5	7	1	3|	8
child 1:	2	4	6	|5	7	1	3|	8

parent 1:	1	2	3	|4	5	6	7|	8
parent 2:	2	~~6~~	~~4~~	|~~5~~	~~7~~	1	3|	8
child 2:	2	1	3	|4	5	6	7|	8

Generalised version of the order-based crossover:

parent 1:	~~2~~	|1	~~2~~	2	~~3~~	~~1~~|	3	3	2	~~1~~
index 1:	~~1~~	|1	~~2~~	3	~~1~~	~~2~~|	2	3	4	~~3~~
parent 2:	1	|2	2	1	3	1|	2	3	2	3
index 2:	1	|1	2	2	1	3|	3	2	4	3
child 1:	1	|2	2	1	3	1|	2	3	3	2

parent 1:	2	|1	2	2	3	1|	3	3	2	1
index 1:	1	|1	2	3	1	2|	2	3	4	3
parent 2:	~~1~~	|2	~~2~~	~~1~~	~~3~~	1|	~~2~~	3	2	3
index 2:	~~1~~	|1	~~2~~	~~2~~	~~1~~	3|	~~3~~	2	4	3
child 2:	2	|1	2	2	3	1|	1	3	2	3

Uniform Crossover

The uniform and generalised uniform crossover purely respect the absolute order of operations. The children are initially empty. A parent is chosen at random and the operation at the first position of the parental chromosome is appended to the offspring. Then this operation is deleted from both parents. This step is repeated until both parent strings are empty and the children contain all operations involved. In the generalised version indices are used to distinguish the reoccurring operations. In the examples below the random strings define which parent is chosen. The bold tasks are taken to build the selected offspring.

parent 1:	**1**	~~2~~	**3**	**4**	**5**	~~6~~	**7**	~~8~~
parent 2:	**2**	**6**	~~4~~	~~5~~	~~7~~	~~1~~	~~3~~	**8**
random:	2	1	1	1	1	2	1	2
child 1:	2	1	3	4	5	6	7	8

parent 1:	**1**	~~2~~	**3**	~~4~~	~~5~~	~~6~~	**7**	~~8~~
parent 2:	**2**	**6**	**4**	**5**	~~7~~	~~1~~	~~3~~	**8**
random:	1	2	1	2	2	2	1	2
child 2:	1	2	3	6	4	5	7	8

Generalised version of uniform crossover:

parent 1:	~~2~~	~~1~~	~~2~~	**2**	**3**	~~1~~	**3**	**3**	~~2~~	~~1~~
index 1:	~~1~~	~~1~~	~~2~~	**3**	**1**	~~2~~	**2**	**3**	~~4~~	~~3~~
parent 2:	**1**	**2**	**2**	**1**	~~3~~	**1**	~~2~~	~~3~~	**2**	~~3~~
index 2:	**1**	**1**	**2**	**2**	~~1~~	**3**	~~3~~	~~2~~	**4**	~~3~~
random:	2	2	2	2	1	1	2	1	2	1
child 1:	1	2	2	1	2	3	1	3	2	3

parent 1:	~~2~~	~~1~~	~~2~~	**2**	~~3~~	~~1~~	~~3~~	**3**	~~2~~	~~1~~
index 1:	~~1~~	~~1~~	~~2~~	**3**	~~1~~	~~2~~	~~2~~	**3**	~~4~~	~~3~~
parent 2:	**1**	**2**	**2**	**1**	**3**	**1**	~~2~~	**3**	**2**	~~3~~
index 2:	**1**	**1**	**2**	**2**	**1**	**3**	~~3~~	**2**	**4**	~~3~~
random:	2	2	2	2	2	1	2	2	1	2
child 2:	1	2	2	1	3	2	1	3	3	2

4.5.6 Migration

In this work isolated subpopulations can be defined for the Genetic algorithm to gain better solutions. Migration specifies how individuals move between these subpopulations. When migration occurs, the best individuals from one subpopulation replace the worst individuals in another subpopulation. Individuals that migrate from one subpopulation to another are copied. They are not removed from the source subpopulation. Migration can take place in one or both directions:

- **Forward** - If the direction is set to forward, migration takes place toward the last subpopulation. That is, the $n-th$ subpopulation migrates into the $(n+1)-th$ subpopulation.
- **Both** - If the direction is set to both, the $n-th$ subpopulation migrates into both the $(n-1)-th$ and the $(n+1)-th$ subpopulation.

4.5.7 Termination Criteria

For Genetic algorithms the following termination or stop criteria can be chosen and the optimisation stops if one of these criteria is fulfilled:

- **Generations** - If a certain number of generations is reached the simulation stops.
- **Time Limit** - If the current simulated time exceeds the given time limit the simulation stops.
- **Fitness Limit** - If a certain value of the fitness function is reached the simulation stops.
- **Stall Generations - Function Tolerance** - If the weighted average change in the fitness function value over the defined number of generations is less than the defined function tolerance the simulation stops.
- **Stall Time Limit** - If there is no improvement in the best fitness value for an interval of time in seconds specified by Stall time the simulation stops.

4.6 Due-date Scheduling

In many scheduling problems, e.g. production systems, the finishing times of certain tasks play an important role concerning the optimisation of the overall cycle time. In the deterministic case each simulation run of a certain system configuration yields exactly the same result. Hence in this work it is possible to extend deterministic scheduling problems with the capability of due dates for certain and interesting tasks because the excess of the due dates is significant for each different system configuration. If the defined due dates are exceeded during the simulation a penalty time will be added to the overall cycle time at the end of the simulation. Now the objective or fitness function of the scheduling problem represents the overall cycle time plus the cumulative penalty time of the due dates. The new value of the fitness function can be used for the direct optimisation of the scheduling problems and the simultaneous indirect optimisation of the due dates.

4.7 Stochastic Optimisation

The analysis and the comparison of the output data of terminating simulation models using random numbers and stochastic distribution functions is a highly sophisticated problem. The difficulty is that the simulation output data are stochastic, so comparing two different system configurations on the basis of only a single run of each is a very unreliable approach. The results of one simulation run are not significant and in general a certain number of replications should be done to avoid making serious errors leading to fallacious conclusions and poor decisions [43]. In this section a sequential paired-t test is introduced and a variance reduction is presented.

4.7.1 Paired-t Confidence Intervall

In this work the comparison of different system configurations is reduced to a pairwise comparison. For stochastic simulation models it is difficult to get exact comparable and significant results for many different system configurations. Therefore only two different system configurations are taken into account to decide which one of the two alternatives is better. On this account the stochastic optimisation is only implemented for Simulated Annealing and Threshold Accepting. The comparison is effected by forming a confidence interval for the difference in the two expectations to see whether the observed difference is significantly different from zero. If the confidence interval misses or contains zero the test for the difference is accepted or rejected, respectively [43].
For $i = 1, 2$, let $X_{i1}, X_{i2}, \ldots, X_{in}$ be a sample of n observations from system configuration i and let $\mu_i = E(X_{ij})$ be the expected response of the interest. Now a confidence interval for $\zeta = \mu_1 - \mu_2$ is constructed. X_{1j} and X_{2j} can be paired to define $Z_j = X_{1j} - X_{2j}$, for $j = 1, 2, \ldots, n$. Then the Z_j's are random variables and $E(Z_j) = \zeta$ define the quantity for the interesting confidence interval:
Using

$$\bar{Z}(n) = \frac{\sum_{j=1}^{n} Z_j}{n}$$

and

$$\widehat{Var}[\bar{Z}(n)] = \frac{\sum_{j=1}^{n} [Z_j - \bar{Z}(n)]^2}{n(n-1)}$$

forms the approximate $100(1-\alpha)$ percent confidence interval

$$\bar{Z}(n) \pm t_{n-1,1-\frac{\alpha}{2}} \sqrt{\widehat{Var}[\bar{Z}(n)]}$$

If the Z_j's are normally distributed, this confidence interval is exact, i.e., it covers ζ with probability $1-\alpha$. Otherwise it can be relied on the central limit theorem [43], which implies that this coverage probability will be near $1-\alpha$ for large n. The defined confidence interval is called the paired-t confidence interval and in its derivation the two-system problem is essentially reduced to one involving a single sample, namely, the Z_j's. In this sense the paired-t approach is the same as the method for analysis of a single system where well-known sequential confidence-interval procedures can be applied. Another important point is that X_{1j} and X_{2j} do not have to be independent nor does it have to be assumed that $Var(X_{1j}) = Var(X_{2j})$. Allowing positive correlation between X_{1j} and X_{2j} can be of great importance, since this leads to a reduction in $Var(Z_j)$ and thus to a smaller confidence interval. Variance reduction techniques like Common Random numbers (CRN) can be used to induce this positive correlation between the observations on the different system configurations [43].

4.7.2 Sequential Estimation

Law and Kelton [43] propose a sequential procedure for estimating a confidence interval for the output of a single system with a specified relative error that takes only as many replications as are actually needed. Let $X_1, X_2, \ldots$ be a sequence of random variables of the replications that need not be normal. The specific objective of the procedure is to obtain an estimate of $\mu = E(X_i)$ with a relative error of γ $(0 < \gamma < 1)$ and a confidence level of $100(1-\alpha)$ percent. Now an initial number of replications $n_0 \geq 2$ is chosen and let

$$\delta(n, \alpha) = t_{n-1,1-\frac{\alpha}{2}} \sqrt{\frac{S^2(n)}{n}}$$

be the usual confidence-interval half-length. Then the sequential procedure is as follows:

1. Make n_0 replications of the simulation and set $n = n_0$
2. Compute $\bar{X}(n)$ and $\delta(n, \alpha)$ from $X_1, X_2, \ldots, X_n$.
3. If $\dfrac{\delta(n, \alpha)}{|\bar{X}(n)|} \leq \gamma'$ stop, where $\gamma' = \gamma(1+\gamma)$ is the adjusted relative error.

$$I(\alpha, \gamma) = [\bar{X}(n) - \delta(n, \alpha), \bar{X}(n) + \delta(n, \alpha)]$$

is an approximate $100(1-\alpha)$ percent confidence interval for μ with the desired precision. Otherwise, n is replaced by $n+1$ and an additional replication is done following step 2 of the procedure.

4.7.3 Variance Reduction

Variance reduction techniques try to reduce the variance of the sample mean. Depending on the selected problem there exist many different variance reduction techniques. In this case Common Random Numbers (CRN) are used [43, 44]. This technique is applied when two or more alternative system configurations are compared. Let X_{1j} and X_{2j} be the observations from the first and second configurations on the $j-th$ independent replication, and $\zeta = \mu_1 - \mu_2 = E(X_{1j}) - E(X_{2j})$. Let $Z_j = X_{1j} - X_{2j}$ for $j = 1, 2, \ldots, n$ replications, then $E(Z_j) = \zeta$ so

$$\bar{Z}(n) = \frac{\sum_{j=1}^{n} Z_j}{n}$$

is an unbiased estimator of ζ and

$$\widehat{Var}[\bar{Z}(n)] = \frac{Var(Z_j)}{n} = \frac{Var(X_{1j}) + Var(X_{2j}) - 2Cov(X_{1j}, X_{2j})}{n}$$

If the simulations of the two different configurations are done independently, i.e., with different random numbers, X_{1j} and X_{2j} will be independent, so that $Cov(X_{1j}, X_{2j}) = 0$. On the other hand, if the simulations of the configurations 1 and 2 can be done somehow that X_{1j} and X_{sj} are positively correlated, then $Cov(X_{1j}, X_{2j}) > 0$, so that the variance of the estimator $\bar{Z}(n)$ is reduced. CRN is a technique where this positive correlation is tried to be induced by using the same random numbers to simulate all configurations. Therefore the deterministic, reproducible nature of random number generators is used. Unfortunately, there is no completely general proof that CRN works which means that it will always reduce the variance. Even if it does work, it is not known beforehand how great a reduction in variance might be. The efficacy of CRN depends wholly on the particular models being compared. To implement CRN properly, the random numbers have to be synchronized across the different system configurations on a particular replication. Ideally, a specific random number used for a specific purpose in one configuration is used for exactly the same purpose in the other configuration [43].

Chapter 5

MATLAB PetriSimM Toolbox

5.1 Introduction

MATLAB, the classical engineering tool, does not really offer tools to handle discrete event systems based on Petri Nets. One commercial tool can be obtained by the MATLAB Connections Program [45, 46]. There are few Petri Net tools based on MATLAB which are free but these tools are rudimentary and can handle only the basics of Petri Nets. A lot of other tools exist for analysis, modelling and simulation of discrete event systems using the advantages of Petri Nets [47, 48] but all these tools cannot handle the optimisation of scheduling problems.
In this work an open source MATLAB toolbox for Petri Nets is presented and introduced. The so called MATLAB PetriSimM toolbox is based on a basic toolbox [49] which deals with analysis, supervisory control synthesis, and non-timed simulation. This basic toolbox is programmed in MATLAB version 5.3 and is therefore adapted to MATLAB version 7.2 (R2006a) to form the MATLAB PetriSimM toolbox. The toolbox is embedded in the MATLAB environment and its usage requires version 7.0 or higher. Furthermore the toolbox is extended with the capability of Timed Petri Nets and timed simulation using the holding durations principle [50, 51, 52]. In another step Coloured Petri Nets are developed for the use in the MATLAB PetriSimM toolbox. The enabling duration principle is added as a second approach of implementing time into Petri Nets. A new way of defining firing sequences is found to be able to model scheduling problems being independent of the occurrence of any conflicts [53, 54]. Finally the toolbox is extended with the optimisation of scheduling problems containing heuristic algorithms like Simulated Annealing, Threshold Accepting and Genetic Algorithms. In case of stochastic processes a sequential paired t-test and variance reduction techniques are used and implemented to solve the stochastic optimisation for sequencing and scheduling problems. To sum up, the sophisticated MATLAB PetriSimM toolbox offers the capabilities of analysis, modelling and sim-

ulation of Petri Nets. Furthermore it is possible to optimise scheduling problems based on Timed, Coloured, and Stochastic Petri Nets. The open source MATLAB PetriSimM toolbox can be used for education in a graduate level and for modelling and simulating real life processes of discrete event systems in equal measure.
In this chapter the used software MATLAB is introduced. Furthermore the basic features of the MATLAB PetriSimM toolbox are outlined and the underlying data structure is shown. Next follows an introduction to the Priority wizard of the toolbox containing the capabilities of defining priorities and sequences to Petri Nets. Another part is the visualisation of results using Gantt charts and marking plots. Afterwards the colour and transition wizards are discussed concerning the use of Coloured Petri Nets. Next follows the implementation of the sophisticated optimisation containing algorithms and the use of the optimisation wizard. Finally, selected benchmarks of the MATLAB PetriSimM toolbox are shown.

5.2 MATLAB

This section should introduce the used software environment MATLAB by answering the following two questions: "What is MATLAB?" and "Why MATLAB is used?".

5.2.1 What is MATLAB?

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

- Math and computation
- Algorithm development
- Data acquisition
- Modelling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. Therefore many technical computing problems, especially

those with matrix and vector formulations, can be solved in a fraction of the time it would take to write a program in a scalar noninteractive language such as C or Fortran. The name MATLAB stands for matrix laboratory. Today, MATLAB engines incorporate special libraries, embedding state-of-the-art software for matrix computation. MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis. MATLAB features a family of add-on application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow them to learn and apply specialised technology. Toolboxes are comprehensive collections of MATLAB functions that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others [41].

5.2.2 Why MATLAB is used?

This glorious description of MATLAB suggests that the MATLAB environment can solve nearly every problem. But with a critical point of view MATLAB is only the second best solution for the problem in many cases. As mentioned in the previous section speed and time spent for solving and modelling is an interesting factor of solving problems. On the one hand it is important to be fast in modelling and on the other hand the speed of the simulation should not left out. In this case there are many reasons for using MATLAB. First of all, the previous version of the toolbox is programmed in MATLAB version 5.3. But this is also done because Petri Nets can be described as matrices and the simulation is done by the use of matrix manipulations. MATLAB offers a lot of built-in matrix manipulation functions which are very fast if they are used in the right way. This means that the user should avoid using loops during the programming. Hence using MATALB seems to be a very good solution for modelling and simulation of Petri Nets. Further advantages are the easy development of graphical user interfaces including the use of graphical objects and the whole powerful MATLAB environment can be used for statistical post-processing and external data file handling. One disadvantage could be the speed of the simulation and optimisation depending on the size of the Petri Net and the underlying matrices. But also other programming languages would have problems with the size of the matrices and the needed memory resulting in speed problems. To sum up, MATLAB seems to be a very good solution and tool for this purpose because the MATLAB PetriSimM toolbox is embedded in the whole MATLAB environment where all features can be used and the toolbox can be easily extended with further functionalities.

5.3 Basic Features

This section should give a short overview on the basic features of the MATLAB PetriSimM toolbox which can be used for modelling, simulation and analysis of discrete event systems based on the behaviour of Petri Nets.

5.3.1 GUI

Figure 5.1 shows a screenshot of the graphical user interface (GUI) of the PetriSimM toolbox. The GUI is divided into a menu bar, a button bar and an axes area. In the menu bar different modes can be chosen. The user can switch between analysis, non-timed simulation, timed simulation based on the holding durations principle, and timed simulation based on the enabling durations principle. Furthermore models can be saved, loaded, exported and printed. For each type of simulation several parameters can be set. Another important part of the menu bar is the options menu where the priority wizard and Gantt chart wizard can be started. Next to the options menu the optimisation menu is placed. There several parameters and modes for the optimisation can be changed and selected. The button bar contains several buttons for building, changing, zooming, simulating and analysing the Petri Net models. In the axes area the Petri Net models can be created, simulated and the so called token game can be shown. Figure 5.1 also contains the Petri Net model of a production cell which is later used for the otpimisation.

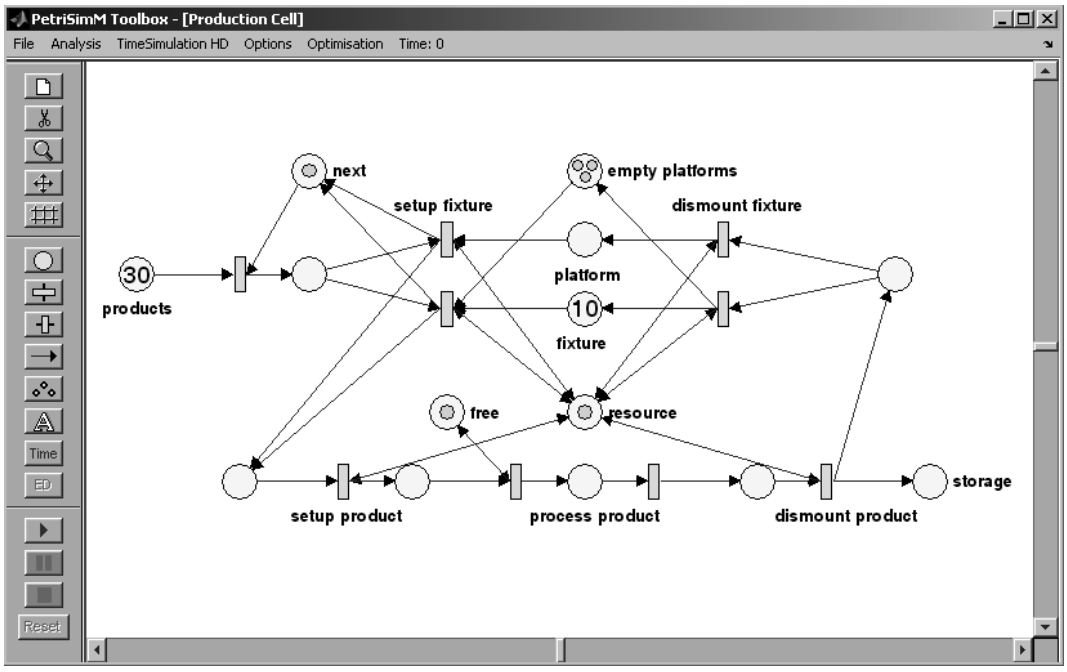


Figure 5.1: Graphical User Interface

5.3.2 Analysis

The PetriSimM toolbox offers several features for the analysis of the modelled Petri Net. In the analysis section interesting properties like P-invariants, T-invariants, reachable sets, cover-ability tree, marking bounds and dead markings can be derived. Furthermore supervisory control methods can be used. All these analysis and control features are developed in the previous version of the toolbox [49] and are not further treated in this context. In this work they are only mentioned for the sake of completeness.

5.3.3 Simulation

The simulation is a main part of the PetriSimM toolbox. It is separated into non-timed simulation, timed simulation using holding durations and timed simulation using enabling durations. For all three simulation modes an animation of the token game can be visualised. But this feature is only used for educational purposes because through the animation the simulation speed is highly increased. It is possible to change2 the animation speed in the parameter section and for the optimisation the animation is deactivated. Another interesting parameter for non-timed simulation is the firing probability. This parameter controls the firing of enabled transitions. This means that it is randomly decided if an enabled transition can fire or not depending on the defined firing probability. For timed simulation time delays can be assigned to the transitions which can be deterministic or stochastic. This means that any probability distribution function can be defined to each transition. For this purpose, any MATLAB m-file can be written resulting to a single positive value, or existing MATLAB probability distribution functions, which can be used to model stochastic time delays. Only the positive part of the used function is taken. If the result of the used stochastic function is negative the time delay is set to zero and a warning is displayed.

5.4 Data Structure

The Petri Net model is created in the axes area of the GUI by using the place, transition, token, and arc buttons. Each object is placed into the axes by clicking the left mouse button while the corresponding button is selected. The Petri Net consists of rectangle and line objects which can be accessed through their handles. The objects are only used for graphical manipulations. They can be moved, created or deleted and they are used to realise the animation of the token game. The handles of the objects are stored and added as application data to the GUI. The simulation

functions need the matrices describing the Petri Net and therefore a function called *getdata* is defined to create the input and output matrices using the objects and their connections. All used data and parameters are also stored as application data. The MATLAB commands *getappdata* and *setappdata* are used to retrieve and save data associated with the figure of the GUI. The following five data sets are defined to store all objects, parameters and results:

- **menus** - In this data set all handles of the menu bar are stored. It is used for changing the menus and their submenus depending on the selected state and mode of the toolbox.
- **buttons** - The buttons data set contains all handles of the buttons to control their visualisation.
- **simulation** - In the simulation data set all relevant parameters for the simulation are stored. Furthermore all simulation results can be accessed.
- **optimisation** - The optimisation data set contains all parameters and results for the optimisation.
- **petrinet** - In this data set all used object handles of the Petri Net are stored.

5.5 Colour Wizard

The PetriSimM toolbox is supposed to be user friendly and therefore several wizards are used and implemented to assign the different parameters and to modify the different settings and properties. Figure 5.2 shows a screenshot of the Colour Wizard. The

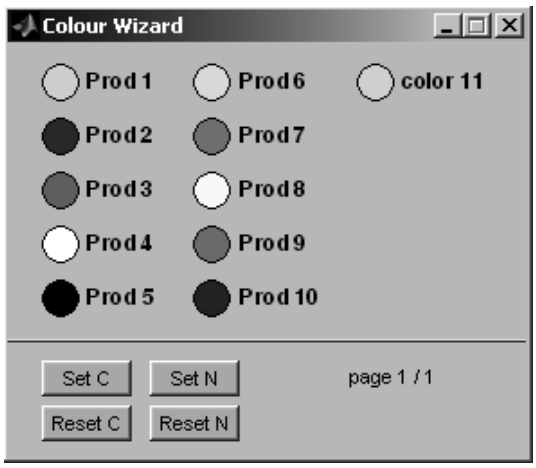


Figure 5.2: Colour Wizard

different colours represent the different types of tokens. In the parameter section the number of desired distinguishable tokens can be defined. If more than one tokens are chosen the so called Colour Wizard can be shown. The Colour Wizard can be used to change the look and the names of the different token colours. The colours are coded as RGB values and can be easily modified. It is also possible to reset all settings to the default values and names. The colours of the used different tokens are interesting for the animation of the simulation. The so called token game is mainly used for educational purposes. The colours of the tokens are also used for the visualisation of the results of the simulation shown in one of the next sections.

5.6 Transition Wizard

The Transition Wizard is used to modify and define the transitions in case of Coloured Petri Nets. Figure 5.3 shows the Transition Wizard containing 10 transitions. The graphical description of Petri Nets can be simplified if many identical sub nets are used. Therefore the transitions of the sub nets are merged together and one transition of the Petri Net can represent and contain many other transitions. The Transition Wizard consists of several buttons corresponding to the following functionalities: transitions can be deleted, new transitions can be added, the names of the transitions can be modified and the weights of the corresponding arcs can be defined.

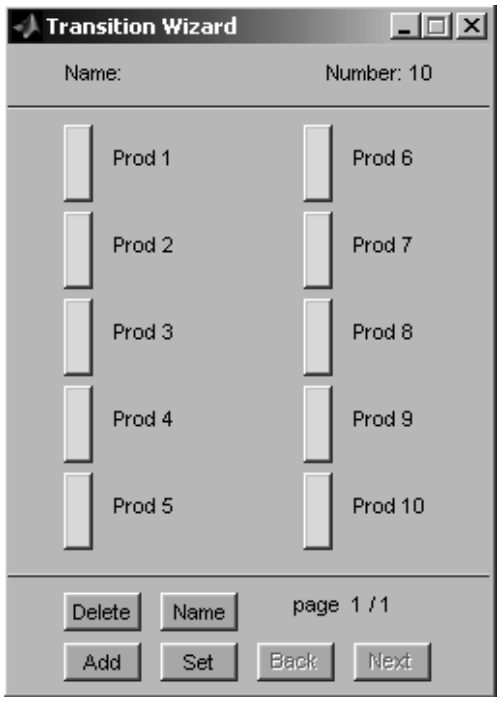


Figure 5.3: Transition Wizard

5.7 Priority Wizard

Another important wizard is the so called Priority Wizard. Figure 5.4 shows a screenshot of the Priority Wizard. This wizard is used to realise and assign the priorities and sequences to the transitions. It is possible to switch between priorities and sequences and the wizard can therefore be used for both modes. In case of priorities transitions can be selected and added to disjoint groups and afterwards priority vectors can be assigned.

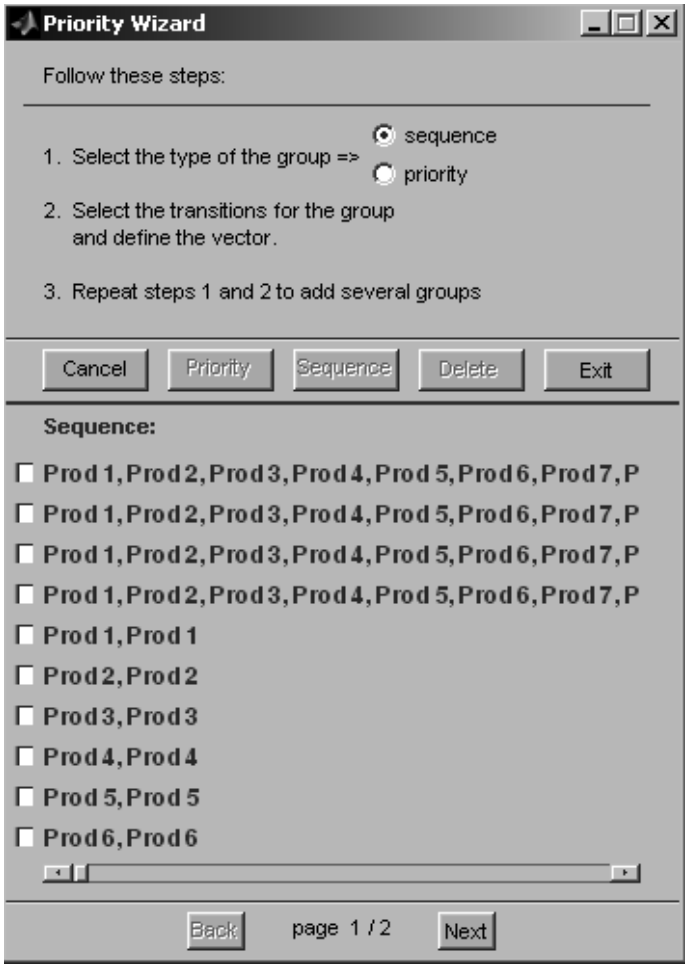


Figure 5.4: Priority Wizard

Figure 5.4 shows the Priority Wizard displaying the sequence mode. In this example several sequence groups can be seen. Each group of transitions can be selected and by the use of the sequence button a sequence or firing list can be assigned. In special cases and models it may happen that the same firing list should be used for other groups similar in size. Sequence lists can then be defined as so called source lists. This means that for each group there are two possibilities to assign a sequence list.

On the one hand a sequence list can be directly set and on the other hand another existing sequence list can be taken corresponding to another group of transitions similar in size. Further, a sequence priority vector can be defined to the group to offer the correct and controlled solving of conflicts between two or more groups of sequence transitions.
Depending on the modelled scheduling problem the assigned sequence list can directly represent the firing list for the respective transitions. It may also happen that the defined sequence contains the information of scheduling but a special function is needed to decode the sequence to the right firing list. Another problem specification is described by the following case. The selected scheduling problem consists of many different sequences but one sequence is affected by another sequence. This means that a change of the first sequence leads to a simultaneous change of the second sequence. The PetriSimM toolbox offers a solution of such problems where sequences depend on other sequences or conditions. In the parameter section it is possible to register a so called sequence function handle and several function arguments. This function handle has to correspond to an existing m-file representing the user defined sequence function. The predefined output and needed input of the sequence function are given by the simulation part of the stored data structure. Therefore this functionality should only used by advanced users. The modelling of scheduling problems based on Petri Nets is limited by the size and the properties of Petri Nets. Using Coloured Petri Nets enables the modelling of complex problems. Scheduling problems mainly depend on the used sequences. By the use of arbitrary sequence functions the firing of the transitions does not directly depend on the given input sequences. More information is put into the Petri Net model and therefore the complexity of the scheduling problem being modelled can be extended again.

5.8 Results

Results are an important and interesting part for each simulation study. Scheduling problems focus on the allocation and the utilisation of resources over time. Using Petri Nets offer a very simple approach for getting the measurement of the interesting parameters. In this case only the number of tokens in the places are stored over time during the simulation. Afterwards it is possible to access and to show selected results for each place of the Petri Net. Figure 5.5 shows the Gantt Chart Wizard. This wizard can be used to select arbitrary places and to show a Gantt chart or a marking plot considering several parameters and properties. The Gantt Chart wizard can have different buttons and functionalities depending on the selected simulation mode and the number of used colours. In this case figure 5.5 shows the Gantt Chart

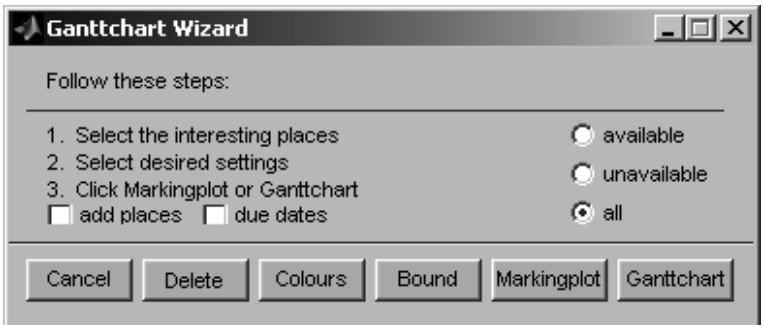


Figure 5.5: Gantt Chart Wizard

Wizard for the holding durations principle. The results can be separated to the information for available, unavailable or all tokens. Furthermore, it is possible to show the information combined in one graphic for all selected places using the "add places" check box. The information concerning due dates can be visualised in the Gantt chart by activating the "due dates" check box. For Coloured Petri Nets the desired colours represented by the different token types can be defined. The number of tokens in each place can be bounded and defined by the use of the bound button. The results in the graphics are then inverted. This means that the bound values minus the numbers of tokens are shown. In this section the different and used types of graphics are shown and explained. First the Gantt chart is introduced and afterwards the marking plot is shown. Next follows the implementation of the data handling because storing of all information for a simulation can be a highly sophisticated problem.

5.8.1 Gantt Chart

A Gantt chart is the graphical representation of the duration of tasks against the progression of time. Figure 5.6 shows an example of a Gantt chart.

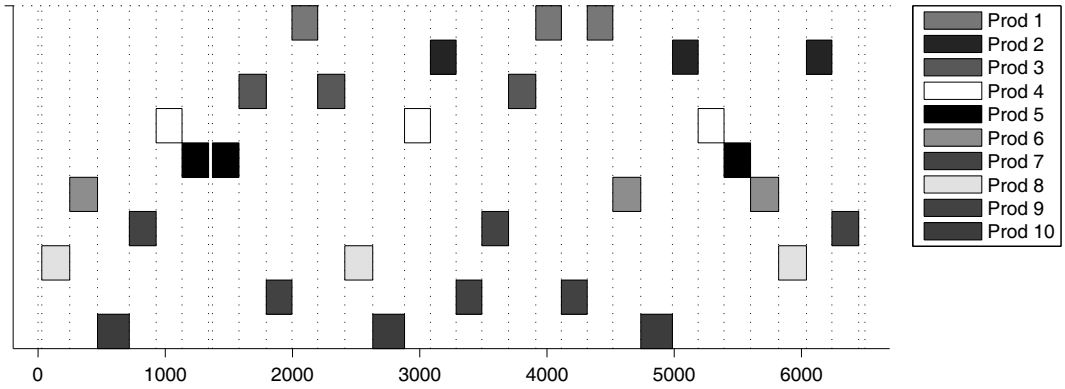


Figure 5.6: Gantt Chart

5.8.2 Marking Plot

Among the Gantt chart functionality it is possible to show a so called marking plot. Any places can be selected and after a timed simulation run the number of tokens over the places can be shown depending on the simulation mode. Figure 5.7 displays the number of all tokens inside one place over simulated time for five different colours.

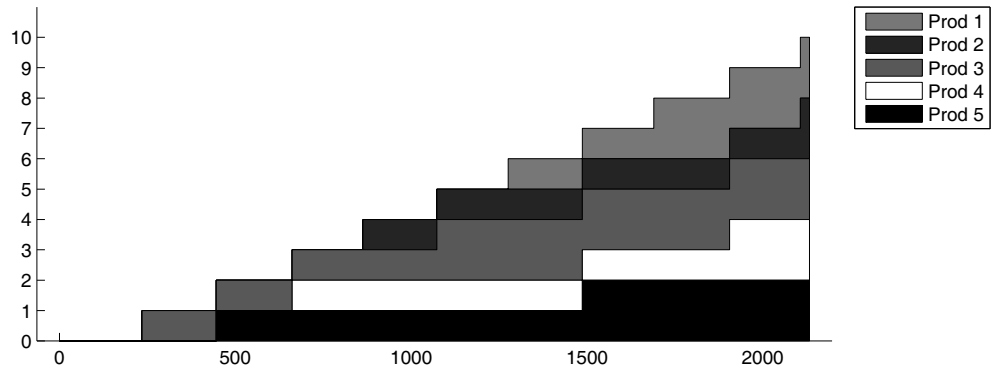


Figure 5.7: Marking Plot

5.8.3 Data Handling

The allocation of the places over time is needed to show and to analyse results of the simulation. This means that the number of tokens has to be stored for each place and for each colour in each iteration step. The number of iterations depends on the specific problem. Therefore this value is not fixed and in general it can be very large. On this account very big matrices are created during the simulation. Dynamic memory allocation could be used to solve this problem. But the access to big matrices causes speed problems corresponding to the size of the used matrices. If the matrices and the stored information are increased too much memory problems can also occur. In the PetriSimM toolbox these problems are solved by the use of external binary files. The built-in MATLAB command *fwrite* is used to store the data during the simulation. At the end of the simulation the *fread* command is used to store the matrices in MATLAB as application data of the GUI. If the memory of the program is exceeded the binary files are transformed to text files in order that they can be used for post-processing using other tools or programming languages. The big advantage of this implementation is the fact that the data storing takes the same time in each iteration of the simulation algorithm. The used built-in functions are very fast and therefore only a marginal increase of computation time can be determined for the simulation function.

5.9 Optimisation

5.9.1 Introduction

The optimisation is one of the main parts of the PetriSimM toolbox. All described heuristic algorithms are implemented, whereas Simulated Annealing and Threshold Accepting are not included in the version 7.2 of MATLAB (R2006a). Therefore these two algorithms are newly developed and added to the MATLAB PetriSimM toolbox. The MATLAB environment contains the so called Genetic Algorithm and Direct Search Toolbox. This toolbox already provides the optimisation based on Genetic algorithms but the special case of scheduling problems is not really covered. The direct coding of scheduling problems is not implemented and no crossover and mutation functions are defined. The existing MATLAB toolbox offers the possibility to create user-defined functions for special purposes. This means that a user can program and add functionalities if it is necessary for selected problems. Hence, the existing Genetic algorithm toolbox could be used and extended to realise the optimisation of scheduling problems. For this purpose it seems more practical to use selected functions of the existing Genetic algorithm and to adapt them to the MATALB PetriSimM toolbox. This approach prevents the system from the overhead caused by the general programming of the original MATLAB toolbox. Additionally needed functionalities are directly implemented to the PetriSimM toolbox. On the one hand fitness scaling, selection, and migration are realised by existing functions and on the other hand population coding, crossover, and mutation are newly implemented. Furthermore, newly developed features are the simultaneous optimisation of more than one input sequences and the stochastic optimisation using a paired t-test and variance reduction techniques.
In this section the optimisation wizard is shown and the used options and features are described. Next follows the different parameters and functions of the optimisation algorithms. Furthermore, the possible results of the optimisation are discussed. An important part of the deterministic optimisation is due-date scheduling. It is therefore shown how due dates are defined and assigned to Petri Nets. Last but not least the implementation of the variance reduction techniques are described.

5.9.2 Optimisation Wizard

The optimisation wizard contains the basic control of the optimisation. Figure 5.8 shows a screenshot of the optimisation wizard. First of all the optimisation mode can be selected whereas the Genetic algorithm is only available in the deterministic mode. Next follows the selection of the optimisation algorithm. The sequence button of the optimisation wizard plays an important role because the underlying functionality is

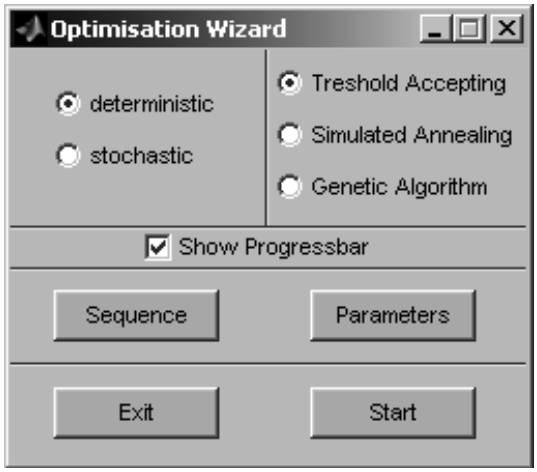


Figure 5.8: Optimisation Wizard

used to define and to select the different transition groups including the sequences for the optimisation. In the parameters section all settings for the selected optimisation algorithm can be defined. But these functionalities are treated in the next section. During the optimisation it is possible to show a progressbar to watch the progress of the optimisation and to hold up the interaction to the optimisation function for the possibility of user-defined program aborts. The progressbar can also be deactivated to increase the computational speed of the optimisation function.

5.9.3 Parameters

The results of the optimisation can be influenced by the choice of the used parameters. Depending on the different modes and algorithms all used parameters are listed below.

Genetic Algorithm:

- **Population** - The population parameters are the population size, the creation function and a possible initial population. The default setting for the creation function is an uniform function but also a user-defined function can be used to create the initial population. If a matrix of correct values is defined as initial population no creation function is used.

- **Fitness scaling** - For fitness scaling the built-in MATLAB functions *Rank, Proportional, Top* and *Shift linear* can be selected. Furthermore, user-defined functions can be implemented and defined as function handles.

- **Selection** - Selection can be done by the built-in MATLAB functions *Stochastic uniform, Remainder, Uniform, Roulette, Tournament* and by user-defined functions.

- **Reproduction** - The reproduction parameters control the elite count and the crossover fraction.
- **Mutation** - Mutation is realised by order-based mutation (OBM), swap-based mutation (SBM), position-based mutation (PBM), and by their generalised versions (GOBM, GSBM, GPBM). Custom functions can be developed with arbitrary function arguments to define new mutation functions.
- **Crossover** - On the one hand order crossover (OX1) and partially matched crossover (PMX) can be used and on the other hand the other six crossover functions described in this work can be selected for crossover: order crossover (OX2), position crossover (PX), uniform crossover (UX), and their generalised versions (GOX, GPX, GUX).
- **Sequence - repetition** - In case of sequences with repetition the initial sequences can be transformed to sequences without repetition whereas equal tasks are treated as unequal ones.
- **Migration** - If the population is defined as vector, migration can be defined between subpopulations. The migration parameters are given by direction, fraction of migrating individuals, and the interval defining the occurrence of migration.
- **Stop criteria** - The stop criteria are defined by generations, time limit, fitness limit, stall generations, stall time limit, and function tolerance.
- **Optimisation** - The optimisation parameters include the choice between minimisation and maximisation. Furthermore, due-date scheduling can be activated and deactivated.

Simulated Annealing - Threshold Accepting:

- **Neighbourhood functions** - All mutation functions of the Genetic algorithm can be selected as neighbourhood functions.
- **Cooling - Lowering** - Cooling and lowering, respectively, is done by the use of the following three functions: exponential, linear and logarithmic. User-defined functions with arbitrary function arguments can be added.
- **Stop criteria** - The stop criteria are defined by generations, time limit, and fitness limit for both algorithms.
- **Statistical parameters** - In case of stochastic optimisation the statistic parameters are given by initial runs, confidence level, and relative error.

- **Optimisation** - Next to the choice between minimisation and maximisation the variance reduction can be activated and deactivated.

5.9.4 Results

The results of the optimisation are automatically shown in a special results window after the termination of the optimisation algorithm. The results can also be accessed by clicking into the optimisation menu of the menu bar. Figure 5.9 shows a screenshot of the optimisation results window. All used algorithms can be selected if optimisation results are existing. Otherwise the radio buttons corresponding to the algorithms are disabled. The results for each optimisation mode can be shown by selecting one of the enabled radio buttons. The used computational time for the optimisation can then be seen. Other interesting information is the number of computed generations and the termination reason of the optimsiation algorithm. Furthermore, the overall cycle time for the initial and best solution are displayed. The optimised and initial sequences can be accessed by using the sequence button of the results window. A modified priority wizard is opened to select each optimised transition group. Each transition group can then be chosen and the initial and best solution for the selected sequence are shown.

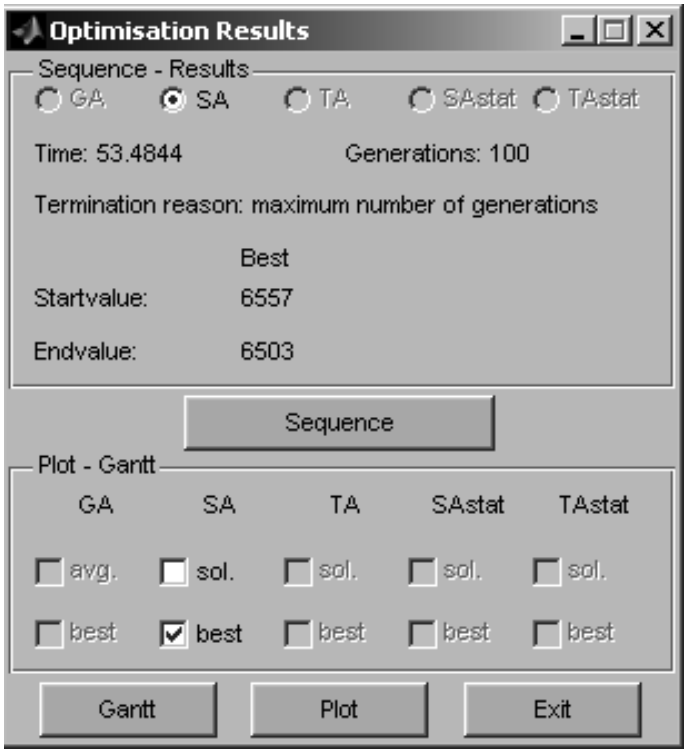


Figure 5.9: Results window

The results window additionally offers a plot function. Plots of existent data can be shown and compared for each optimisation algorithm. The plot button is used to open the plot window after selecting the corresponding check boxes. Figure 5.10 shows the plot window including results of an optimisation using Threshold Accepting. In this example 100 iterations are done. Two graphs are shown where the green line represents the best solution and the blue line represents the value of the objective function for each iteration.

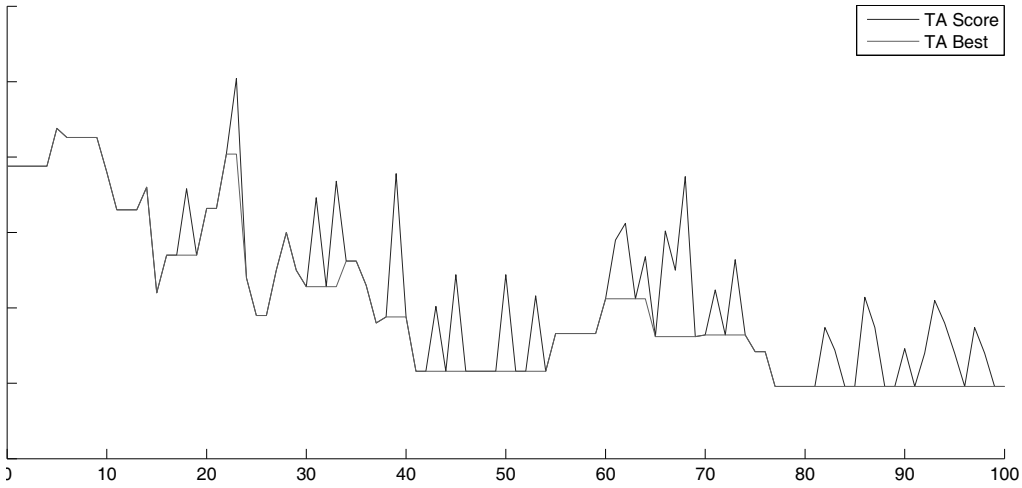


Figure 5.10: Plot of Optimisation Results

Gantt charts can be used to compare different system configurations. In particular, they can visualise the difference between the initial and best solution derived from the optimisation of scheduling problems. Each optimisation mode can therefore be selected in the results window to realise this functionality. The gantt button is used to simulate the solution corresponding to the selected check box. In principle, the simulation function works without storing allocation data to gain speed during the optimisation. In this case the allocation data storing is activated and the allocation of the places over time is added as application data to the figure of the PetriSimM toolbox. Consequently, a Gantt chart of the simulated solution can be shown using the functionalities of the Gantt Chart Wizard.

5.9.5 Due Dates

Due-date scheduling is an important tool for the deterministic optimisation of scheduling problems. The finishing times of certain tasks can be essential for the optimisation. The PetriSimM toolbox offers the possibility to define and assign due dates to each place of the Petri Net. Figure 5.11 shows a screeshot of the due

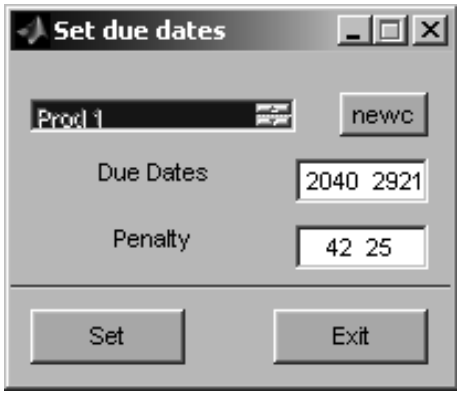


Figure 5.11: Due dates input box

dates input box. Desired finishing times and penalties can be defined for each colour and for each entity of the system. If the defined due dates are exceeded during the simulation, the defined penalty times are multiplied with the difference of the real and desired finishing times. There are two approaches to implement the calculation of the overall penalty time. On the one hand the information about the allocation of all places over time can be stored during the simulation and on the other hand the needed values can be computed during the simulation without storing unessential data. The latter approach is realised by the following MATLAB code called in each iteration of the simulation function:

```
for j=1:length(optdata.duedatesidx)
    if m(optdata.duedatesidx(j)) > start(j) && countvec(j) > 0
       diff=m(optdata.duedatesidx(j)) - start(j);
       for k=1:min(diff,maxn(j))
          start(j)=start(j)+1;
          countvec(j)=countvec(j)-1;
          casvectmpj(mm(j))=t;
          mm(j)=mm(j)+1;
       end
    elseif m(optdata.duedatesidx(j)) < start(j) && countvec(j) > 0
       start(j)=start(j)-max(1,m(optdata.duedatesidx(j)));
    end
end
```

This variant should yield a decrease of computational speed because not all information is needed and stored during the simulation. At the end of the simulation the penalty times can be calculated using the computed finishing times which are stored in the variable *casvectmp*. The following MATLAB code is used to get the overall penalty time of the simulation run:

```
penalty=0;
for j=1:length(optdata.duedatesidx)
    tmp=casvectmpj-optdata.duedatesj;
    pen=optdata.duedatespenaltyj;
    idx=find(tmp>0);
    tmpval=tmp(idx)*pen(idx)';
    if isempty(tmpval)
       tmpval=0;
    end
    penalty=penalty+tmpval;
end
t=t+penalty;
```

Both approaches are implemented in the MATLAB PetriSimM toolbox to compare the used computational speed. Table 5.1 shows the results of the comparison for the simulation of a Petri Net consisting of 14 places, 9 transitions and 31 colours corresponding to a 434 $\times$ 270 matrix . The size and unit of the values and the power of the used computer system are not relevant because the percentage of the interesting difference should be significant for the comparison. The due dates column represents the cumulative number of due dates which are assigned to the places of the Petri Net using all colours and tokens.

Due dates	Approach 1	Approach 2	Difference
92 (1 place)	7.84	7.46	-5.09 %
276 (3 places)	7.85	7.56	-3.84 %
460 (5 places)	7.88	7.74	-1.81 %
644 (7 places)	7.93	7.95	+0.25 %
828 (9 places)	7.99	8.13	+1.73 %
1104 (12 places)	8.05	8.41	+4.28 %
1288 (14 places)	8.11	8.65	+6.24 %

Table 5.1: Comparison of Due Dates Calculation

The use of the different approaches depends on the number of assigned due dates. If this number exceeds the value corresponding to half the places using all colours and tokens, approach one should be selected instead of approach two and vice versa, respectively. The difference is calculated for one simulation run and it therefore plays an essential role for the duration of the optimisation where usually a large number of simulation runs are needed.

5.9.6 Variance Reduction

In this work Common Random Numbers (CRNs) are implemented because different system configurations are compared. As mentioned in the last chapter, the synchronisation of the use of random numbers is one of the most important conditions. MATLAB provides randomness and several stochastic distribution functions based on the two basic random number functions *rand* and *randn*. *Rand* returns a pseudo-random, scalar value drawn from a uniform distribution on the unit interval and *randn* returns a pseudo-random, scalar value drawn from a normal distribution with mean 0 and standard deviation 1.
In the PetriSimM toolbox the variance reduction can be chosen as an option of the stochastic optimisation using Simulated Annealing and Threshold Accepting. The synchronisation of the random numbers is done in the following way. Each stochastic distribution function assigned to a transition has its own random number stream. First of all, different initial states of the used random number generators are defined for all transitions. The following MATLAB code is used to set the initial seeds for the transitions:

```
current=rand('state');
currentn=randn('state');
sim.statistic.rand=random(state.StatisticCount+options.seed,optdata.dT);
sim.statistic.randn=randomn(state.StatisticCount+options.seed,optdata.dT);
rand('state',current);
randn('state',currentn);
```

The current states are temporarily stored to reset them after assigning the initial states to the data structure of the PetriSimM toolbox. The input seed is randomly changed for each different iteration using the following code at the beginning of the simulation run:

```
options.seed=unidrnd(100000);
```

The following two functions implement the assignment of the initial states to each transition for the *rand* and *randn* function:

```
function rnd=random(count,dT)
rnd=zeros(35,dT);
for i=1:dT
    rand('state',dT*count+i);
    rnd(:,i)=rand('state');
end
```

```
function rndn=randomn(count,dT)
rndn=zeros(2,dT);
for i=1:dT
   randn('state',dT*count+i);
   rndn(:,i)=randn('state');
end
```

If the variance reduction technique is activated for the stochastic optimisation the following code is performed to evaluate the stochastic distribution functions assigned to the transitions. The states for the needed random number functions are redefined corresponding to the involved transition. After evaluating the distribution function the new states are stored again for the use in the next evaluation.

```
current=rand('state');
currentn=randn('state');
rand('state',sim.statistic.rand(:,fire(j)));
randn('state',sim.statistic.randn(:,fire(j)));
ti=eval(TimeTfire(j));
sim.statistic.rand(:,fire(j))=rand('state');
sim.statistic.randn(:,fire(j))=randn('state');
rand('state',current);
randn('state',currentn);
```

Table 5.2 shows the results of the additional computational effort using the implemented variance reduction technique. One simulation run is performed for a selected Petri Net model whereas different numbers of transitions can be defined. Again the percentage of the difference is significant and independent of the used computer architecture.

Transitions	Normal	VR	Difference	%
18	0.057	0.063	0.006	+9.52%
45	0.256	0.267	0.009	+4.12%
90	0.782	0.793	0.009	+1.39%
135	1.725	1.735	0.010	+0.58%
180	2.910	2.920	0.010	+0.34%
225	7.895	7.916	0.021	+0.27%
270	11.364	11.374	0.010	+0.09%
315	18.323	18.337	0.014	+0.08%

Table 5.2: Results with and without Variance Reduction

The increase of the additional effort for the variance reduction is marginal. The percentage of the difference is decreasing relatively to the increasing of the used transitions. The speed difference results from only one simulation run but in the next chapter it can be shown that the additional effort is accounted for because for the sequential paired t-test the number of needed runs can be decreased and therefore the overall computational time for the stochastic optimisation can be decreased.

5.10 Benchmarks

5.10.1 Introduction

Petri Nets offer the possibility to model on a very low level. The use of Coloured Petri Nets provides a way of defining compact graphical descriptions for great models. But problems of great complexity result in very big underlying matrices. Therefore Petri Nets are bounded and limited to the size of their corresponding matrices because the algorithmic manipulation realising the simulation implies great computational effort. The number of enabled transitions has to be checked and computed in nearly every iteration of the simulation function. Furthermore, the resolution and processing of conflicts are time and memory consuming. The use of more powerful computer systems would extend the limitations and constraints of modelling and simulation but the increase of complexity accompanies the increase of the needed computational power.
In this section several benchmarks are done to test and compare the used algorithms and functions concerning the size of the used matrices. Furthermore, the different implemented simulation modes and time approaches are considered. As mentioned in the previous section absolute values and results are not significant. In this case the relative difference of the selected comparisons are used because these results should be more or less independent of the power of the used computer system.

5.10.2 Deterministic - Stochastic

In the PetriSimM toolbox deterministic and stochastic time delays can be assigned to the transitions. To realise the stochastic simulation the use of stochastic distribution functions is implemented. In the process, it is tried to ensure a fast implementation concerning the simulation and a simultaneous flexibility to assign arbitrary functions based on built-in functions or user-defined m-files. Table 5.3 shows the results of the comparison. In this case computational times of simulation runs are compared whereas deterministic and stochastic time delays are used. The size of the Petri Net model is iteratively increased. It is shown that the additional computational effort for the stochastic simulation amounts to about 40 percent of the deterministic one. The increase of the size of the used matrices do not really influence the additional effort. The amount of the additional effort depends on the number of transitions using stochastic time delays. In this example nearly all time delays change from deterministic to stochastic.

Size of Petri Net	Deterministic	Stochastic	Difference	%
84 x 45	0.163	0.275	0.112	+40.73%
154 x 90	0.453	0.788	0.325	+41.77%
224 x 135	0.861	1.519	0.658	+43.32%
194 x 180	1.451	2.598	1.147	+44.15%
364 x 225	4.446	7.588	3.142	+41.41%
434 x 270	6.213	11.008	4.795	+43.56%
504 x 315	10.448	18.563	8.115	+43.72%
574 x 360	15.447	27.561	12.114	+43.95%

Table 5.3: Difference Deterministic and Stochastic Simulation

5.10.3 Holding - Enabling

In this work two different approaches of time implemented to Petri Nets are used. The holding duration and enabling duration principles differ in their implementation and modelling power. This benchmark should compare the use of the two principles concerning the computational effort of the simulation for different dimensions and different numbers of conflicts. The information about the number of enabled transitions is needed in every iteration of the simulation function. For the holding duration principle two different checks are necessary. On the one hand the number of enabled transitions for available tokens is needed and on the other hand this information for all tokens is additionally required in some cases to decide whether a deadlock is occurred to detect the desired end of the simulation. This computation can be easily done using the following MATLAB commands and functions:

```
idx=idxvec(all(M(:,ones(dT,1))>=I))
```

In contrast to holding durations the enabling duration principle defines one type of token and therefore only one check has to be done in every iteration of the simulation function to compute the number of enabled transitions. Transitions using enabling durations offer the capabilities of multiple enabling and preempting. Therefore additional information is needed in every iteration. To handle the multiple enabling the degree of enabling is calculated by the following command:

```
degreetmp=floor(min(iv(iv>0).\m(iv>0)))
```

Conflicts also cause an additional effort using enabling durations. Each involved conflict has to be checked after firing to realise the possible preempting and to handle the defined memory management of the transitions.

Both approaches have advantages and disadvantages depending on the characteristic of the present problem. Therefore two different scenarios are simulated and modelled using holding and enabling durations principles to compare the needed computational speed of the simulations. The first problem which is modelled and compared includes many conflicts whereas the second scenario is conflict-free.

Size of Petri Net	Conflicts	Holding	Enabling	Difference	%
84 x 45	14	0.152	0.315	0.163	+107.23%
154 x 90	24	0.553	1.024	0.471	+85.17%
224 x 135	34	1.354	2.289	0.935	+69.05%
294 x 180	44	2.546	4.287	1.741	+68.38%
434 x 270	64	6.470	9.358	2.888	+44.64%
574 x 360	84	15.521	21.805	6.284	+40.49%
714 x 450	104	27.979	36.096	8.117	+29.01%
1064 x 675	154	85.702	97.249	11.547	+13.47%
1414 x 900	204	204.635	229.234	24.599	+12.02%
1484 x 945	214	241.003	268.560	27.557	+11.43%

Table 5.4: Difference Holding and Enabling I

Table 5.4 shows the results of the first scenario. The holding durations principle always leads to faster computational times than the enabling durations principle. The percentage of the needed additional time decreases depending on the increase of the number of conflicts and the size of the Petri Net. These results arise from the fact that the additional processing of the conflicts is the decisive factor in case of small matrices. By contrast, the determination of the number of enabled transitions implies a greater computational effort for large-sized matrices. On this account the enabling durations principle regains lost time because in this case less checks have to be done to compute the number of enabled transitions.
Table 5.5 presents the difference of the holding and enabling durations principles concerning the second scenario. Maybe this conflict-free situation is not a typical problem modelled by Petri Nets but such problems can appear if scheduling problems are considered. This scenario shows that the enabling durations principle is faster than holding durations if few or no conflicts have to be treated.
The results of the two scenarios show that the use of the implemented time approach depends on the characteristic of the present problem. Basically, the enabling durations principle is more powerful and therefore only transitions including infinite-server semantics are used to realise the conducted comparisons. If many conflicts have to

be resolved the holding durations principle should be chosen. Otherwise enabling durations should be used in case of few conflicts. Therefore the choice of the time definition affects the computational duration of the simulation. This is an important factor for the duration of the optimisation of scheduling problems because basically a lot of simulation runs are needed to find a feasible and good solution of the optimsiation problem.

Size of Petri Net	Holding	Enabling	Difference	%
3 x 190	0.020	0.019	0.001	-5.26%
3 x 435	0.037	0.035	0.002	-5.71%
3 x 780	0.065	0.062	0.003	-4.84%
3 x 1225	0.107	0.093	0.014	-15.05%
3 x 1770	0.175	0.141	0.034	-24.11%
3 x 3160	0.342	0.297	0.045	-15.15%
3 x 4950	0.713	0.594	0.119	-20.03%
3 x 11175	2.383	1.834	0.549	-24.48%
3 x 19900	5.536	4.464	1.072	-24.01%
3 x 31125	10.260	7.857	2.403	-30.58%
3 x 44850	18.429	15.098	3.331	-22.07%

Table 5.5: Difference Holding and Enabling II

The results of the benchmarks show the high increase of the needed computational effort corresponding to the size of the Petri Net and its underlying matrices.

Chapter 6

Case Studies

6.1 Introduction

In this chapter two case studies are processed to test and compare the implemented optimisation algorithms. Both problems described here are already used for the benchmarks in the previous chapter. The first case study contains the modelling, simulation, and optimsiation of a special type of production cell [55, 56]. The Petri Net model of this problem consists of many conflicts and therefore the holding durations principle is used to model the production cell. All used time delays are deterministic and due dates and arrival times are therefore assigned for selected products. This is a second reason for choosing holding durations because the arrival times can be easily realised by the use of initial unavailable tokens.
The modelling, simulation, and optimisation of the well-known travelling salesman problem [57] is done in the second case study. Maybe Petri Nets are not really the best solution for modelling this problem but it is very useful to show the capabilities of the MATLAB PetriSimM toolbox. The conflicts of the underlying Petri Net are disabled by the definition of the sequence list and therefore no real conflicts have to be considered. For this reason the enabling durations are used to model this problem to reduce the computational duration of the optimisation. Furthermore, a stochastic version of the travelling salesman problem is implemented. In this case the time delays of the transitions are modelled by stochastic distribution functions. Variance reduction is used to speed up the stochastic optimisation and selected benchmarks are done to show the advantage of the implemented technique.
This chapter is divided in two parts. Both scenarios are treated in own sections. First of all, the problem formulation is described for the selected problem. Second of all, follows the implementation and last but not least selected results of the case studies are shown.

6.2 Production Cell

6.2.1 Problem Formulation

In this production cell different types of products can be processed. Each product type requires a special fixture and for each type exist only one fixture. This fixture is used to fix the product into the suitable position for processing. Platforms are used to move and shift the products inside the production cell.

The process flow is separated into five different production steps (figure 6.1):
Setup of fixtures: If the fixture for the next product is not mounted onto a platform it has to be mounted onto an empty one. This procedure takes the setup-time for the fixture. Otherwise the platform with the fixture for the next product is taken without loosing time.
Setup of products: Products are mounted onto the fixtures. For this step the setup-time for the product is needed.
Processing: In this step the product is processed automatically and computer controlled. This procedure takes the processing time for the product and the processing is characterised by long process times.
Dismounting of products: After the processing step the products are dismounted from the fixture. For this process part the dismounting time for the product is needed.
Dismounting of fixtures: Fixtures are dismounted depending on the sequence order of the products. If the fixture is dismounted it takes the dismounting time for the fixture. Otherwise the platform with the fixture on it is waiting for setup the product.

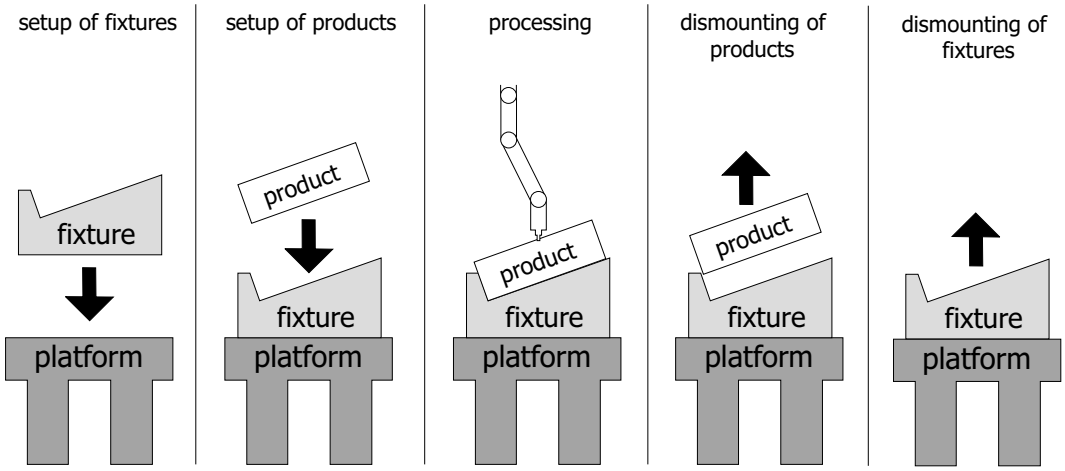


Figure 6.1: Process Flow of Production Cell

The processing of the products is automated and is independent of any restrictions. By contrast, one resource is shared by the four other process steps of the process flow. This means that the process steps have to be operated one after another. Therefore the following prioritisation is used and implemented to optimise the process flow:

1. dismounting of fixtures
2. dismounting of products
3. setup of products
4. setup of fixtures

Due Dates

In case of due dates selected products have given arrival and finishing times. After the products have passed the processing part of the production cell they are finished. If the desired finishing times are exceeded a penalty time will be added to the overall cycle time of the production.

6.2.2 Implementation

The Petri Net model of the production cell is automatically generated by the use of a programmed template. The function *productioncell()* creates all needed places, transitions, tokens and colours whereas the following input parameters can be used: names of the products, the number of platforms, the initial sequence order of the products, and the time delay matrix.

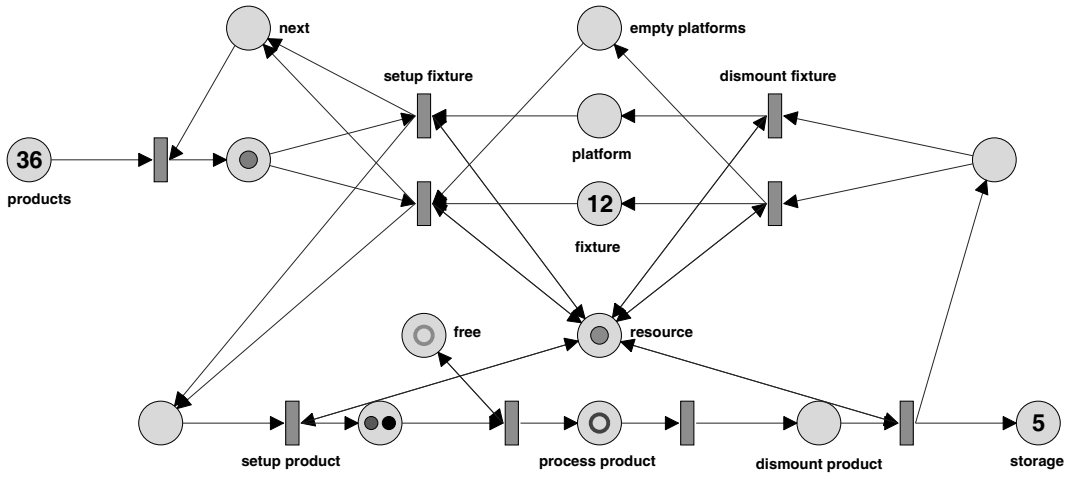


Figure 6.2: Model of Production Cell

The model (figure 6.2) is separated into the five parts of the process flow. The sharing of the resource is implemented as a conflict of the place "resource" and its connected transitions (setup fixture, dismount fixture, setup product and dismount product). The different colours represent the different product types and fixtures. In this case an additional colour is used to model the empty platforms, the availability of the resource and all other constraints. The determination of the optimal dismounting sequence for the fixtures is a highly sophisticated problem. Usually there are less platforms than fixtures available in the production cell. On this account the fixtures have to be dismounted on time. On the one hand, a deadlock can occur if the fixtures are not dismounted because then all platforms are occupied and no new fixture can be set up. On the other hand, needless time consuming steps are done if the fixtures are dismounted too early. A user defined sequence function called *platformsequence()* is used and implemented to solve this problem. The dismounting sequence of the fixtures is calculated depending on the current sequence of the products and the number of used platforms.

6.2.3 Results

In this case study a production cell is considered where 15 different products can be processed. The batch size for each product is defined from 2 to 4 resulting in 45 processed products and a Petri Net model consisting of 224×135 sized matrices.
150 iterations are performed for the Simulated Annealing and Threshold Accepting algorithm to optimise the production sequence. The implemented cooling strategies and neighbourhood functions are compared. For the Genetic algorithm three different population sizes are considered and 20 generations are derived. Selected parameter specifications are tested and the different implemented crossover functions are compared. Finally, the results for all algorithms are compared and shown.

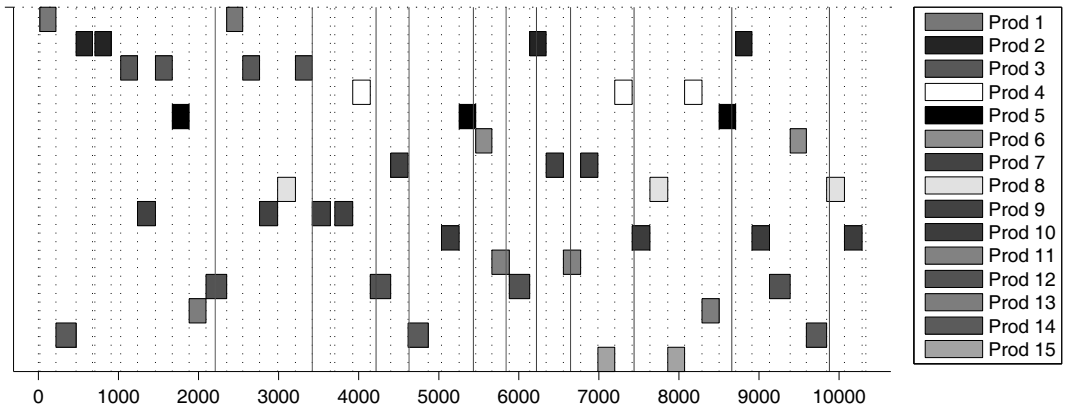


Figure 6.3: Gantt Chart of Initial Solution

Figure 6.3 shows the Gantt chart for the initial sequence of the products representing the initial solution of the optimisation problem. The coloured vertical lines stand for the desired finishing times of the selected products. In the initial sequence not all products are available on time and therefore additional gaps are created in the present Gantt chart.

Simulated Annealing

Figure 6.4 shows the results of the comparison of the different cooling strategies for the SA algorithm. In this case the exponential and linear cooling strategies lead to equal results after 150 iterations. The logarithmic strategy yields a worse output because this strategy allows an increase of the objective function in an advanced phase of the optimisation.

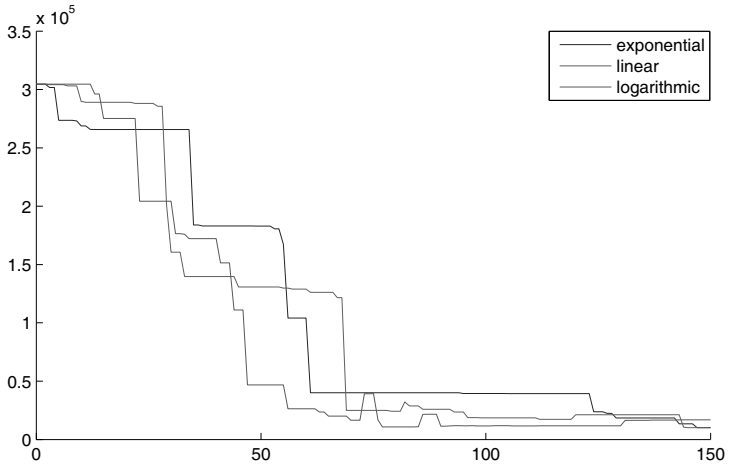


Figure 6.4: Comparison of Cooling Strategies

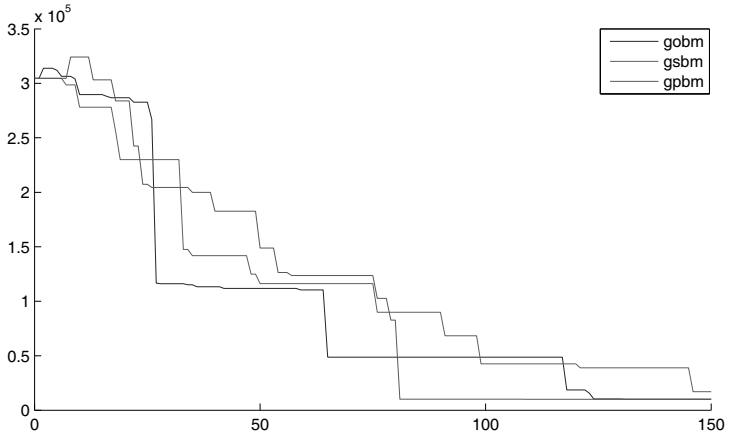


Figure 6.5: Comparison of Neighbourhood Functions - SA

Figure 6.5 shows the comparison of the neighbourhood functions. In this production cell the same products can occur more often. Therefore sequences with repetition have to be used for the optimisation. For this reason the generalised versions of the neighbourhood functions are compared. All three functions lead to nearly the same results and so they are suitable for the present optimisation problem in equal measure.

Threshold Accepting

Figure 6.6 and figure 6.7 show the comparison of the cooling strategies and neighbourhood functions for the Threshold Accepting algorithm. Again the exponential and linear strategies yield better results than the logarithmic strategy. In this case all neighbourhood functions are also similarly suitable for the optimisation.

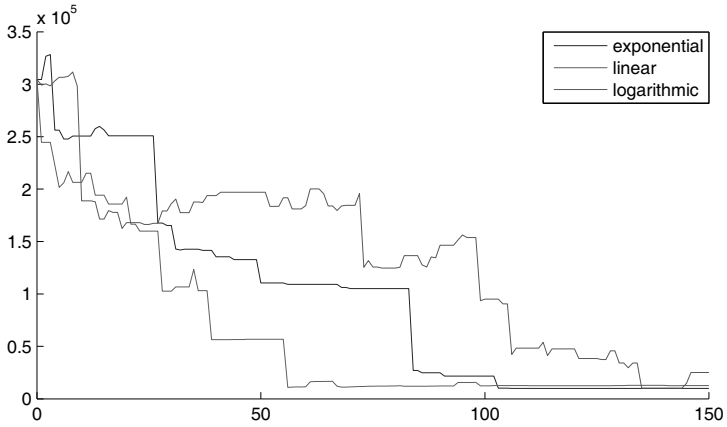


Figure 6.6: Comparison of Lowering Strategies

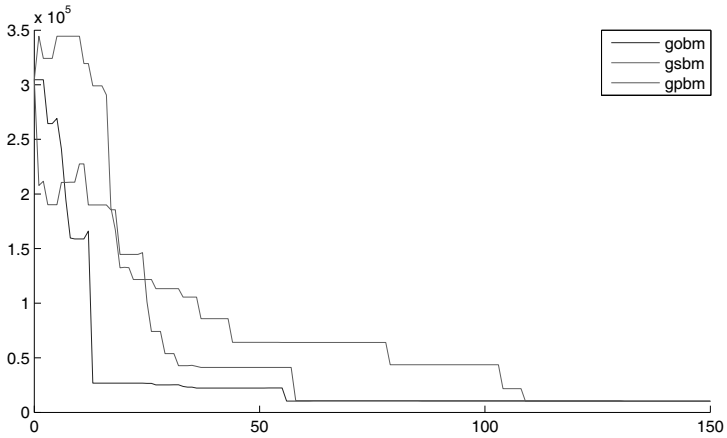


Figure 6.7: Comparison of Neighbourhood Functions - TA

Genetic Algorithm

Figure 6.8 shows the comparison of different population sizes for the Genetic algorithm. The population based on 100 individuals leads to the best results. This population contains the most different solutions and therefore the optimisation starts with a better initial best solution.

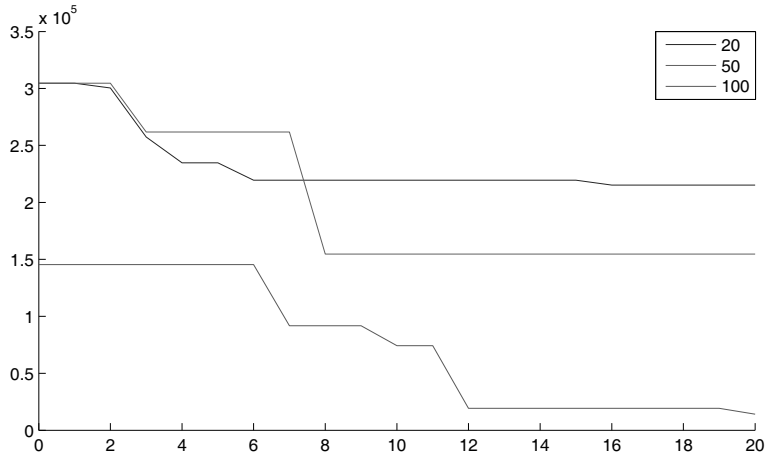


Figure 6.8: Comparison of Initial Populations

Figure 6.9 shows the comparison of different crossover functions for the GA. As mentioned before, the generalised versions of the crossover functions are used because the optimisation problem consists of sequences with repetition. In this case the GOX and GPX operators both yield the best results.

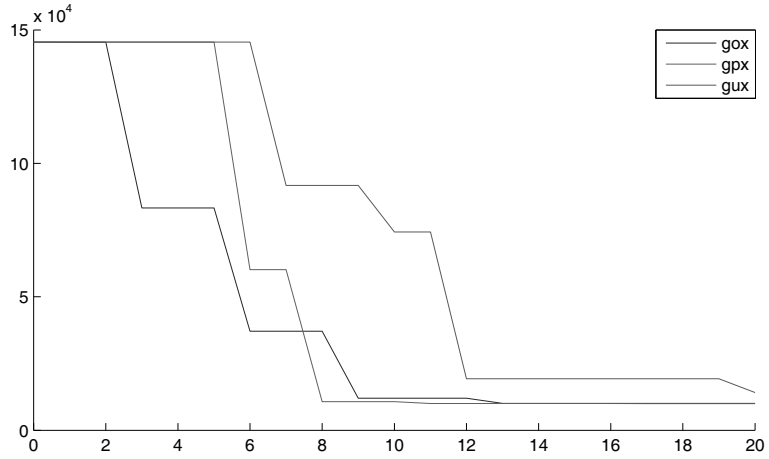


Figure 6.9: Comparison of Crossover Functions

Comparison

Figure 6.10 shows the results of the comparison for Simulated Annealing, Threshold Accepting and the Genetic Algorithm. All three heuristic algorithms lead to very good and similar results.

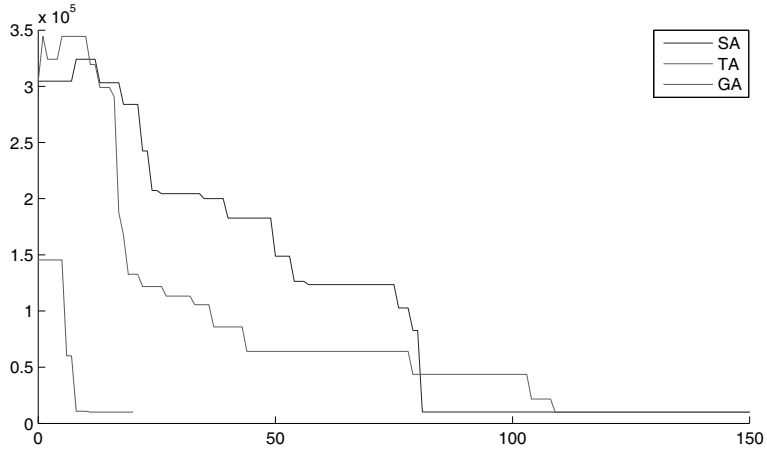


Figure 6.10: Comparison of SA, TA and GA

Table 6.1 contains the needed computational times for each optimisation algorithm and the calculated duration of the production. In this example the Genetic algorithm produced the best result but 2000 simulation runs are needed for 20 generations using a population of 100 individuals. Therefore the computational effort for GA amounts to about fourteen times as much as that for 150 iterations of the two other algorithms. The TA algorithm performs the optimisation in the fastest way.

	SA	TA	GA
Comp. Times	133.9	129.4	1775.4
Results	9976	9984	9949

Table 6.1: Comparison of Duration - SA, TA and GA

The Gantt chart of the optimised sequence representing the best solution of the optimisation problem is shown in figure 6.11. Now all products are available on time and therefore no additional gaps are created in the present Gantt chart. Furthermore, all products are finished on time and therefore no penalty times are added to the overall cycle time of the production.

Table 6.2 shows the additional penalty values for the initial and best solution. If the defined due dates are exceeded the difference of the real and desired finishing

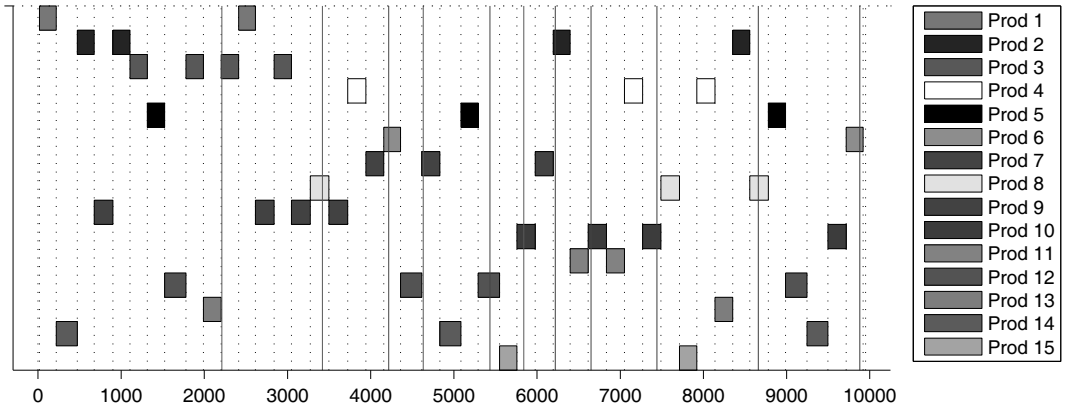


Figure 6.11: Gantt Chart of Best Solution

times is multiplied by the factor 100. Hence, the cumulative penalty time for the duration of the production amounts 2942 time units. The second results shown in the table 6.2 are the so called setup times for the initial and best solution. In this example the setup time means the time when no product is in the processing part of the production cell. The optimised solution brings in about 89 per cent improvement for the setup time.

	Initial	Best
Penalty	294200	0
Setup time	4.39 %	0.49 %

Table 6.2: Comparison of Penalty and Setup Times

6.3 Travelling Salesman Problem

6.3.1 Problem Formulation

The well-known Travelling Salesman Problem (TSP) asks for the shortest route to visit a collection of cities and return to the starting point. In this work the symmetric version of the TSP is treated. This means that, for any two cities A and B, the distance from A to B is the same as that from B to A. The costs for each connection are derived depending on the distance and the average speed of the salesman. In the deterministic case this speed is constant but also the stochastic case is considered where the average speed is modelled as stochastic distribution function.

6.3.2 Implementation

Many approaches and algorithms exist to model and to solve the Travelling Salesman Problem. Maybe Petri Nets are not really the best and fastest solution but the TSP can be easily modelled and optimised by the use of the MATLAB PetriSimM toolbox. Figure 6.12 shows the Petri Net model for the TSP which is automatically generated by the use of a programmed template. The function *tspsym()* creates all needed places, transitions and tokens whereas the following input parameters can be used: number of the cities, the initial sequence order of the visits, and the distance matrix.

Figure 6.12: Model of Travelling Salesman Problem

In this case only one colour is used to model the different cities. The symmetric TSP consists of $\frac{n^2-n}{2}$ different transitions representing all possible movements for n cities. The arising conflicts of the Petri Net are deactivated by the use of a sequence list. The input sequence contains only the city numbers and therefore a user defined sequence function (*tspsequence()*) is implemented to transform the input sequence to the corresponding sequence list for the transitions.

6.3.3 Results

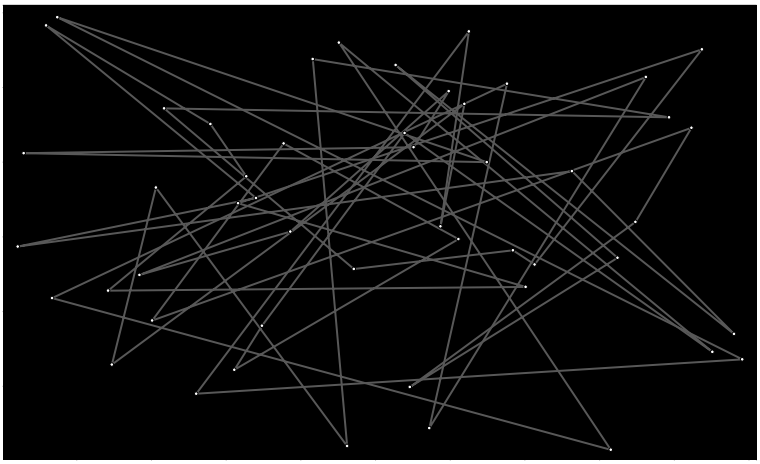


Figure 6.13: Plan of Initial Solution

In the deterministic case a TSP consisting of 50 cities is optimised. The corresponding Petri Net model is made up of 1 place and 1225 transitions. Figure 6.13 shows the plan of the cities and their connections representing the initial solution of the optimisation problem. In this context 1000 iterations are done for the SA and TA algorithm. Again, the neighbourhood functions are compared whereas the exponential cooling and lowering strategy is used. 100 generations are performed for the Genetic algorithm based on a population of 100 individuals to test and compare different parameter specification including the different crossover functions.

Simulated Annealing and Threshold Accepting

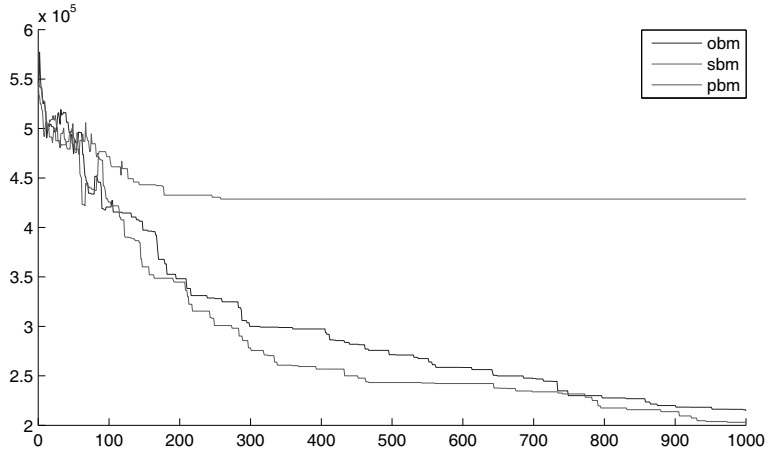


Figure 6.14: Comparison of Neighbourhood Functions - SA

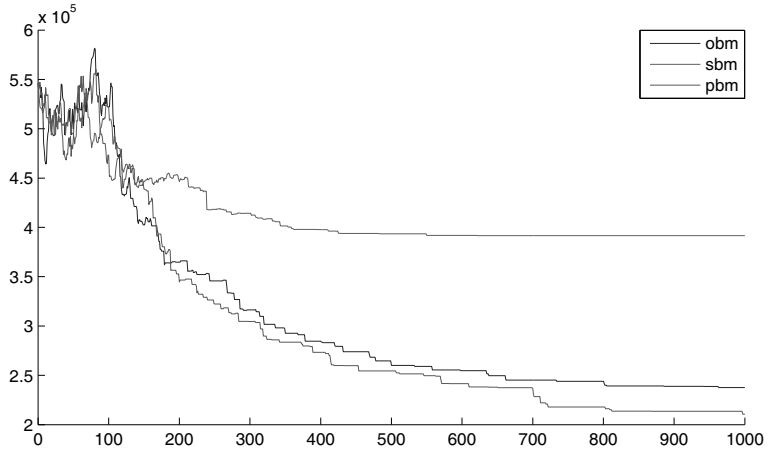


Figure 6.15: Comparison of Neighbourhood Functions - TA

Figure 6.14 and figure 6.15 show the comparison of the neighbourhood functions for the Simulated Annealing and Threshold Accepting algorithms. Each city is uniquely found in the sequence of the visits and therefore the normal and not generalised neighbourhood functions OBM, SBM and PBM are used. The PBM function leads to the best results in both cases. The progress of the optimisation based on the SBM function suggests the impracticalness of this neighbourhood function for the optimisation of the Travelling Salesman Problem.

Genetic Algorithm

Figure 6.16 shows the comparison of the crossover functions. As mentioned before the normal and not generalised crossover functions can be used for this problem. In this case the OX1 and OX2 crossover functions yield the best results whereas the OX2 operator is superior to the OX1 operator.

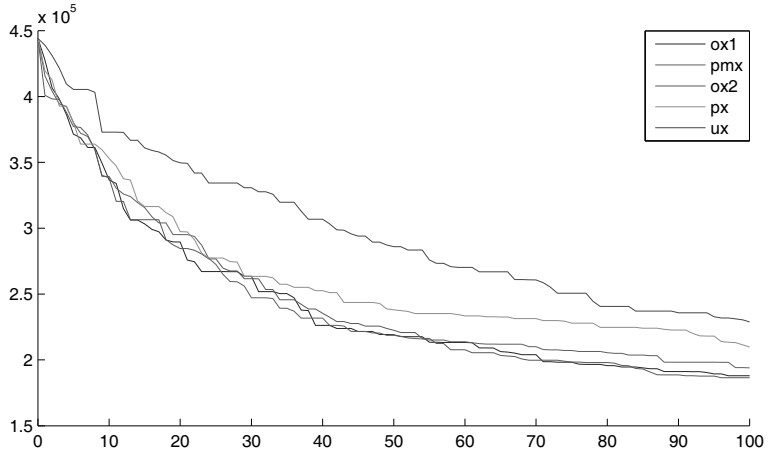


Figure 6.16: Comparison of Crossover Functions

Comparison

Figure 6.17 shows the results of the comparison for all three heuristic algorithms. The optimisation of the Travelling Salesman Problem leads to very good and similar achievements using Simulated Annealing, Threshold Accepting and the Genetic algorithm. Table 6.3 presents the needed computational times and the final duration of the round trip for all optimisation algorithms. In this example the Genetic algorithm leads to the best result but 10000 simulation runs are performed to realise 100 generations based on a population size of 100 individuals. Therefore the computational effort for GA amounts about ten times as much as for 1000 iterations of the two other optimisation algorithms. Again, the Threshold Accepting algorithm is the fastest method relating to computational speed .

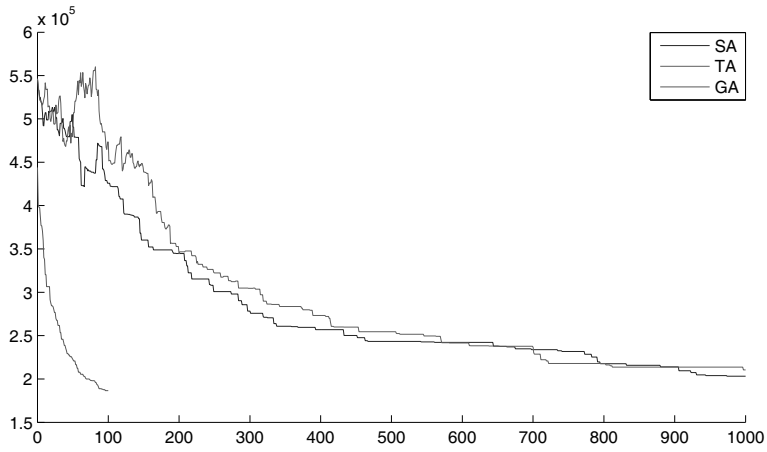


Figure 6.17: Comparison of SA, TA and GA

	SA	TA	GA
Comp. Time	92.25	86.17	1022.66
Results	203120	210500	186470

Table 6.3: Comparison of Duration - SA, TA and GA

Figure 6.18 shows the plan of the cities containing the best and optimised route of the present TSP. This result visualises the improvement of the optimisation compared to the initial solution of the problem.

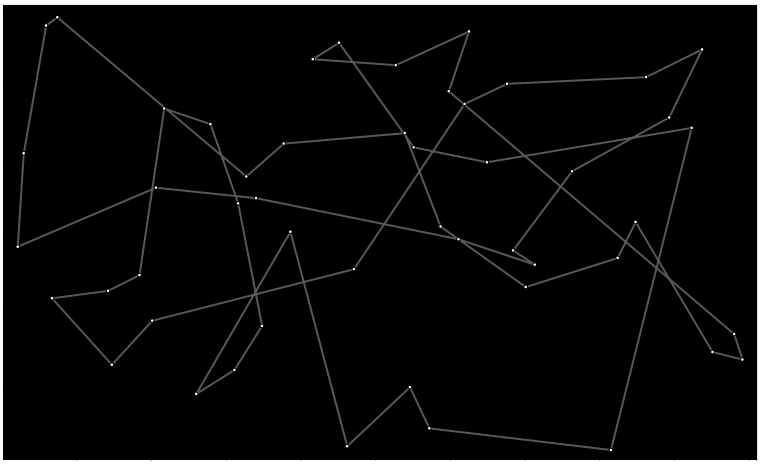


Figure 6.18: Plan of Best Solution

Stochastic

In the stochastic case the average speed of the salesman is modelled by the use of stochastic distribution functions. The stochastic optimisation is only possible for Simulated Annealing and Threshold Accepting algorithms because two different system configurations are always compared. Basically a certain number of simulation runs are needed to get significant results. In this example 20 cities are taken into account to realise the Travelling Salesman problem for stochastic time delays. 150 iterations are done for each optimisation algorithm. Figure 6.19 shows the plan of the cities and their connections representing the initial solution of the problem.

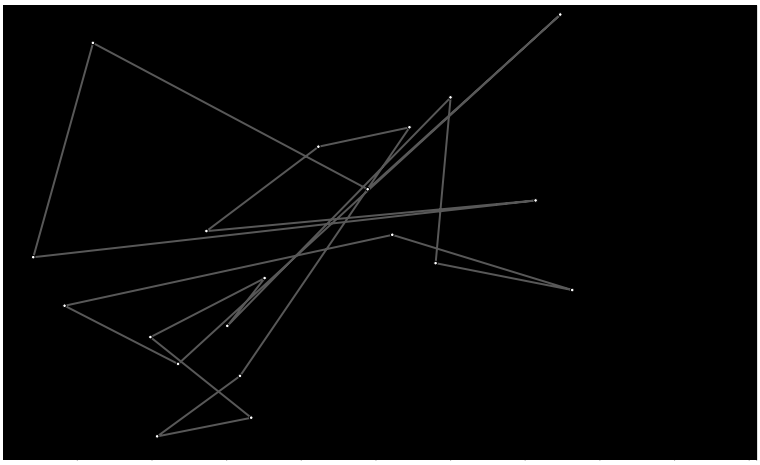


Figure 6.19: Plan of Initial Solution

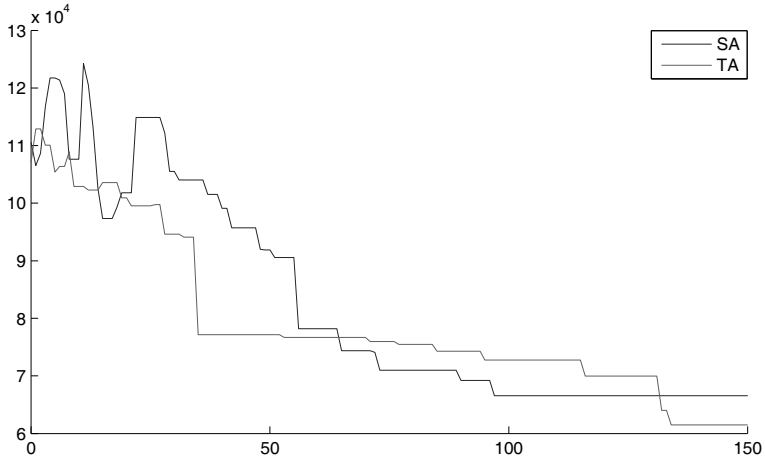


Figure 6.20: Comparison of SA and TA

Figure 6.20 shows the comparison of Simulated Annealing and Threshold Accepting. After 150 iterations the TA algorithm leads to better results.

Table 6.4 contains the comparison of the duration for both algorithms and presents the time improvement achieved by the use of variance reduction. It can be seen that the variance reduction highly reduces the computational effort for the stochastic optimisation. In both cases about 70 percent of time can be saved.

Duration	Normal	VR	Difference	%
TA	6555.7	1416.1	5139.6	-78.40%
SA	2543.1	898.3	1644.8	-64.68%

Table 6.4: Time Improvement of Variance Reduction

The decrease of the computational effort accompanies the reduction of needed simulation runs. Table 6.5 shows the difference for both optimisation algorithms. The use of variance reduction techniques decrease the number of needed simulation runs in an essential way. About 80 percent of runs can be saved in both cases.

Runs	Normal	VR	Difference	%
TA	138598	23834	114764	-82.80%
SA	53318	14857	38461	-72.14%

Table 6.5: Reduction of needed Simulation Runs

Figure 6.21 shows the plan of the cities containing the optimised route of the TSP.

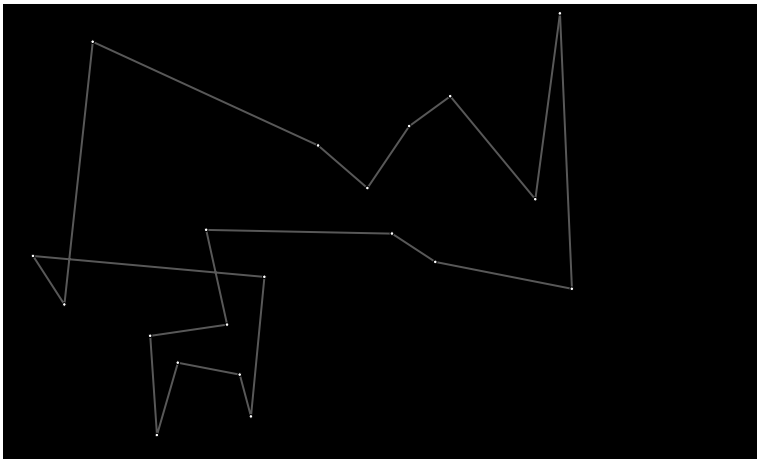


Figure 6.21: Plan of Best Solution

Chapter 7

Summary and Outlook

This work provides a general framework for modelling, simulation and optimisation of scheduling problems based on Petri Nets. The basic definitions of Petri Nets are extended with the capability of time delays to enable the simulation over the time domain. Two different approaches of adding time to Petri Nets are implemented. The holding durations and enabling durations principles provide a suitable basis for modelling time dependent problems. Coloured Petri Nets offer an easier graphical description for building more complex models. Stochastic Petri Nets introduce the use of stochastic time delays realised by stochastic distribution functions. Petri Nets are extended with the capability of modelling scheduling problems whereas arbitrary firing sequences can be defined. If all conflicts are solved beforehand the scheduling problem is modelled by the use of the basic properties of Petri Nets.
All features and functionalities are developed and implemented in the open source MATLAB PetriSimM toolbox which is embedded in the powerful MATLAB environment. The toolbox is suitable for educational purposes as well as for modelling, simulation, and optimisation of real life processes. Several results can be shown and all produced data can be used for internal or external post-processing. Only two steps have to be done by the user of the toolbox. The first one is the development of the Petri Net model, which is building all needed conditions and constraints of the present scheduling problem. The second step is the choice of the best optimisation method and the correct and optimal specification of the needed parameters for the otpimisation algorithm. Three different heuristic methods are implemented and can be selected. Simulated Annealing, Threshold Accepting, and Genetic algorithms are forming the choice for realising the optimisation of scheduling problems. Depending on the present problem each optimisation algorithm has its advantages and disadvantages. No general proof can be done to decide which algorithm fits the best for all problems. Many optimisation studies and runs have to be processed to get significant results because in this case randomness plays a certain role. Furthermore,

the search for the optimal and best suited parameters is a highly sophisticated problem and mainly depends on the present problem specification. The heuristics are implemented to form and offer a wide spread basis for the optimisation of scheduling problems. All methods can be easily extended with further functionalities and functions and new algorithms can also be implemented to the open source toolbox.
The implemented methods and functionalities are tested in two case studies. The optimisation leads to good and similar results for all three algorithms and no significant difference can be determined. Threshold Accepting is the fastest algorithm because of the simpler definition. No randomness is needed to check and to decide if a worse solution is accepted or not. Genetic algorithms are the most extensive method because the computational effort of each generation depends on the population size. The stochastic optimisation is time-consuming because in this case the number of needed simulation runs is modelled by a random number. If the difference of two alternative system configurations is nearby zero, basically a lot of simulation runs are needed to get a significant decision. The use of variance reduction shows a high decrease of computational time because many simulation runs can be saved for each comparison.
The different benchmarks point out the main limitation of Petri Nets. Due to the fact that Petri Nets model processes on a very low level the underlying matrices can get very big for complex problems. A high amount of computational speed is needed for checking the number of enabled transitions and for processing and resolving the occurring conflicts. The increase of the power of the used computer system would increase the speed of the simulation. The speed of one simulation run highly influences the duration of the optimisation because basically a lot of simulation runs are needed to optimise a certain scheduling problem. Future work can be the improvement of the used algorithms with respect to the computational speed. Furthermore the number of enabling checks could be reduced to gain speed during the simulation.
Genetic algorithms can be a very useful heuristic method for many different scheduling problems because the optimisation is based on a wide spread pool of possible solutions. On this account Genetic algorithms take more simulation runs and computational effort compared to Simulated Annealing and Threshold Accepting. In future work a parallelisation can be implemented to get rid of this disadvantage. The calculation and evaluation of the particular individuals of the population could be split into several network computers in each generation. This concept of parallelisation can also be used for the stochastic optimisation to improve the needed computational time for getting significant results. Further investigations could be the extension and graphical integration of the analysis capabilities and the continuative development of the automated creation of Petri Net models based on templates or based on input and output matrices.

Bibliography

[1] Carl Adam Petri: *Kommunikation mit Automaten.* Dissertation, Universität Bonn, 1962.

[2] F. Bause and P. Kritzinger: *Stochastic Petri Nets - An Introduction to the Theory (2nd edition).* Vieweg Verlag Germany, 2002, ISBN 3-528-15535-3.

[3] Wolgang Reisig: *A Primer in Petri Net Design.* Springer-Verlag Berlin Heidelberg New York, 1992, ISBN 3-540-52044-9.

[4] Wolfgang Reisig: *Lectures on Petri Nets I: Basic Models.* Springer-Verlag Berlin Heidelberg, 1998, ISBN 3-540-65306-6.

[5] Wolgang Reisig: *Petri Nets An Introduction.* Springer-Verlag Berlin Heidelberg New York, 1985, ISBN 3-540-13723-8.

[6] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis: *Modelling with Generalized Stochastic Petri Nets.* Wiley Series in Parallel Computing, John Wiley and Sons, 1994, ISBN 0-471-93059-8.

[7] F.D.J. Bowden: *A brief survey and synthesis of the roles of time in petri nets.* Mathematical and Computer Modelling, 31:55–68, 2000.

[8] W.M. Zuberek: *Timed Petri Nets. Definitions,Properties and Applications.* Microelectronics and Reliability, 31(4):627–644, 1991.

[9] P. M. Merlin and D. J. Farber: *Recoverability of Communication Protocols: Implications of a Theoretical Study.* IEEE Trans. Comm., 24(9):1036–1043, September 1976.

[10] Kurt Jensen: *High-level Petri Nets.* Springer-Verlag Berlin Heidelberg, 1991, ISBN 3-540-54125-X.

[11] Kurt Jensen: *Coloured Petri Nets.* Springer-Verlag Berlin Heidelberg New York, 1992, ISBN 3-540-55597-8.

[12] Peter J. Haas: *Stochastic Petri Nets. Modelling, Stability, Simulation.* Springer Verlag New York, 2002, ISBN 0-387-95445-7.

[13] J. B. Dugan, K. S. Trivedi, R. M. Geist, and V. F. Nicola: *Extended Stochastic Petri Nets: Applications and Analysis.* Performance '84, Models of Computer System Performance, Proc. of the 10th Int. Symp. 1984, Paris, France, 507-519.

[14] P. Langrock und W. Jahn: *Einführung in die Theorie der Markovschen Ketten und ihre Anwendungen.* BSB B. G. Teubner Verlagsgesellschaft, Leipzig, 1979.

[15] Oliver Cantoni: *Statistical Learning Theory and Stochastic Optimization.* Springer-Verlag Berlin Heidelberg, 2004, ISBN 3-540-22572-2.

[16] James J. O'Brian: *Scheduling Handbook.* McGraw-Hill Book Company, 1969, ISBN 07-047601-2.

[17] Tadeusz Sawik: *Production Planning and Scheduling in Flexible Assembly Systems.* Springer-Verlag Berlin Heidelberg, 1999, ISBN 3-540-64998-0.

[18] Joze Balic, Yannis A. Phillis, Nikos Tsourveloudis, and Ivo Pahole: *Flexibility in Manufacturing - Models and Measurement.* University of Maribour, Technical University of Crete, 2002, ISBN 86-435-0510-2.

[19] Peter Brucker: *Scheduling Algorithms.* Springer-Verlag Berlin Heidelberg, 2001, ISBN 3-540-41510-6.

[20] Simon French: *Sequencing and Scheduling.* Ellis Horwood Limited, 1982, ISBN 0-13-806365-6.

[21] B. Griffer and G. L. Thompson: *Algorithms for Solving Production-Scheduling Problems.* Operations Research, 8(4):487–503, 1960.

[22] Christian Kelling: *Simulationsverfahren für zeiterweiterte Petri-Netze.* Dissertation, Technische Universität Berlin, 1995.

[23] Konrad Zuse: *Anwendungen von Petri-Netzen.* Vieweg Verlag, 1982, ISBN 3-528-09616-0.

[24] Wolfgang Reisig: *Lectures on Petri Nets II: Applications.* Springer-Verlag Berlin Heidelberg, 1998, ISBN 3-540-65307-4.

[25] Dejan Gradišar and Gašper Mušič: *Simulation of Production Systems Based on Timed Petri Nets and Matlab.* Proc. SAUM 2004 Conf., Belgrade, Serbia, 267-272.

[26] W. M. Zuberek: *Timed Petri Nets in Modeling and Analysis of Simple Schedules for Manufacturing Cells.* Computers and Mathematics with Applications, 37:191–206, 1999.

[27] András Jávor: *AI controlled high level Petri nets in simulating FMS.* AI, Simulation, and Planning in High Autonomy Systems, 1993. Integrating Virtual Reality and Model-Based Environments. Proceedings. Fourth Annual Conference, Tucson, Arizona, 302-308.

[28] András Jávor: *Comparison 2 – CASSANDRA 3.0.* Simulation News Europe, Issue 4, July 1994, 32.

[29] Dirk C. Mattfeld: *Evolutionary Search and the Job Shop.* Physica-Verlag Heidelberg, 1996, ISBN 3-7908-0917-9.

[30] René V. V. Vidal (editor): *Applied Simulated Annealing.* Springer-Verlag Berlin Heidelberg, 1993, ISBN 3-540-56229-X.

[31] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller: *Equation of State Calculations by Fast Computing Machines.* Journal of Chemical Physics, 21:1087–1092, 1953.

[32] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi: *Optimization by Simulated Annealing.* Science, 220(4598):671–680, 1983.

[33] Nourani Y. and Andresen B.: *A Comparison of Simulated Annealing Cooling Strategies.* Journal of Physics A: Mathematical and General, 31(41):8373–8385, 1998.

[34] Gunter Dueck and Tobias Scheuer: *Threshold Accepting: A General Purpose Optimisation Algorithm Appearing Superior to Simulated Annealing.* Journal of Computational Physics, 90:161–175, 1990.

[35] John H. Holland: *Adaptation in natural and artificial systems.* The MIT Press, 1992, ISBN 0-262-08213-6.

[36] David E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addision-Wesley Publishing Comp., Inc., 1989, ISBN 0-201-15767-5.

[37] David A. Coley: *An Introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publishing Co. Pte. Ltd., 1999, ISBN 981-02-3602-6.

[38] Lawrence Davis (editor): *Handbook of Genetic Algorithms.* International Thomsen Computer Press, 1996, ISBN 1-85032-825-0.

[39] Mituso Gen, Yasuhiro Tsujimura, and Erika Kubota: *Solving Job-Shop Scheduling Problems by Genetic Algorithm.* Proc. 16th International Conference on Computers and Industrial Engineering, Ashikaga, Japan, 1994, pp. 576-579.

[40] D. Goldberg and R. Lingle: *Alleles, Loci and the Travelling Salesman.* Proc. of the First International Conference on Genetic Algorithms and their Applications, San Matteo, Italy, 1985.

[41] The MathWorks Inc.: *MATLAB Help - Documentation for MathWorks Products.* http://www.mathworks.com/access/helpdesk/help/helpdesk.html, 2006.

[42] Christian Bierwirth: *A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms.* OR Spektrum, 17:87–92, 1995.

[43] Averill M. Law and W. David Kelton: *Simulation Modelling and Analysis.* McGraw-Hill, Inc., 1991, ISBN 0-07-036698-5.

[44] Paul Brately, Bennet L. Fox, and Linus E. Schrage: *A Guide to Simulation.* Springer-Verlag Berlin Heidelberg, 1987, ISBN 3-540-96467-3.

[45] The MathWorks Inc.: *MATLAB Connections – Third Party Products and Services.* http://www.mathworks.com/products/connections, 2006.

[46] M. H. Matcovschi, C. Mahulea, and O. Pastravanu: *Petri Net Toolbox for MATLAB.* Proc. MED 2003 Conf., Rhodes, Greece.

[47] TGI group: *Petri Nets World.* University of Hamburg, Germany, 2006, http://www.informatik.uni-hamburg.de/TGI/PetriNets/.

[48] McLeod Institute of Simulation Sciences Hungarian Center: *CASSANDRA 3.0.* http://itm.bme.hu/mcleod/cassandra.html, 2002.

[49] Gašper Mušič, Borut Zupančič, and Drago Matko: *Petri Net Based Modelling and Supervisory Control Design in Matlab.* Proc. EUROCON 2003 Conf., Ljubljana, Slovenia, 362-366.

[50] Thomas Löscher, Felix Breitenecker, and Gašper Mušič: *Petri Net Modelling and Simulation in Matlab – A Petri Net Toolbox.* Simulation News Europe, Issue 43, July 2005, 20-21.

[51] Thomas Löscher, Felix Breitenecker, Gašper Mušič, and Dejan Gradišar: *A Matlab-based Tool for Timed Petri Nets.* Proc. ERK 2005 Conf., Portorož, Slovenia, 273-276.

[52] Thomas Löscher, Dejan Gradišar, Felix Breitenecker, and Gašper Mušič: *Timed Petri Net Simulation in Matlab: A Production Cell Case Study.* Proc. MATHMOD 2006 Conf., Vienna, Austria.

[53] Thomas Löscher and Felix Breitenecker: *Petri Net Modelling and Simulation of Production Processes with PetriSimM, a MATLAB-based Toolbox.* Proc. 12. ASIM - Fachtagung Simulation in Produktion und Logistik 2006, Kassel, Germany, 313 - 319.

[54] Gašper Mušič, Thomas Löscher, and Dejan Gradišar: *An Open Petri Net Modelling and Analysis Environment in Matlab.* Proc. IMM 2006 Conf., Barcelona, Spain, 123-128.

[55] Thomas Löscher: *Simulationsbasierte Optimierung zur Verbesserung der Produktionsabläufe in einer flexiblen Fertigunszelle.* Diplomarbeit, Technische Universität Wien, 2004.

[56] Thomas Löscher, Markus Klug, and Felix Breitenecker: *Simulation-based Optimization of Production Plans for a Production Cell using Heuristic Methods - Comparison of Tabu Search, Simulated Annealing and Threshold Accepting.* Proc. EUROSIM 2004 Conf., Paris, France.

[57] E. L. Lawler, Jan Karel Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys: *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley & Sons Ltd, New York, 1985, ISBN 0-471-90413-9.

Curriculum Vitae

Persönliche Daten

Name:	Dipl.-Ing. Thomas Löscher
Adresse:	Sandweg 13
	2070 Retz
Geburtstag:	8. Dezember 1979
Geburtsort:	Horn

Schulbildung

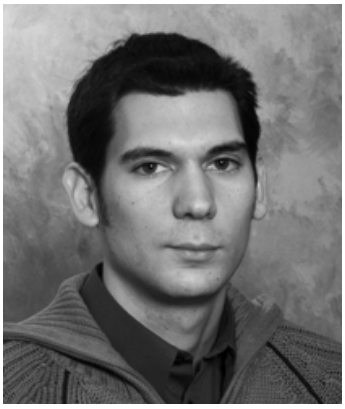
1986 bis 1990	Volksschule in Retz
1990 bis 1998	Bundesrealgymnasium in Hollabrunn
Juni 1998	Matura

Präsenzdienst

Oktober 1998 bis Mai 1999	Radetzkykaserne Horn

Studium

Oktober 1999	Immatrikulation an der Technischen Universität Wien Mathematische Computerwissenschaften (Stzw) aus Technische Mathematik
März 2004	Abschluss der Diplomarbeit mit dem Thema: "Simulationsbasierete Optimierung zur Verbesserung der Produktionsabläufe in einer flexiblen Fertigungszelle"
27. April 2004	Abschluss des Diplomstudiums
seit Mai 2004	Doktoratsstudium der techn. Wissenschaften
Feb. 2005 bis Juni 2005	Auslandsstudium an der Universität Ljubljana, Slowenien



About the author ...

Thomas Löscher studied Technical Mathematics at the University of Technology of Vienna. During his diploma study he specialised in modelling, simulation and optimisation of discrete event systems. In his PhD study he continued his research in the field of simulation-based optimisation. He studied one semester abroad based on cooperation with the faculty of electrical engineering of the university Ljubljana, Slovenia. There he found the basis for his PhD thesis where he optimised scheduling problems based on Timed Petri nets. The results of his work are published in this volume. During his PhD study he also worked for the ARC Seibersdorf research company in the field of discrete event simulation. Currently he is employed as research associate at the Profactor Company in the business domain Process and System Intelligence.

About this volume ...

This volume deals with modelling and simulation of scheduling and sequencing problems based on Petri Nets. In particular, Timed, Coloured, and Stochastic Petri Nets are used to model and implement specific scheduling problems in the field of production processes and other discrete event systems. The Petri Net models are simulated over the time domain and a simulation-based optimisation is implemented to optimise the input sequences. In case of stochastic processes the comparison of alternative system configurations is a highly sophisticated problem. A sequential paired t-test and variance reduction techniques are used and implemented to solve the stochastic optimisation for sequencing and scheduling problems. In the course of this work the PetriSimM toolbox is developed and embedded in the MATLAB environment.

Über diese Reihe …

Die Bände der ASIM - Reihe ***Fortschrittsberichte Simulation*** zeigen neueste Lösungsansätze, Methoden und Anwendungen der Simulation in Theorie und Praxis. Die Reihe umspannt Grundlagen und Anwendung der Simulation in einem immer breiter werdenden Spektrum, z. B. Ingenieurwissenschaften, Naturwissenschaften, Medizin, Ökonomie, Ökologie und Soziologie.
Fortschrittsberichte Simulation konzentrieren sich auf Monographien mit speziellem Charakter, wie z. B. Dissertationen und Habilitationen, Berichte zu ASIM-Workshops mit referierten Beiträgen, Berichte zu Forschungsprojekten, Handbücher zu Simulationswerkzeugen, Benchmarks, und ähnliches.
ASIM - Arbeitsgemeinschaft **SIM**ulation - die deutschsprachige Simulationsver-einigung, zugleich Fachausschuss der GI (Gesellschaft für Informatik), hat diese Reihe im ARGESIM / ASIM - Verlag als Ergänzung und Nachfolge zur ASIM- Reihe ***Fortschritte in der Simulationstechnik - Frontiers in Simulation*** ins Leben gerufen, um ein rasches und kostengünstiges Publikationsmedium für neue Entwicklungen in der Simulationstechnik anbieten zu können.